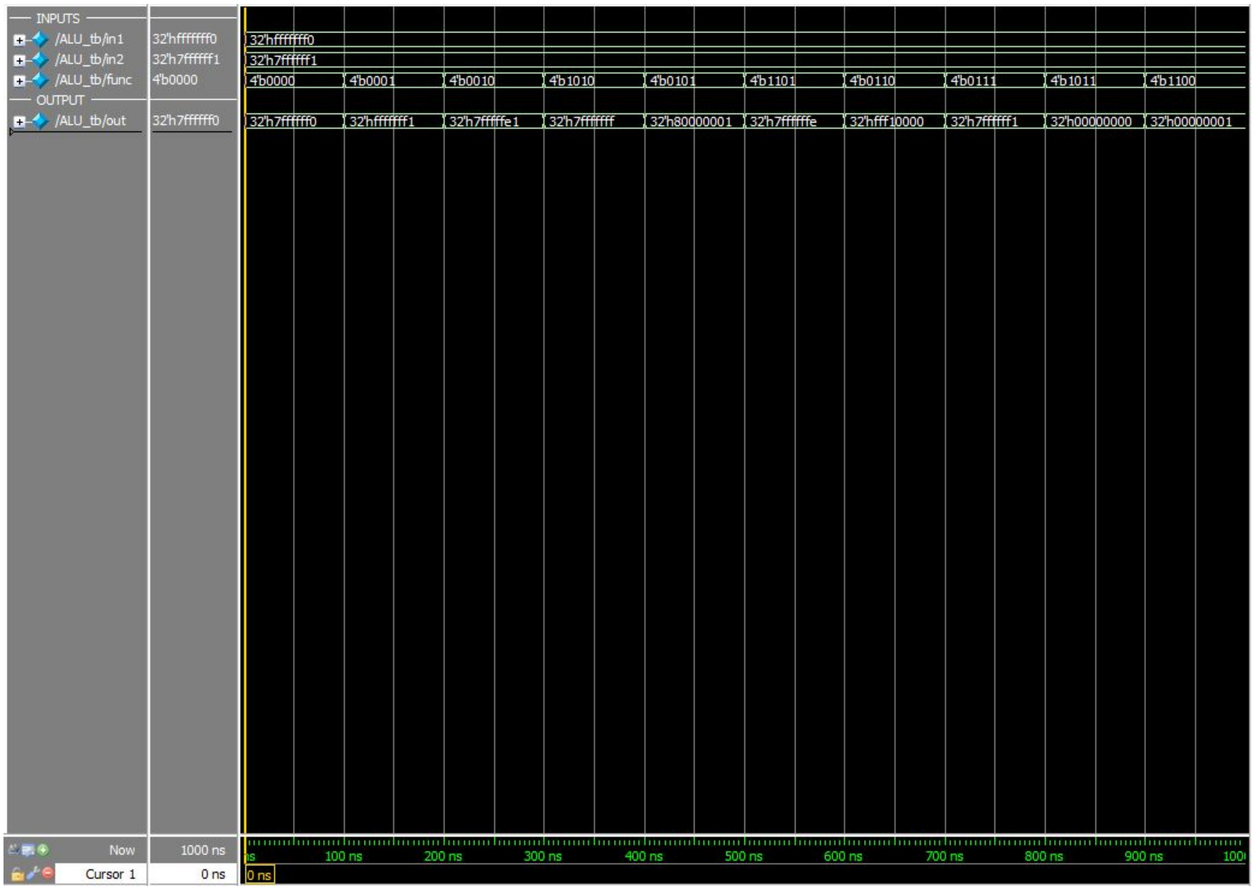
Introduction

The idea of a pipeline processor is to utilize more parts of the processor at the same time. The full operation can be split into multiple stages such that the same operations done multiple times can be overlapped together and executed simultaneously. This design is broken into 5 stages: fetch, decode, execute, memory, and writeback. One of the main problems with pipelined registers is that dependent instructions that are not finished executing results in hazards. The hazard unit detects and resolves the hazards.

ALU

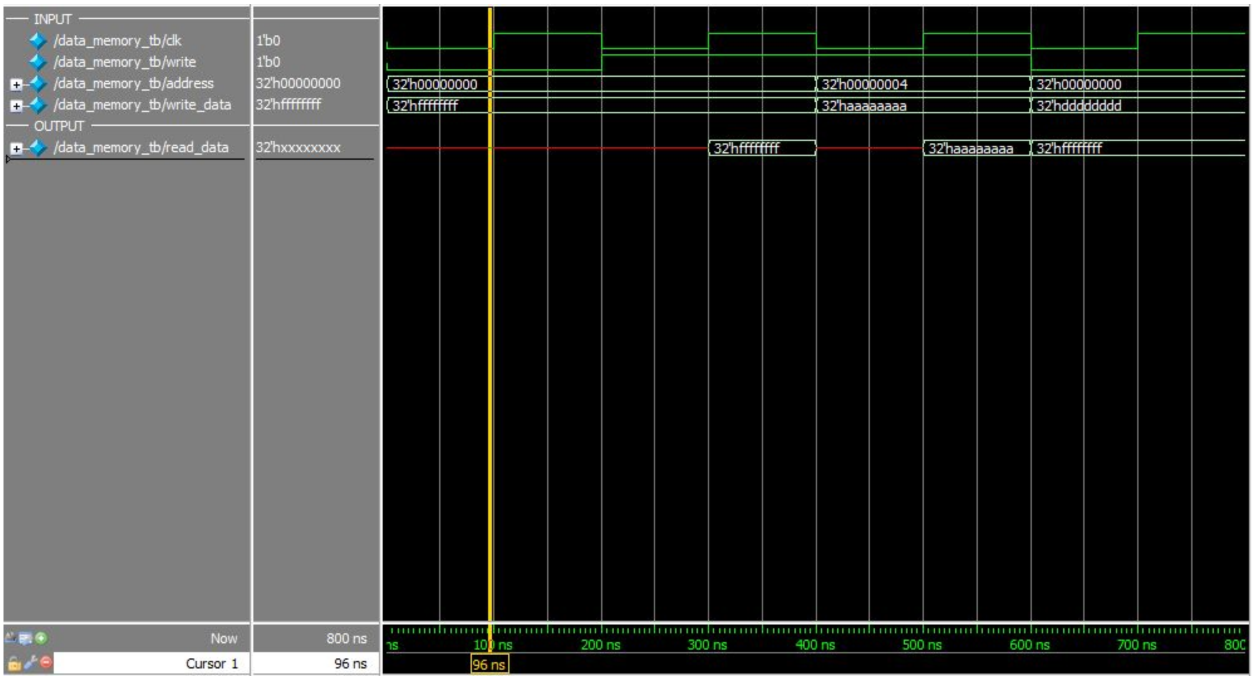
The ALU's basic operations are add, sub, and, or, slt, lui, xor and in2 for mfhi and mflo. The last bit in ALUop is designed to negate in2 most notably for sub and xor. Unsigned operations are the same representation wise except for slt. An extra operation configuration is used for sltu to check the sign bits.



ALU_tb waveform

Data Memory

The data memory holds 64 slots of 32 bit words. Each word is byte addressable, but indexing in Verilog is every integer, so the address is shifted to the right twice to align with byte addressing. Writing to the memory is on the positive edge and the enable signal needs to be on.

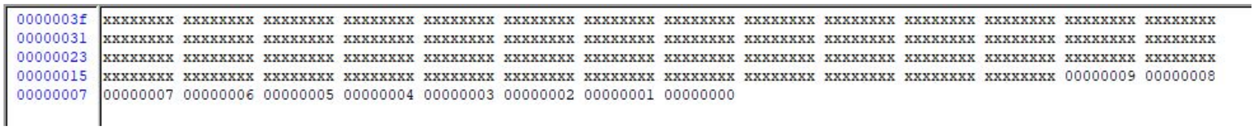


```
INPUT
  /data_memory_tb/clk          1'b0
  /data_memory_tb/write        1'b0
  /data_memory_tb/address      32'h00000000
  /data_memory_tb/write_data   32'hffffffff
OUTPUT
  /data_memory_tb/read_data    32'hxxxxxxxx
```

data_memory_tb waveform

Instruction Memory

The instruction memory can hold 64 32 bit instructions. Addressing into the instruction memory is the same as data memory. Initializing instructions into the memory uses memfile.dat. The figure shows the first 10 memory slots initialized with constants from 0-9.

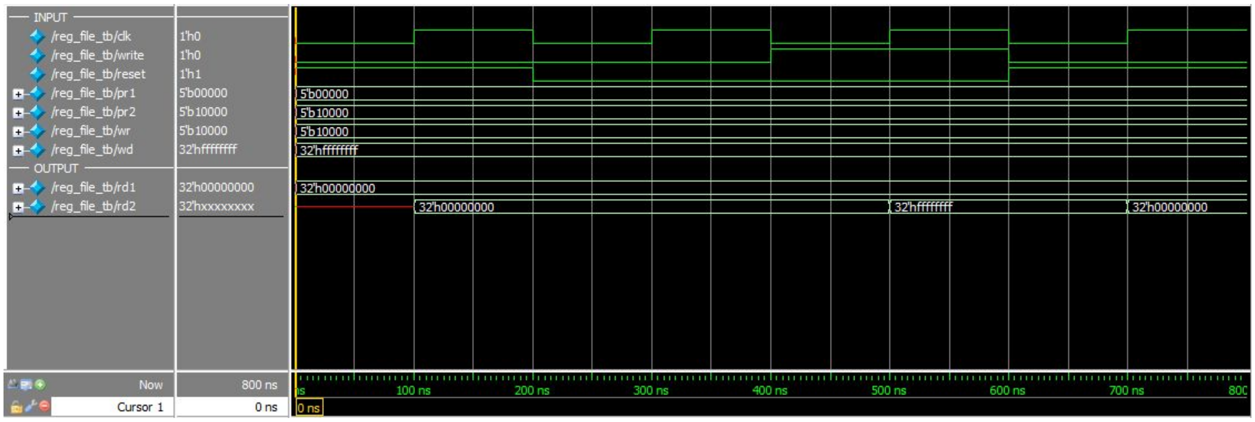


```
0000003f xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000031 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000023 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000015 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx 00000009 00000008
00000007 00000007 00000006 00000005 00000004 00000003 00000002 00000001 00000000
```

Instruction Memory

Regfile

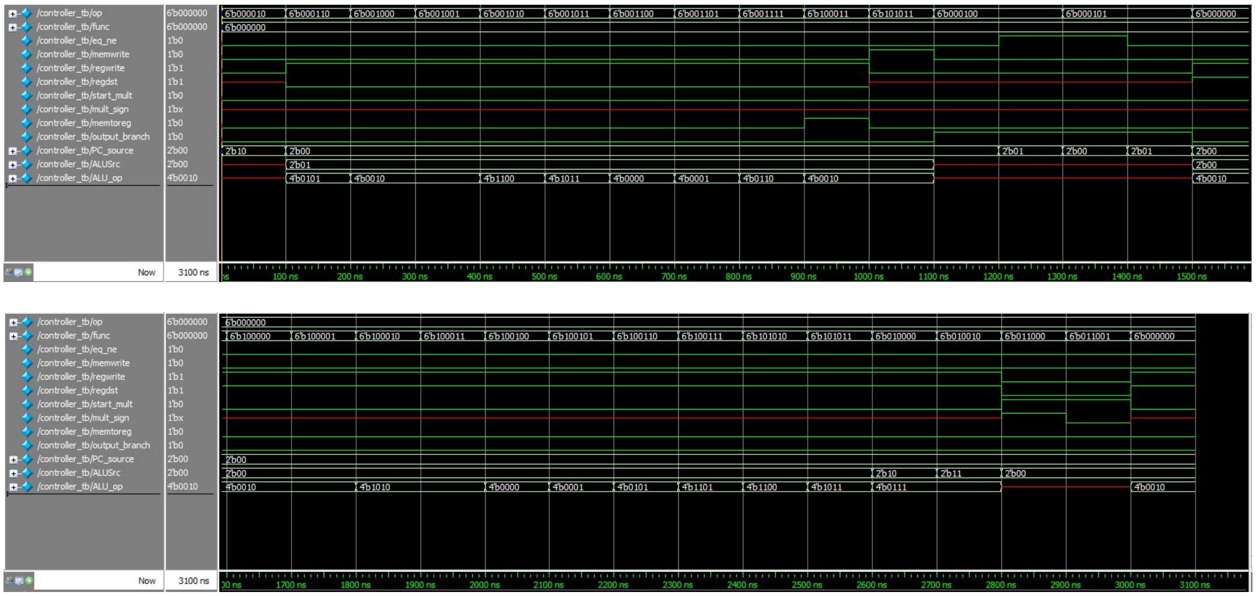
The register file contains 32 32 bit registers. On reset, all the registers are reset to 0. Writing on the positive edge requires an enable signal. Two read ports are asynchronous and should update if data is written into that register in the same cycle.



reg_file_tb

Controller

The controller outputs all the enable signals, MUX selects, and ALUOp. Op is used first, then funct if the instruction is an R type. Eq_ne is from the equality check for early branch prediction. The default output for all 0s instructions are essentially nops.

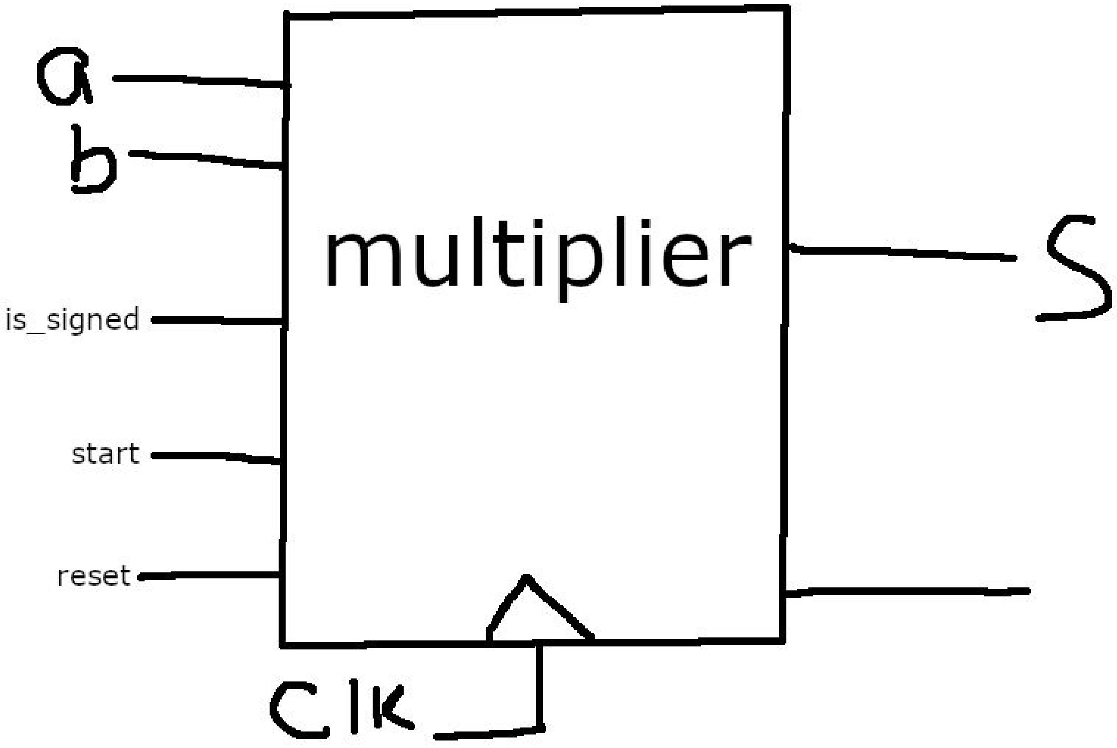


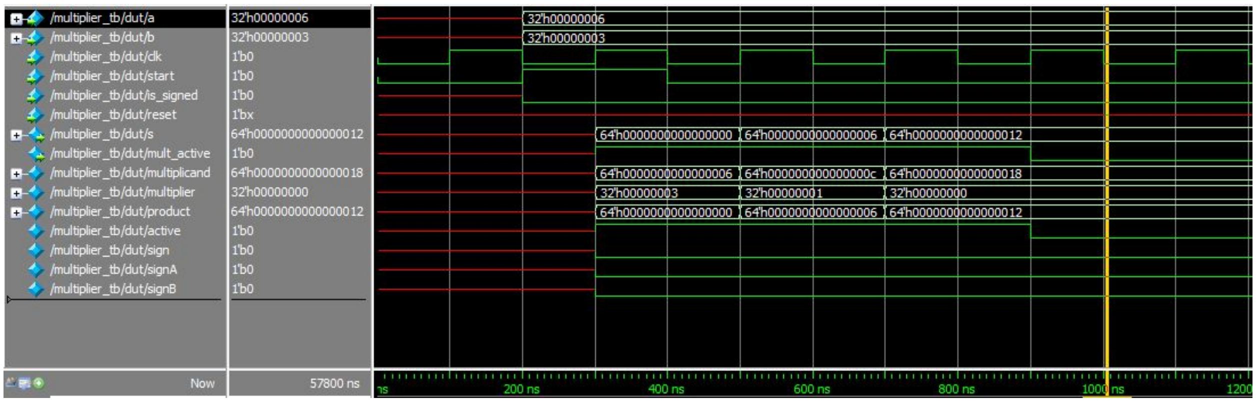
/controller_tb/op 6'b000000
/controller_tb/func 6'b000000
/controller_tb/eq_ne 1'b0
/controller_tb/memwrite 1'b0
/controller_tb/regwrite 1'b1
/controller_tb/regdst 1'b1
/controller_tb/start_mult 1'b0
/controller_tb/mult_sign 1'bx
/controller_tb/memtoreg 1'b0
/controller_tb/output_branch 1'b0
/controller_tb/PC_source 2'b00
/controller_tb/ALUSrc 2'b00
/controller_tb/ALU_op 4'b0010
6'b000010 6'b000110 6'b001000 6'b001001 6'b001010 6'b001011 6'b001100 6'b001101 6'b001111 6'b100011 6'b101011 6'b000100 6'b000101 6'b000000
6'b000000
2'b10 2'b00 2'b01 2'b00 2'b01 2'b00
2'b01 2'b00
4'b0101 4'b0010 4'b1100 4'b1011 4'b0000 4'b0001 4'b0110 4'b0010 4'b0010
Now 3100 ns
100 ns 200 ns 300 ns 400 ns 500 ns 600 ns 700 ns 800 ns 900 ns 1000 ns 1100 ns 1200 ns 1300 ns 1400 ns 1500 ns
6'b000000
6'b100000 6'b100001 6'b100010 6'b100011 6'b100100 6'b100101 6'b100110 6'b100111 6'b101010 6'b101011 6'b010000 6'b010010 6'b011000 6'b011001 6'b000000
2'b00
2'b00 2'b10 2'b11 2'b00
4'b0010 4'b1010 4'b0000 4'b0001 4'b0101 4'b1101 4'b1100 4'b1011 4'b0111 4'b0010
Now 3100 ns
1700 ns 1800 ns 1900 ns 2000 ns 2100 ns 2200 ns 2300 ns 2400 ns 2500 ns 2600 ns 2700 ns 2800 ns 2900 ns 3000 ns 3100 ns

controller_tb waveform

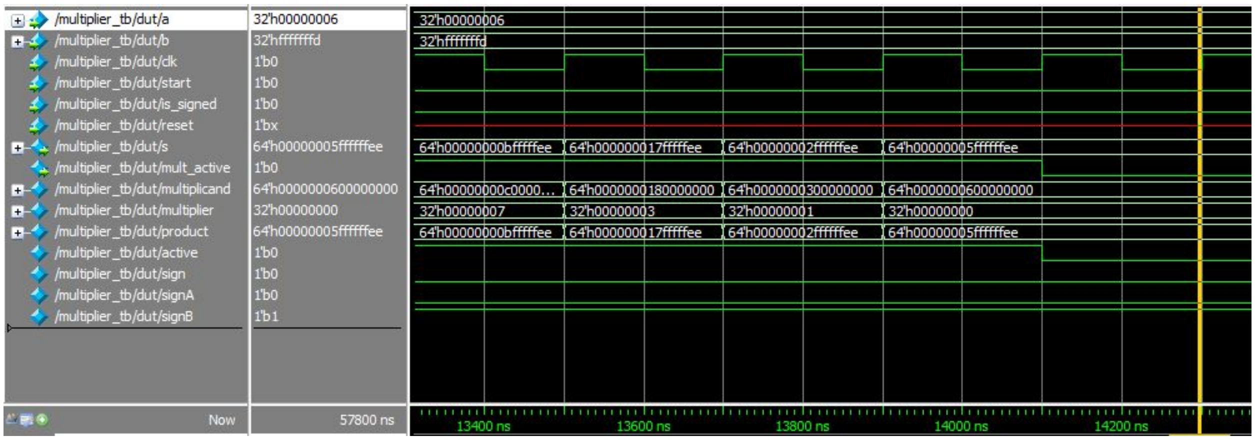
Multiplier

A serial multiplier takes the two inputs and adds up their partial sums. The multiplicand is shifted to the left and added into the product. The multiplier is shifted to the right and the LSB is checked if the addition is needed. For signed multiplication, the unsigned multiplication is performed first, then the sign is added if needed. An active signal is outputted for the processor to stall until the multiplication is finished. The serial multiplier finishes as soon as the multiplier is shifted to all 0s. The product can be 64 bits long, so it is stored in its own location. To move the value to the regfile, mfhi and mflo will move the high 32 bits and low 32 bits respectively. The first for figures are unsigned and the last four are signed.

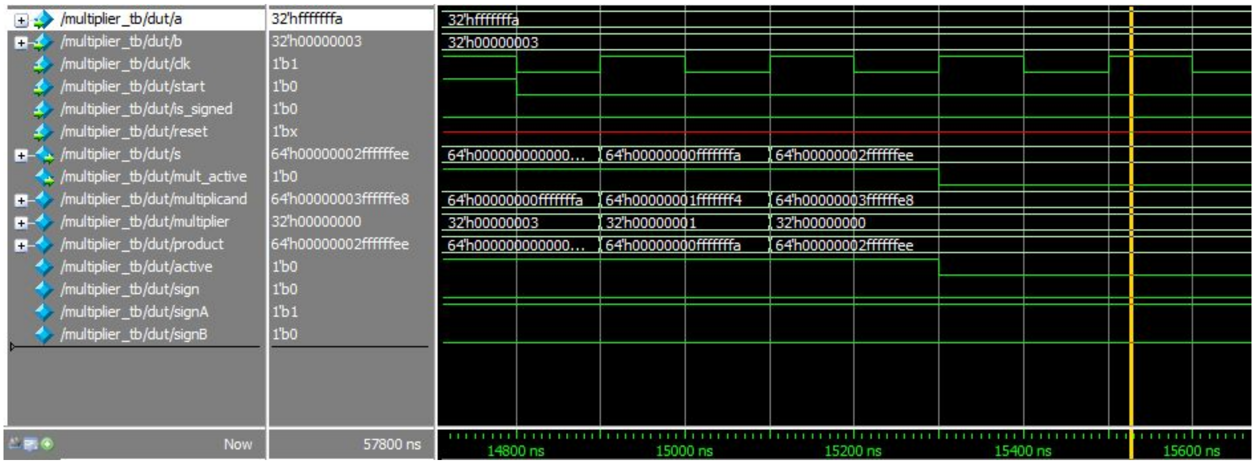




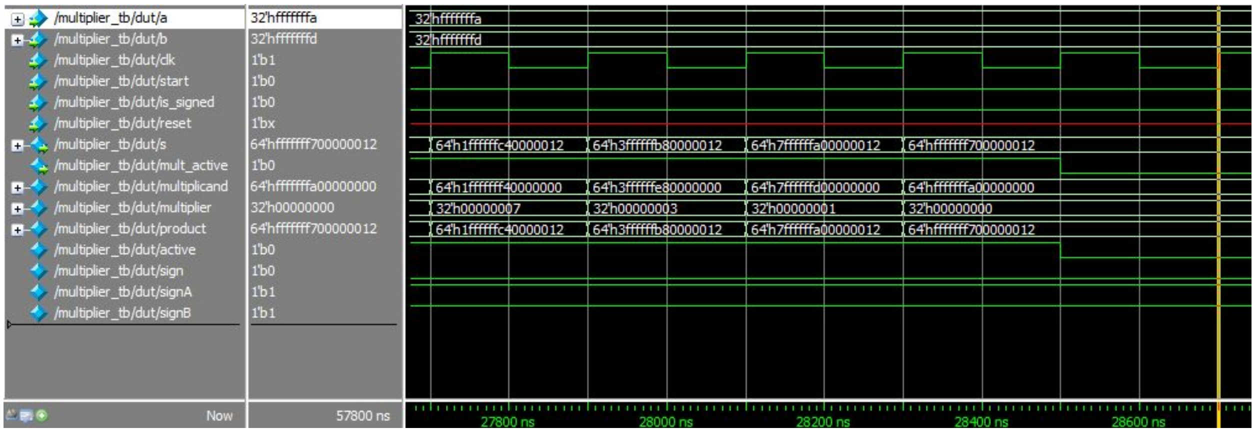
/multiplier_tb/dut/a	32'h00000006
/multiplier_tb/dut/b	32'h00000003
/multiplier_tb/dut/clk	1'b0
/multiplier_tb/dut/start	1'b0
/multiplier_tb/dut/is_signed	1'b0
/multiplier_tb/dut/reset	1'bx
/multiplier_tb/dut/s	64'h0000000000000012
/multiplier_tb/dut/mult_active	1'b0
/multiplier_tb/dut/multiplicand	64'h0000000000000018
/multiplier_tb/dut/multiplier	32'h00000000
/multiplier_tb/dut/product	64'h0000000000000012
/multiplier_tb/dut/active	1'b0
/multiplier_tb/dut/sign	1'b0
/multiplier_tb/dut/signA	1'b0
/multiplier_tb/dut/signB	1'b0
Now	57800 ns



/multiplier_tb/dut/a	32'h00000006
/multiplier_tb/dut/b	32'hfffffffd
/multiplier_tb/dut/clk	1'b0
/multiplier_tb/dut/start	1'b0
/multiplier_tb/dut/is_signed	1'b0
/multiplier_tb/dut/reset	1'bx
/multiplier_tb/dut/s	64'h00000005ffffffee
/multiplier_tb/dut/mult_active	1'b0
/multiplier_tb/dut/multiplicand	64'h0000000600000000
/multiplier_tb/dut/multiplier	32'h00000000
/multiplier_tb/dut/product	64'h00000005ffffffee
/multiplier_tb/dut/active	1'b0
/multiplier_tb/dut/sign	1'b0
/multiplier_tb/dut/signA	1'b0
/multiplier_tb/dut/signB	1'b1
Now	57800 ns

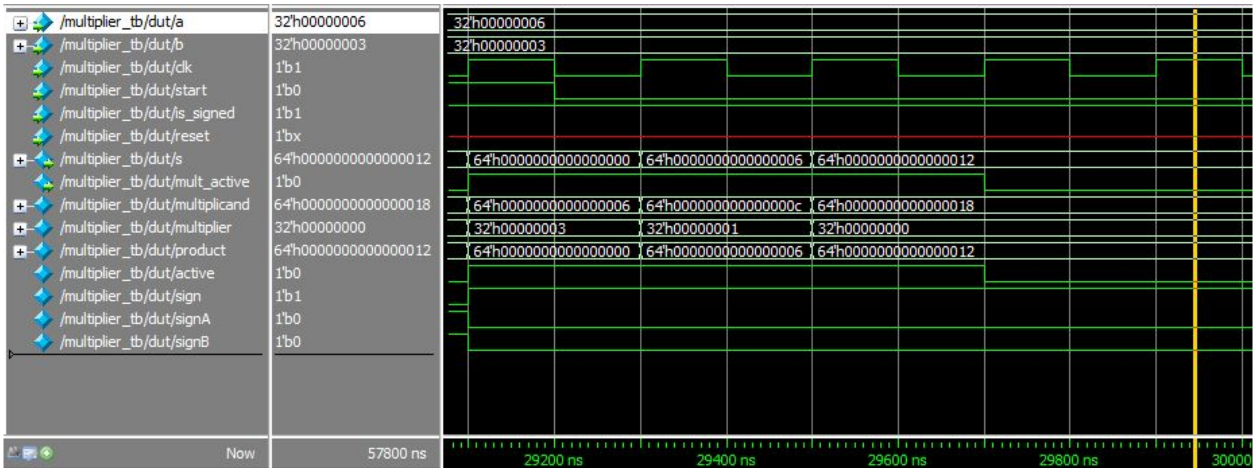


/multiplier_tb/dut/a	32'hfffffffa
/multiplier_tb/dut/b	32'h00000003
/multiplier_tb/dut/clk	1'b1
/multiplier_tb/dut/start	1'b0
/multiplier_tb/dut/is_signed	1'b0
/multiplier_tb/dut/reset	1'bx
/multiplier_tb/dut/s	64'h00000002ffffffee
/multiplier_tb/dut/mult_active	1'b0
/multiplier_tb/dut/multiplicand	64'h00000003ffffffe8
/multiplier_tb/dut/multiplier	32'h00000000
/multiplier_tb/dut/product	64'h00000002ffffffee
/multiplier_tb/dut/active	1'b0
/multiplier_tb/dut/sign	1'b0
/multiplier_tb/dut/signA	1'b1
/multiplier_tb/dut/signB	1'b0
Now	57800 ns

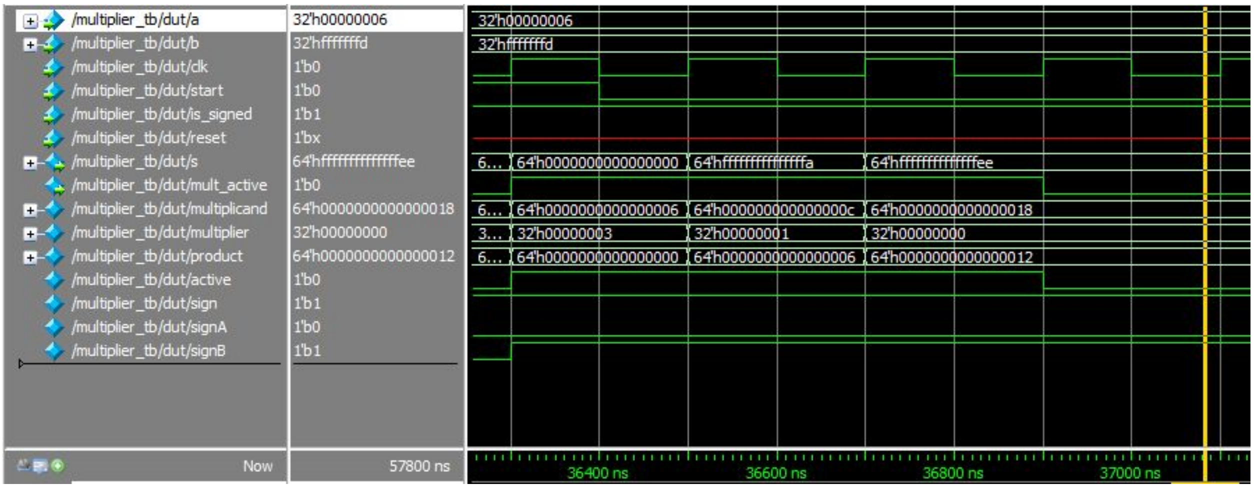


```
/multiplier_tb/dut/a             32'hfffffffa
/multiplier_tb/dut/b             32'hfffffffd
/multiplier_tb/dut/clk           1'b1
/multiplier_tb/dut/start         1'b0
/multiplier_tb/dut/is_signed     1'b0
/multiplier_tb/dut/reset         1'bx
/multiplier_tb/dut/s             64'hfffffff700000012
/multiplier_tb/dut/mult_active   1'b0
/multiplier_tb/dut/multiplicand  64'hfffffffa00000000
/multiplier_tb/dut/multiplier    32'h00000000
/multiplier_tb/dut/product       64'hfffffff700000012
/multiplier_tb/dut/active        1'b0
/multiplier_tb/dut/sign          1'b0
/multiplier_tb/dut/signA         1'b1
/multiplier_tb/dut/signB         1'b1
Now                              57800 ns
```

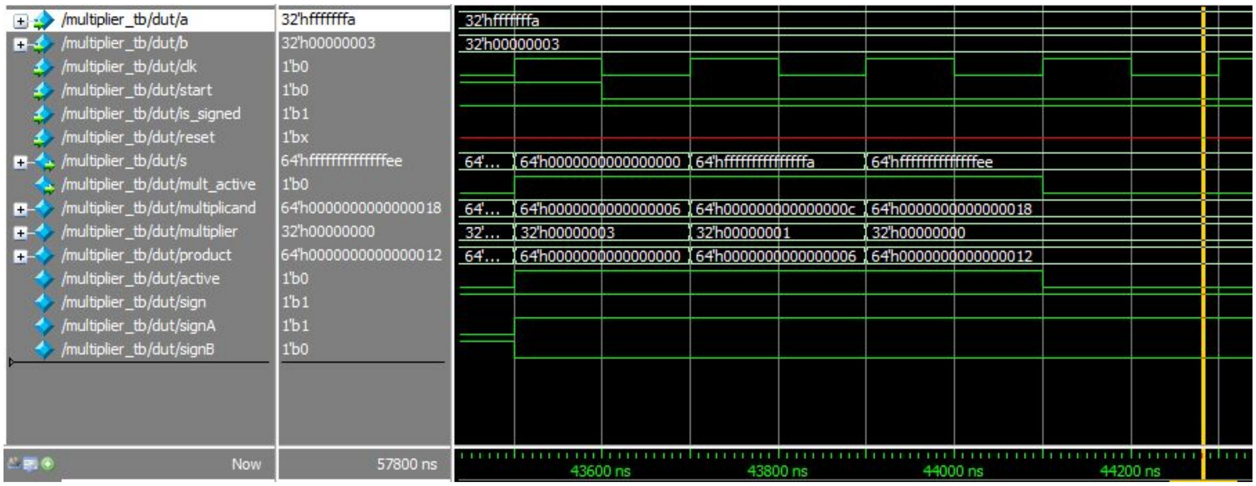
multu waveforms



```
/multiplier_tb/dut/a             32'h00000006
/multiplier_tb/dut/b             32'h00000003
/multiplier_tb/dut/clk           1'b1
/multiplier_tb/dut/start         1'b0
/multiplier_tb/dut/is_signed     1'b1
/multiplier_tb/dut/reset         1'bx
/multiplier_tb/dut/s             64'h0000000000000012
/multiplier_tb/dut/mult_active   1'b0
/multiplier_tb/dut/multiplicand  64'h0000000000000018
/multiplier_tb/dut/multiplier    32'h00000000
/multiplier_tb/dut/product       64'h0000000000000012
/multiplier_tb/dut/active        1'b0
/multiplier_tb/dut/sign          1'b1
/multiplier_tb/dut/signA         1'b0
/multiplier_tb/dut/signB         1'b0
Now                              57800 ns
```

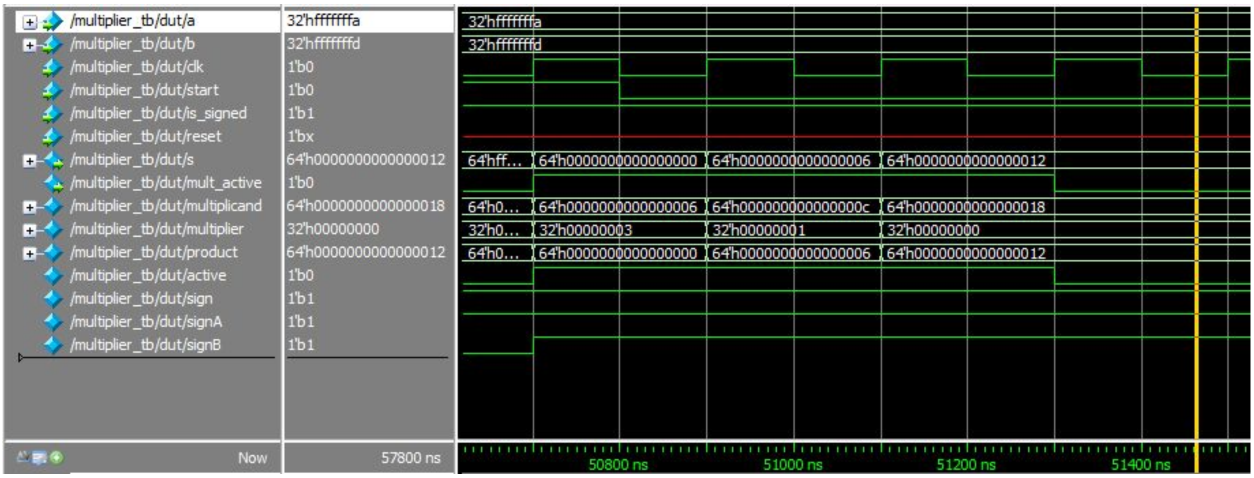


```
/multiplier_tb/dut/a             32'h00000006
/multiplier_tb/dut/b             32'hfffffffd
/multiplier_tb/dut/clk           1'b0
/multiplier_tb/dut/start         1'b0
/multiplier_tb/dut/is_signed     1'b1
/multiplier_tb/dut/reset         1'bx
/multiplier_tb/dut/s             64'hffffffffffffffee
/multiplier_tb/dut/mult_active   1'b0
/multiplier_tb/dut/multiplicand  64'h0000000000000018
/multiplier_tb/dut/multiplier    32'h00000000
/multiplier_tb/dut/product       64'h0000000000000012
/multiplier_tb/dut/active        1'b0
/multiplier_tb/dut/sign          1'b1
/multiplier_tb/dut/signA         1'b0
/multiplier_tb/dut/signB         1'b1
Now                              57800 ns
```



/multiplier_tb/dut/a	32'hfffffffa
/multiplier_tb/dut/b	32'h00000003
/multiplier_tb/dut/clk	1'b0
/multiplier_tb/dut/start	1'b0
/multiplier_tb/dut/is_signed	1'b1
/multiplier_tb/dut/reset	1'bx
/multiplier_tb/dut/s	64'hffffffffffffffee
/multiplier_tb/dut/mult_active	1'b0
/multiplier_tb/dut/multiplicand	64'h0000000000000018
/multiplier_tb/dut/multiplier	32'h00000000
/multiplier_tb/dut/product	64'h0000000000000012
/multiplier_tb/dut/active	1'b0
/multiplier_tb/dut/sign	1'b1
/multiplier_tb/dut/signA	1'b1
/multiplier_tb/dut/signB	1'b0
Now	57800 ns

32'hfffffffa
32'h00000003
64'... 64'h0000000000000000 64'hfffffffffffffffa 64'hffffffffffffffee
64'... 64'h0000000000000006 64'h000000000000000c 64'h0000000000000018
32'... 32'h00000003 32'h00000001 32'h00000000
64'... 64'h0000000000000000 64'h0000000000000006 64'h0000000000000012
43600 ns
43800 ns
44000 ns
44200 ns



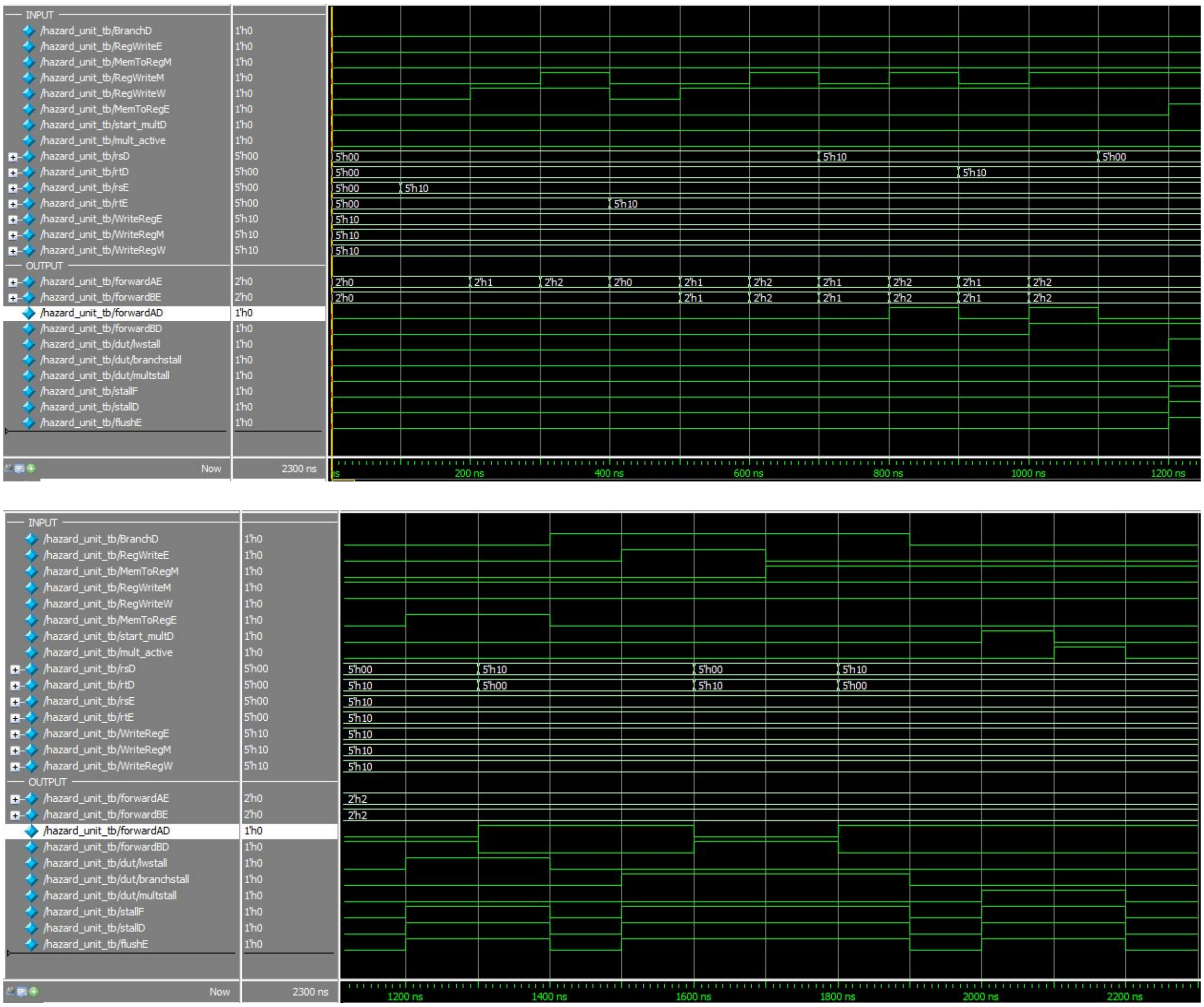
/multiplier_tb/dut/a	32'hfffffffa
/multiplier_tb/dut/b	32'hfffffffd
/multiplier_tb/dut/clk	1'b0
/multiplier_tb/dut/start	1'b0
/multiplier_tb/dut/is_signed	1'b1
/multiplier_tb/dut/reset	1'bx
/multiplier_tb/dut/s	64'h0000000000000012
/multiplier_tb/dut/mult_active	1'b0
/multiplier_tb/dut/multiplicand	64'h0000000000000018
/multiplier_tb/dut/multiplier	32'h00000000
/multiplier_tb/dut/product	64'h0000000000000012
/multiplier_tb/dut/active	1'b0
/multiplier_tb/dut/sign	1'b1
/multiplier_tb/dut/signA	1'b1
/multiplier_tb/dut/signB	1'b1
Now	57800 ns

32'hfffffffa
32'hfffffffd
64'hff... 64'h0000000000000000 64'h0000000000000006 64'h0000000000000012
64'h0... 64'h0000000000000006 64'h000000000000000c 64'h0000000000000018
32'h0... 32'h00000003 32'h00000001 32'h00000000
64'h0... 64'h0000000000000000 64'h0000000000000006 64'h0000000000000012
50800 ns
51000 ns
51200 ns
51400 ns

mult waveform

Hazard Unit

If the subsequent instruction is dependent on the previous registers, but the value has not been written back yet (stage 5), then the data needs to be forwarded to the correct stage. If a value is needed from memory right after a lw instruction, the processor is forced to stall until the data is read from memory. With early branch prediction, the processor knows when to take the branch at the decode stage, but this causes another data hazard. The processor needs to stall, then forward the data to the decode stage if needed. The fetch and decode registers are stalled and execute registers are flushed to prevent unwanted instructions from being executed. The first figure shows data forwarding and the second shows all types of stalling.



```
INPUT
/hazard_unit_tb/BranchD        1'h0
/hazard_unit_tb/RegWriteE      1'h0
/hazard_unit_tb/MemToRegM      1'h0
/hazard_unit_tb/RegWriteM      1'h0
/hazard_unit_tb/RegWriteW      1'h0
/hazard_unit_tb/MemToRegE      1'h0
/hazard_unit_tb/start_multD    1'h0
/hazard_unit_tb/mult_active    1'h0
/hazard_unit_tb/rsD            5'h00
/hazard_unit_tb/rtD            5'h00
/hazard_unit_tb/rsE            5'h00
/hazard_unit_tb/rtE            5'h00
/hazard_unit_tb/WriteRegE      5'h10
/hazard_unit_tb/WriteRegM      5'h10
/hazard_unit_tb/WriteRegW      5'h10
OUTPUT
/hazard_unit_tb/forwardAE      2'h0
/hazard_unit_tb/forwardBE      2'h0
/hazard_unit_tb/forwardAD      1'h0
/hazard_unit_tb/forwardBD      1'h0
/hazard_unit_tb/dut/lwstall    1'h0
/hazard_unit_tb/dut/branchstall 1'h0
/hazard_unit_tb/dut/multstall  1'h0
/hazard_unit_tb/stallF         1'h0
/hazard_unit_tb/stallD         1'h0
/hazard_unit_tb/flushE         1'h0
Now                            2300 ns
```

```
INPUT
/hazard_unit_tb/BranchD        1'h0
/hazard_unit_tb/RegWriteE      1'h0
/hazard_unit_tb/MemToRegM      1'h0
/hazard_unit_tb/RegWriteM      1'h0
/hazard_unit_tb/RegWriteW      1'h0
/hazard_unit_tb/MemToRegE      1'h0
/hazard_unit_tb/start_multD    1'h0
/hazard_unit_tb/mult_active    1'h0
/hazard_unit_tb/rsD            5'h00
/hazard_unit_tb/rtD            5'h00
/hazard_unit_tb/rsE            5'h00
/hazard_unit_tb/rtE            5'h00
/hazard_unit_tb/WriteRegE      5'h10
/hazard_unit_tb/WriteRegM      5'h10
/hazard_unit_tb/WriteRegW      5'h10
OUTPUT
/hazard_unit_tb/forwardAE      2'h0
/hazard_unit_tb/forwardBE      2'h0
/hazard_unit_tb/forwardAD      1'h0
/hazard_unit_tb/forwardBD      1'h0
/hazard_unit_tb/dut/lwstall    1'h0
/hazard_unit_tb/dut/branchstall 1'h0
/hazard_unit_tb/dut/multstall  1'h0
/hazard_unit_tb/stallF         1'h0
/hazard_unit_tb/stallD         1'h0
/hazard_unit_tb/flushE         1'h0
Now                            2300 ns
```

hazard_unit_tb waveform

Datapath

Some notes on datapath:

ALU Src has 2 bits to take into account mfhi and mflo as inputs. Connecting writing ports of regfile to writeback registers instead of the wires at the end of the memory stage delays the writing by one cycle. The controller in theory has don't care terms as output, but it is safer to set a known value to those control signals because it messes up values for future cycles like stall signals for example. On reset, all the register files are set to all 0s. This is essentially a nop.

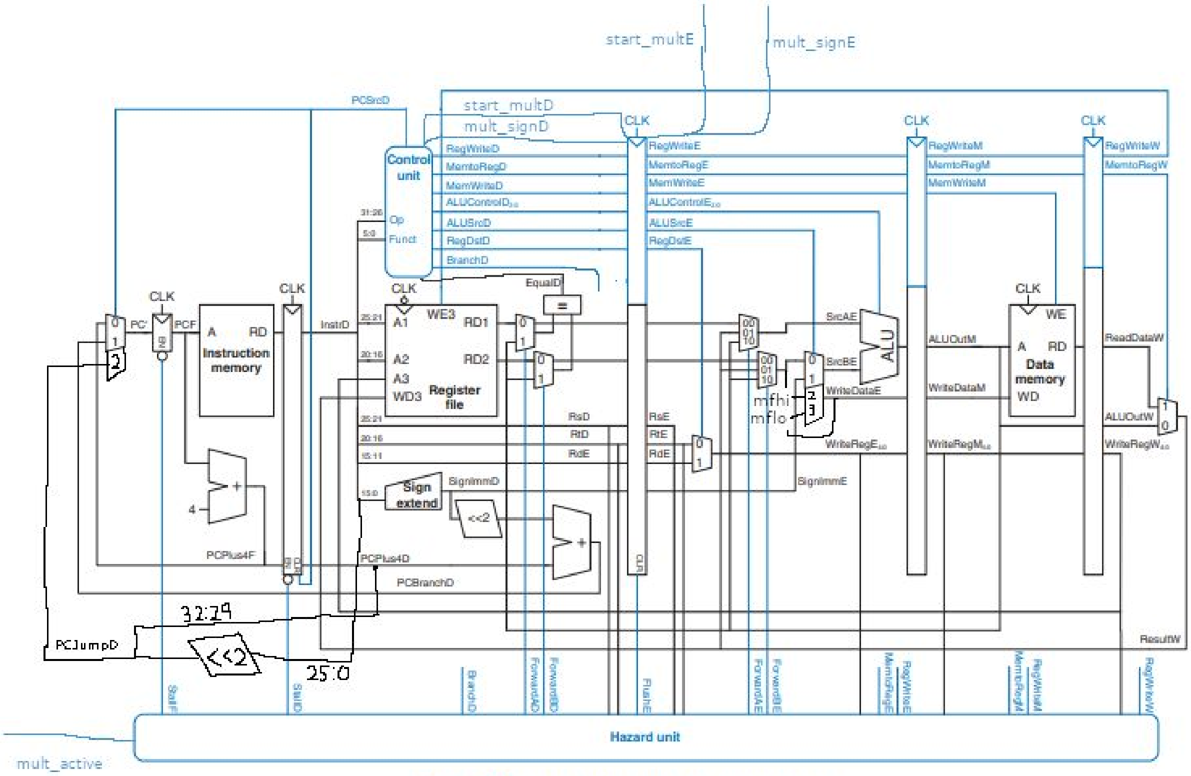


Figure 7.58 **Pipelined processor with full hazard handling**

Memfile1.dat

This program is the same as the one in figure 7.60 on page 437. The regfile and memory should have these values:

$2 = 7

$3 = 12

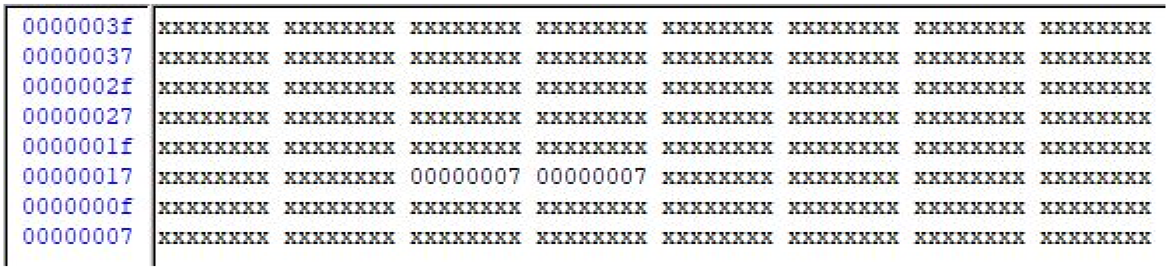
$4 = 1

$5 = 11

$7 = 7

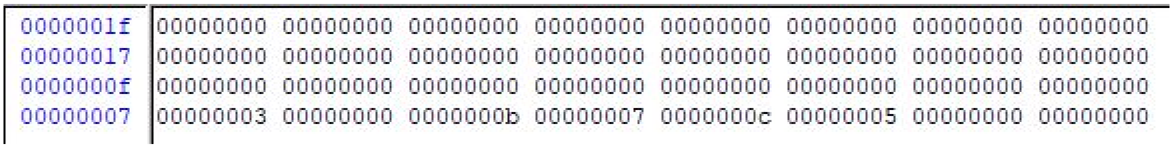
mem[80] = $7 = 7

mem[84] = $2 = 7



```
0000003f xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000037 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000002f xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000027 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000001f xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000017 xxxxxxxx xxxxxxxx 00000007 00000007 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000000f xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000007 xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

Data memory

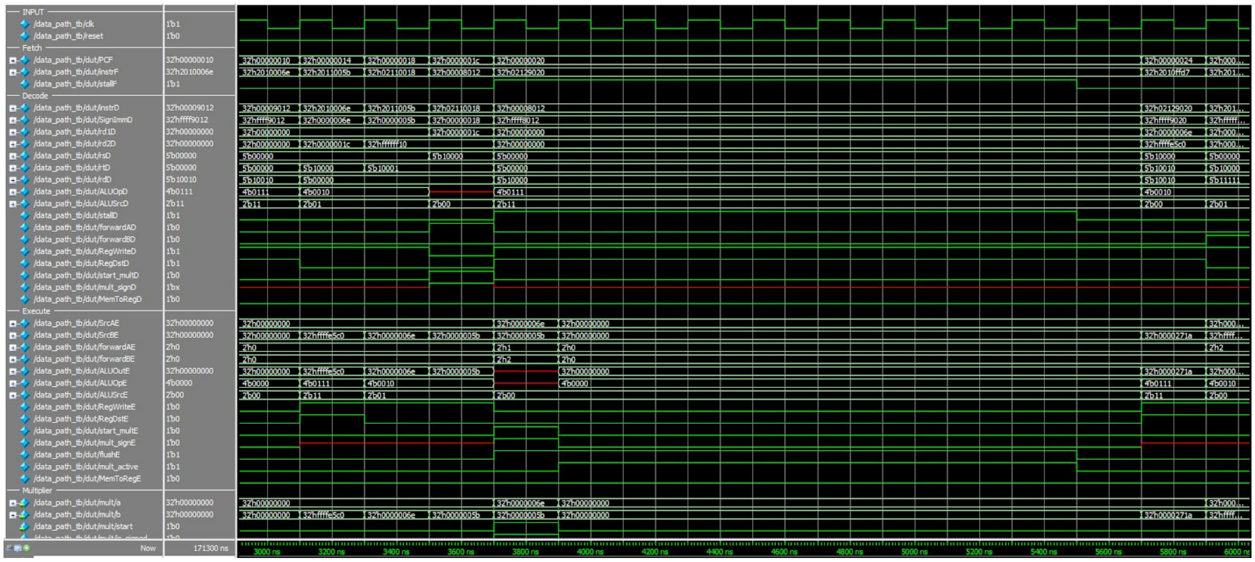


```
0000001f 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000017 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0000000f 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000007 00000003 00000000 0000000b 00000007 0000000c 00000005 00000000 00000000
```

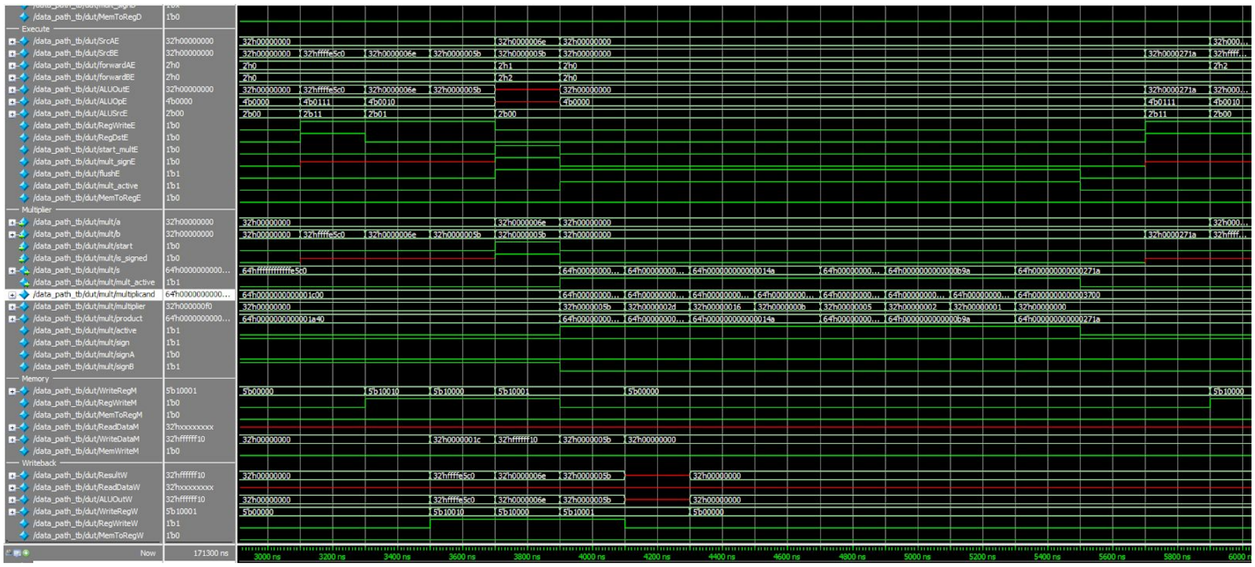
Regfile

Memfile2.dat

This program finds the dot product of two vectors. The vectors are not initialized in memory, but one element from each vector is stored at a time in the regfile. The answer is 0d4985 or 0h1379. It takes 75 cycles to write the dot product back into the regfile.



```
INPUT
/data_path_tb/clk                 1'b1
/data_path_tb/reset               1'b0
Fetch
/data_path_tb/dut/PCF             32'h00000010
/data_path_tb/dut/instrF          32'h2010006e
/data_path_tb/dut/stallF          1'b1
Decode
/data_path_tb/dut/instrD          32'h00009012
/data_path_tb/dut/SignImmD        32'hffff9012
/data_path_tb/dut/rd1D            32'h00000000
/data_path_tb/dut/rd2D            32'h00000000
/data_path_tb/dut/rsD             5'b00000
/data_path_tb/dut/rtD             5'b00000
/data_path_tb/dut/rdD             5'b10010
/data_path_tb/dut/ALUOpD          4'b0111
/data_path_tb/dut/ALUSrcD         2'b11
/data_path_tb/dut/stallD          1'b1
/data_path_tb/dut/forwardAD       1'b0
/data_path_tb/dut/forwardBD       1'b0
/data_path_tb/dut/RegWriteD       1'b1
/data_path_tb/dut/RegDstD         1'b1
/data_path_tb/dut/start_multD     1'b0
/data_path_tb/dut/mult_signD      1'bx
/data_path_tb/dut/MemToRegD       1'b0
Execute
/data_path_tb/dut/SrcAE           32'h00000000
/data_path_tb/dut/SrcBE           32'h00000000
/data_path_tb/dut/forwardAE       2'h0
/data_path_tb/dut/forwardBE       2'h0
/data_path_tb/dut/ALUOutE         32'h00000000
/data_path_tb/dut/ALUOpE          4'b0000
/data_path_tb/dut/ALUSrcE         2'b00
/data_path_tb/dut/RegWriteE       1'b0
/data_path_tb/dut/RegDstE         1'b0
/data_path_tb/dut/start_multE     1'b0
/data_path_tb/dut/mult_signE      1'b0
/data_path_tb/dut/flushE          1'b1
/data_path_tb/dut/mult_active     1'b1
/data_path_tb/dut/MemToRegE       1'b0
Multiplier
/data_path_tb/dut/mult/a          32'h00000000
/data_path_tb/dut/mult/b          32'h00000000
/data_path_tb/dut/mult/start      1'b0
Now                               171300 ns
```

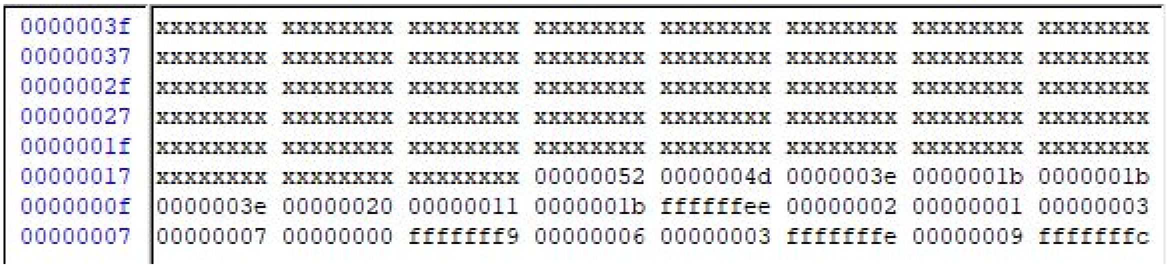


```
/data_path_tb/dut/MemToRegD          1'b0
Execute
/data_path_tb/dut/SrcAE              32'h00000000
/data_path_tb/dut/SrcBE              32'h00000000
/data_path_tb/dut/forwardAE          2'h0
/data_path_tb/dut/forwardBE          2'h0
/data_path_tb/dut/ALUOutE            32'h00000000
/data_path_tb/dut/ALUOpE             4'b0000
/data_path_tb/dut/ALUSrcE            2'b00
/data_path_tb/dut/RegWriteE          1'b0
/data_path_tb/dut/RegDstE            1'b0
/data_path_tb/dut/start_multE        1'b0
/data_path_tb/dut/mult_signE         1'b0
/data_path_tb/dut/flushE             1'b1
/data_path_tb/dut/mult_active        1'b1
/data_path_tb/dut/MemToRegE          1'b0
Multiplier
/data_path_tb/dut/mult/a             32'h00000000
/data_path_tb/dut/mult/b             32'h00000000
/data_path_tb/dut/mult/start         1'b0
/data_path_tb/dut/mult/is_signed     1'b0
/data_path_tb/dut/mult/s             64'h0000000000...
/data_path_tb/dut/mult/mult_active   1'b1
/data_path_tb/dut/mult/multiplicand  64'h0000000000...
/data_path_tb/dut/mult/multiplier    32'h000000f0
/data_path_tb/dut/mult/product       64'h0000000000...
/data_path_tb/dut/mult/active        1'b1
/data_path_tb/dut/mult/sign          1'b1
/data_path_tb/dut/mult/signA         1'b0
/data_path_tb/dut/mult/signB         1'b1
Memory
/data_path_tb/dut/WriteRegM          5'b10001
/data_path_tb/dut/RegWriteM          1'b0
/data_path_tb/dut/MemToRegM          1'b0
/data_path_tb/dut/ReadDataM          32'hxxxxxxxx
/data_path_tb/dut/WriteDataM         32'hffffff10
/data_path_tb/dut/MemWriteM          1'b0
Writeback
/data_path_tb/dut/ResultW            32'hffffff10
/data_path_tb/dut/ReadDataW          32'hxxxxxxxx
/data_path_tb/dut/ALUOutW            32'hffffff10
/data_path_tb/dut/WriteRegW          5'b10001
/data_path_tb/dut/RegWriteW          1'b1
/data_path_tb/dut/MemToRegW          1'b0
Now                                  171300 ns
```

Waveform for computing product, then summation of second element.

Memfile3.dat

For two arrays in memory of n length, Y[j+1] = B*X[j]+Y[j]. Instead of using the multiplier for B*X[j], the program instead adds X[j] B times as a running sum. B and n are initialized in $a0 and $a1 respectively. The first 10 memory slots are for X and the next 11 are for Y.



```
0000003f  xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000037  xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000002f  xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000027  xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000001f  xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
00000017  xxxxxxxx xxxxxxxx xxxxxxxx 00000052 0000004d 0000003e 0000001b 0000001b
0000000f  0000003e 00000020 00000011 0000001b ffffffee 00000002 00000001 00000003
00000007  00000007 00000000 fffffff9 00000006 00000003 fffffffe 00000009 fffffffc
```

Data Memory

Conclusion

Pipeline processors are faster than single cycle and multi cycle processors because they can execute multiple instructions at a time. The limiting factors are the amount of hazards as the number of stages increase and the cost of setup and hold times between every pipeline register. Increasing the number of stages might slow down the processor due to the increase in hazards and pipeline register overhead. All the hazards force the processor to stall for a number of cycles and the goal is to decrease the number of stalls as much as possible.