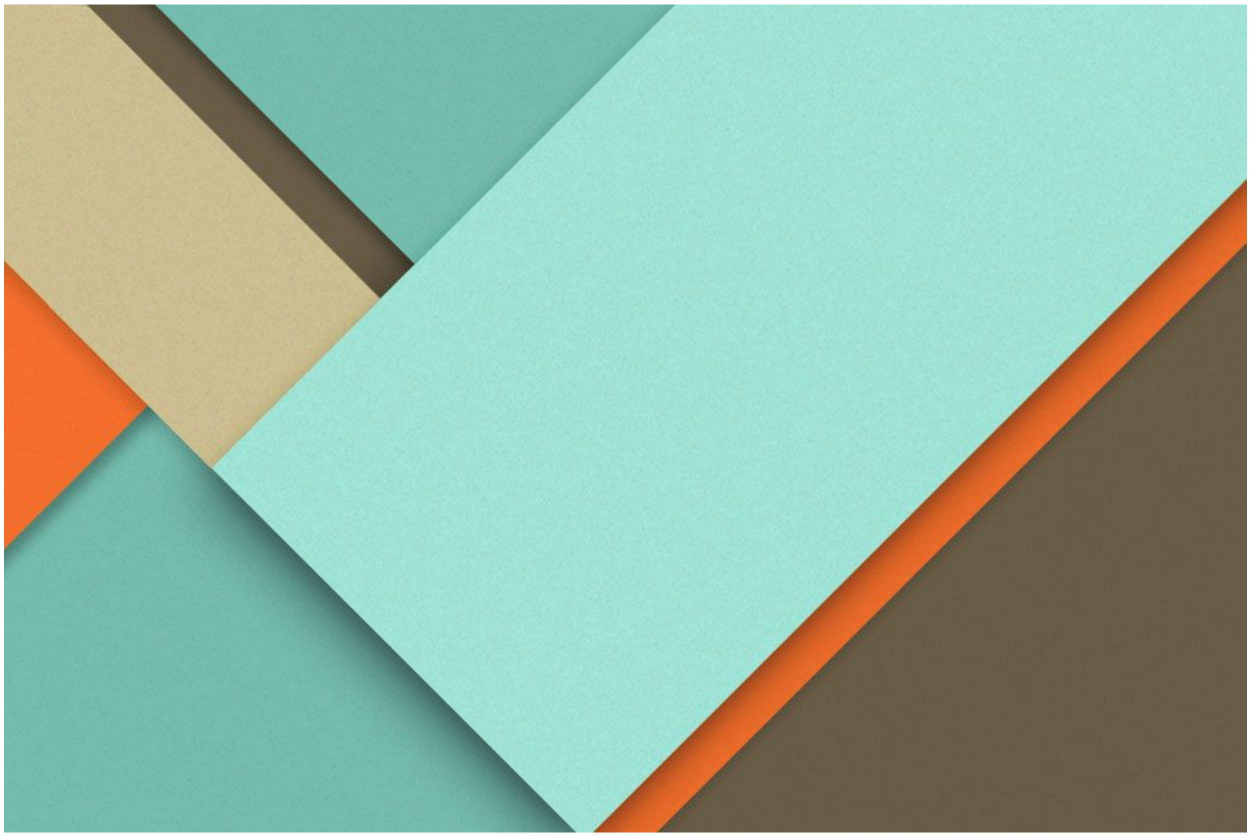


MIPS PIPELINED PROCESSOR

09.06.2017

CATHERINE FARAHAT.
3689.

MOUNIR MAHER.
3323.

MOHAMMED ASHRAF.
3265.

Overview

In this project we implemented a subset of the pipelined MIPS architecture in HDL. We implemented a functioning outline of the pipelined processor for a small set of instructions, including: decoding all the instructions encountered in this project, implementing most of the MIPS pipeline, correct implementation of arithmetic and logic operations, and implementing a hazard detection and avoidance unit for these instructions.

Pipelining is a powerful way to improve the throughput of a digital system. We design a pipelined processor by subdividing the single-cycle processor into five pipeline stages. Thus, five instructions can execute simultaneously, one in each stage. Because each stage has only one-fifth of the entire logic, the clock frequency is almost five times faster. Hence, the latency of each instruction is ideally unchanged, but the throughput is ideally five times better. Microprocessors execute millions or billions of instructions per second, so throughput is more important than latency. Pipelining introduces some overhead, so the throughput will not be quite as high as we might ideally desire, but pipelining nevertheless gives such great advantage for so little cost that all modern high-performance microprocessors are pipelined.

The pipelined processor is designed by:

- Divide single-cycle processor into 5 stages:

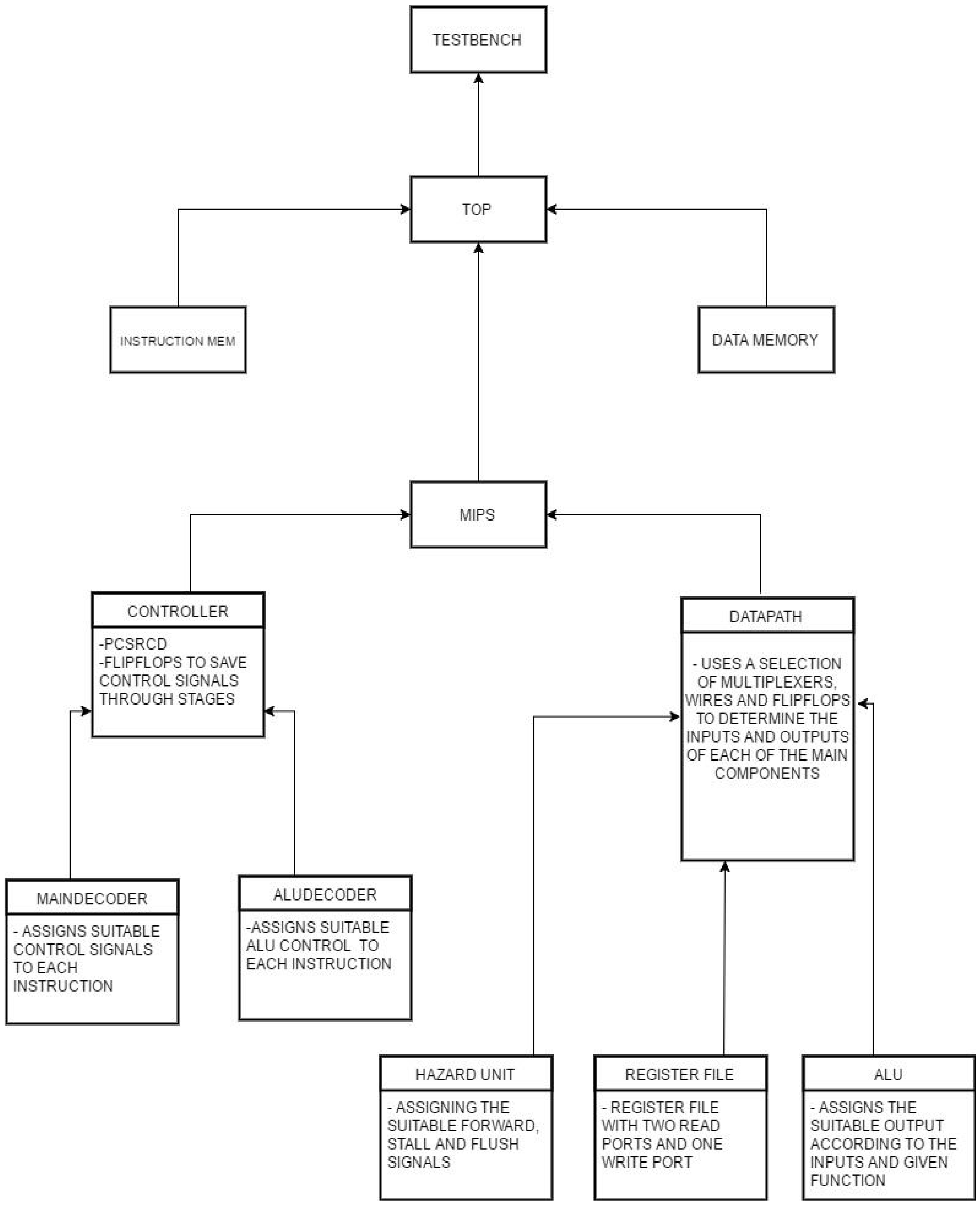
  –Fetch

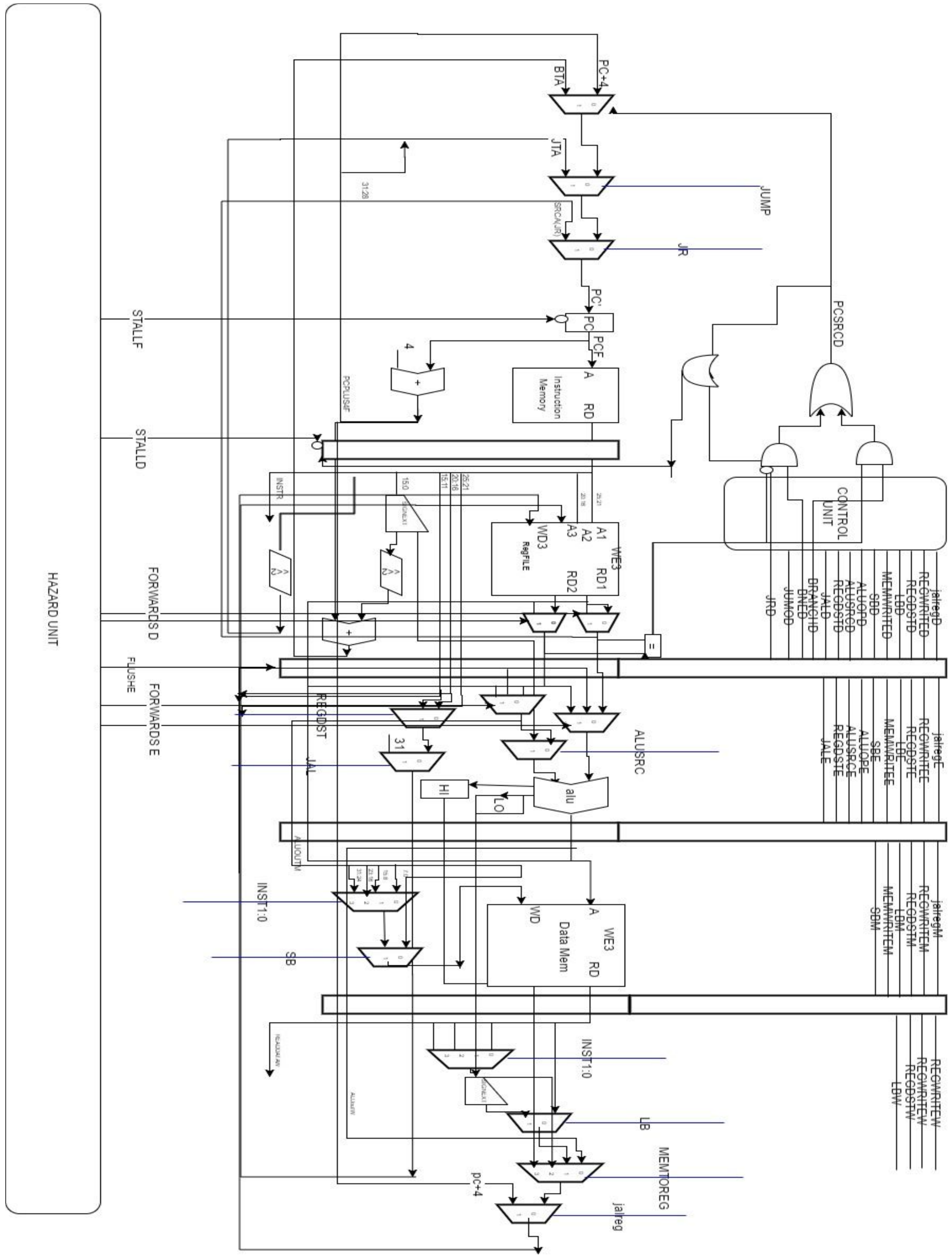
  –Decode

  –Execute

  –Memory

  –Writeback
- Add pipeline registers between stages


TESTBENCH
TOP
INSTRUCTION MEM
DATA MEMORY
MIPS
CONTROLLER
-PCSRCD
-FLIPFLOPS TO SAVE CONTROL SIGNALS THROUGH STAGES
DATAPATH
- USES A SELECTION OF MULTIPLEXERS, WIRES AND FLIPFLOPS TO DETERMINE THE INPUTS AND OUTPUTS OF EACH OF THE MAIN COMPONENTS
MAINDECODER
- ASSIGNS SUITABLE CONTROL SIGNALS TO EACH INSTRUCTION
ALUDECODER
-ASSIGNS SUITABLE ALU CONTROL TO EACH INSTRUCTION
HAZARD UNIT
- ASSIGNING THE SUITABLE FORWARD, STALL AND FLUSH SIGNALS
REGISTER FILE
- REGISTER FILE WITH TWO READ PORTS AND ONE WRITE PORT
ALU
- ASSIGNS THE SUITABLE OUTPUT ACCORDING TO THE INPUTS AND GIVEN FUNCTION


HAZARD UNIT
STALLF
STALLD
FORWARDS D
FLUSHE
FORWARDS E
PC+4
BTA
JTA
JUMP
SRCA(JR)
JR
PC'
PC
PCF
PCSRCD
4
PCPLUS4F
Instruction Memory
A
RD
CONTROL UNIT
jalregD
REGWRITED
REGDSTD
LBD
MEMWRITED
SBD
ALUOPD
ALUSRCD
REGDSTD
JALD
BRANCHD
BNED
JUMOD
JRD
INSTR
RegFILE
WE3
A1
A2
A3
WD3
RD1
RD2
=
REGDST
31
JAL
ALUSRC
jalregE
REGWRITEE
REGDSTE
LBE
MEMWRITEE
SBE
ALUOPE
ALUSRCE
REGDSTE
JALE
alu
HI
LO
ALUOUTM
INST1:0
SB
Data Mem
WE3
A
WD
RD
jalregM
REGWRITEM
REGDSTM
LBM
MEMWRITEM
SBM
INST1:0
LB
MEMTOREG
pc+4
jalreg
REGWRITEW
REGWRITEW
REGDSTW
LBW

Mips pipelined processor main parts

I. Controller

It calls the maindecoder and aludecoder modules to assign the suitable control signals to each instruction.

It assigns the suitable PCSRCD instruction accouding to the following function:

pcsrcD = (branchD & equalD) | (bneD & ~equalD)

It also contains the flip flops that save the control signals of the instruction through the needed stages.

A. Main decoder.

It assigns all the control signals for each instruction.

It takes both the opcode and the function as inputs and assigns the following control signals as outputs according to the input opcode and function.

1. Register write -> 1 bit.
2. Register destination -> 1 bit.
3. Alu source -> 1 bit.
4. Branch -> 1 bit.
5. Branch if not equal -> 1 bit.
6. Mem write -> 1 bit.
7. Mem to register -> 2 bits.
8. Jump -> 1 bit.
9. Alu op -> 2 bits.
10. Store byte -> 1 bit.
11. Load byte -> 1 bit.
12. Jump register -> 1 bit.
13. Jump and link -> 1 bit.
14. Jump and link register -> 1 bit.

Main decoder table

Instr	regwrite	regdst	alusrc	branch	bne	memwrite	memtoreg	jump	aluop	sb	lb	jr	jal	jalreg
R-TYPE	1	1	0	0	0	0	00	0	10	0	0	0	0	0
MFLO	1	1	0	0	0	0	10	0	10	0	0	0	0	0
MFHI	1	1	0	0	0	0	11	0	10	0	0	0	0	0
LW	1	0	1	0	0	0	01	0	00	0	0	0	0	0
SW	0	0	1	0	0	1	00	0	00	0	0	0	0	0
BEQ	0	0	0	1	0	0	00	0	01	0	0	0	0	0
ADDI	1	0	1	0	0	0	00	0	00	0	0	0	0	0
JUMP	0	0	0	0	0	0	00	1	00	0	0	0	0	0
BNE	0	0	0	0	1	0	00	0	01	0	0	0	0	0
SLTI	1	0	1	0	0	0	00	0	11	0	0	0	0	0
SB	0	0	1	0	0	1	00	0	00	1	0	0	0	0
LB	1	0	1	0	0	0	01	0	00	0	1	0	0	0
JR	1	1	0	0	0	0	00	0	10	0	0	1	0	0
JAL	0	0	0	0	0	0	00	1	00	0	0	0	1	1

B. ALU decoder.

It’s main function is basically telling the ALU which function should it do.

It takes aluop as input and assigns the value of alu control(add, sub,slt).

If the instruction is R type the alu control depends on the function.

II. DATA PATH

It is the main part that connects all the signals and wires of the processor.

First part of the data path is calling the hazard unit to handle data and control hazards and assigning the suitable forwarding and stalling signals.

A. Hazard Unit.

Pipelined hazards:

•When an instruction depends on result from instruction that hasn't completed

•Types:

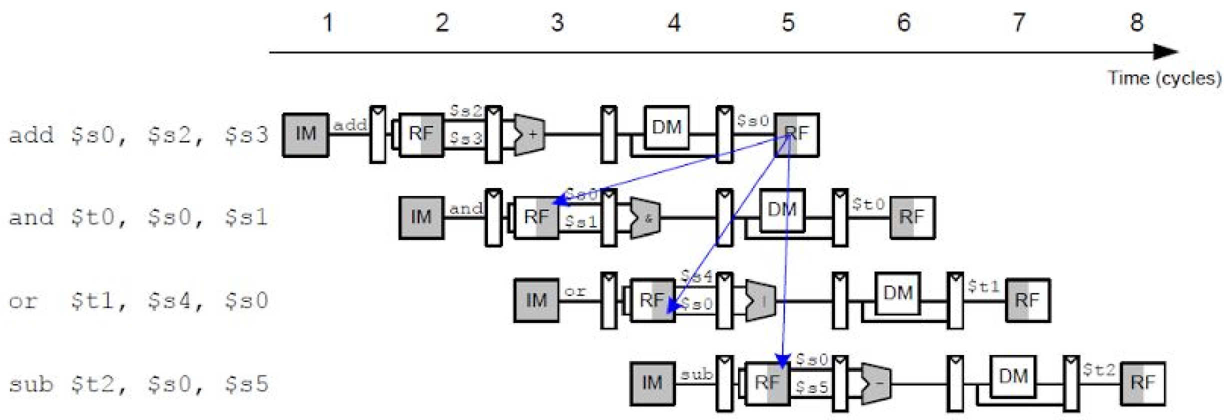
–Data hazard:

register value not yet written back to register file.

–Control hazard:

next instruction not decided yet (caused by branches).

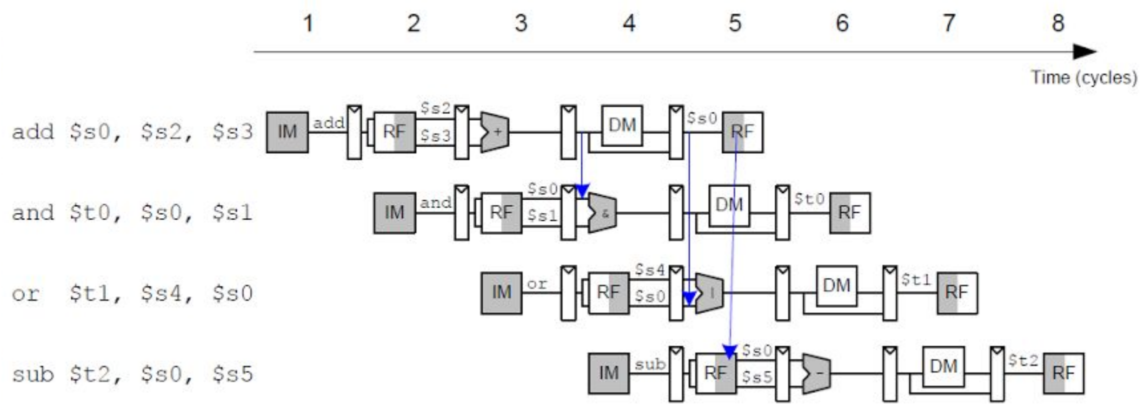
1. Handling data hazards.



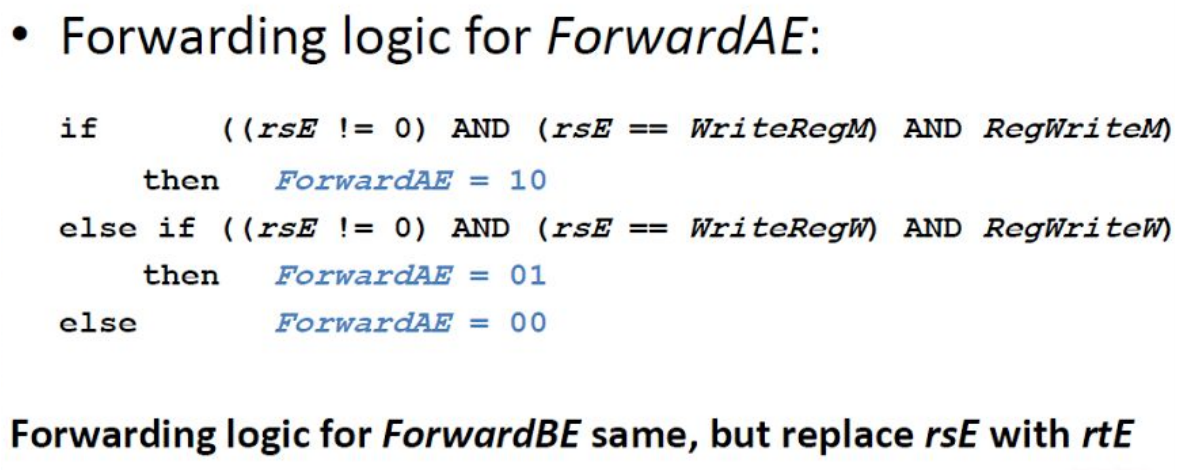
Data hazards occur when the sources of the current instructions are the destination of the previous instruction or the one before that.

According to the instruction there are several ways for solving the data hazards through compiling or in the processor hardware itself but we're designing a processor so we'll only discuss processor solutions for data hazards.

a. Forwarding.



The source is forwarded to execute stage from the memory stage or the write back stage according to the following formulas:



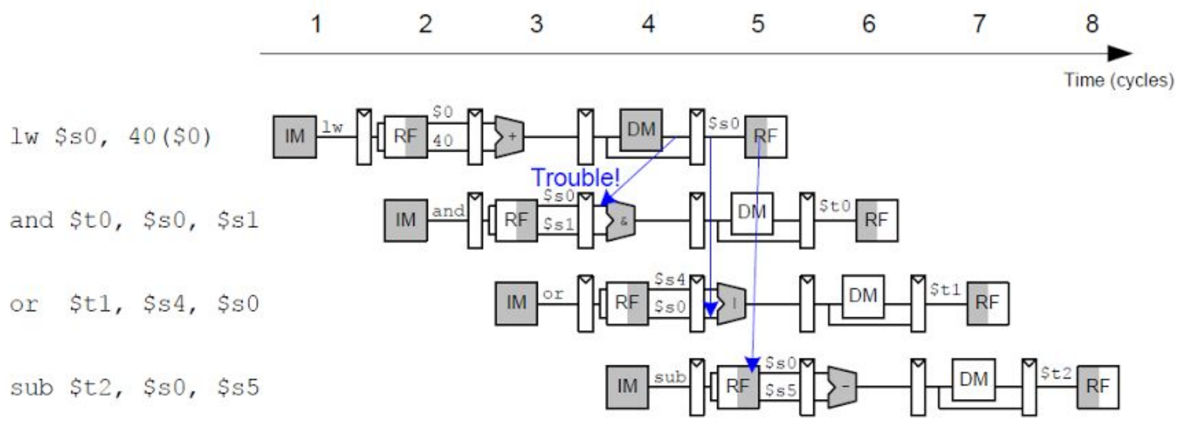
- Forwarding logic for *ForwardAE*:

```
if      ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
     then    ForwardAE = 10
else if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
     then    ForwardAE = 01
else         ForwardAE = 00
```

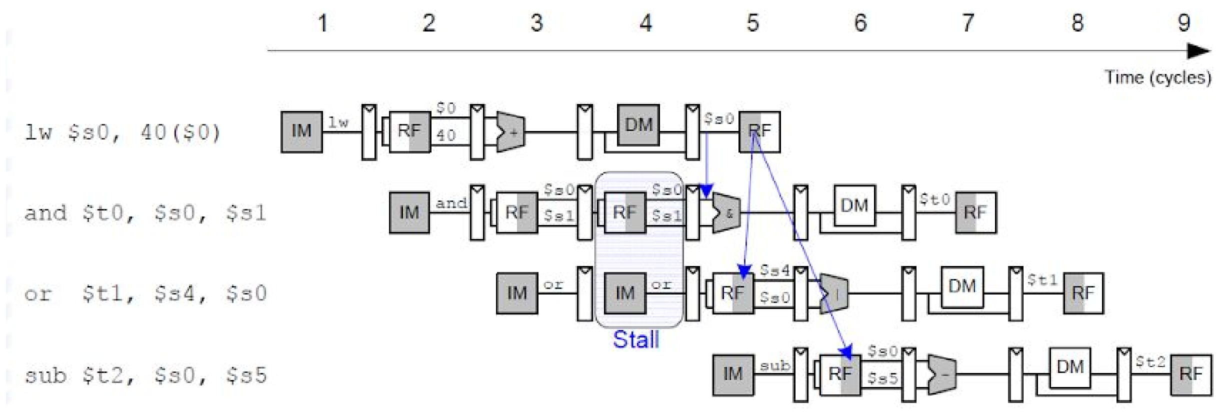
Forwarding logic for *ForwardBE* same, but replace *rsE* with *rtE*

b. stalling.

The previous formula handles the cases when the result of the instruction is ready after the execute stage but it doesn't handle the instructions in which the data is only ready after accessing the memory (load word instruction).



The only way to handle this kind of hazard is by stalling the processor (stalling fetch and decode stages and flushing the execute) to make sure the data is ready before forwarding it to the execute stage.



The formula of stalling is as follows:

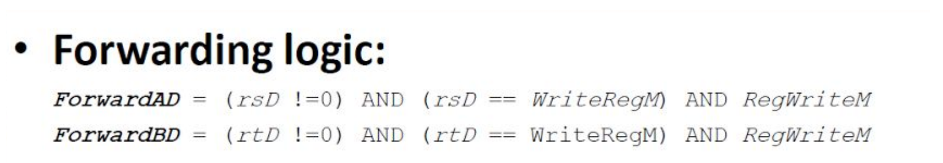
```
lwstall =
   ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE

StallF = StallD = FlushE = lwstall
```

2. Handling control hazards.

Control hazards occur when the next instruction to be fetched is not yet determined due to branching or jumping instructions.

Another hazard occurs in the branch instructions is the early branch resolution (the comparison between the two sources is done in the decode stage to avoid flushing extra instructions if the sources are yet to be determined).

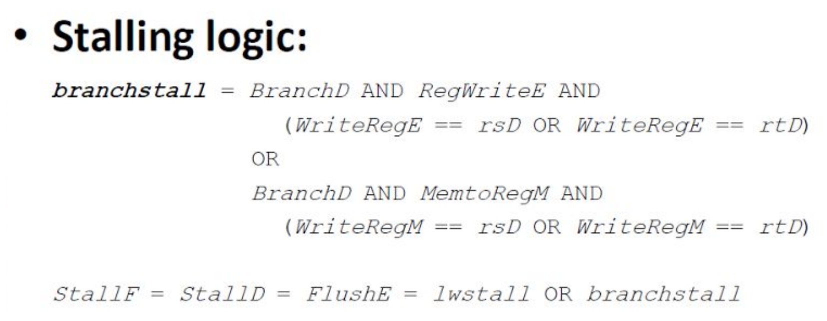
This hazard is handled by forwarding the sources to the decode stage and it's done according to the following formula:



- **Forwarding logic:**

```
ForwardAD = (rsD !=0) AND (rsD == WriteRegM) AND RegWriteM
ForwardBD = (rtD !=0) AND (rtD == WriteRegM) AND RegWriteM
```

If any of the sources is the destination in the previous instruction then there is no escape from stalling the processor according to the following logic:



- **Stalling logic:**

```
branchstall = BranchD AND RegWriteE AND
                (WriteRegE == rsD OR WriteRegE == rtD)
              OR
              BranchD AND MemtoRegM AND
                (WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = lwstall OR branchstall
```

The jump instructions (all kinds) also stall the processor and flush the next instruction.

The hazard unit in our processor handles all the previous cases and it needs the following inputs and outputs to be able to do it's job correctly.

I. Inputs:

- rsD -> 5 bits
- rtD -> 5 bits
- rsE -> 5 bits
- rtE -> 5 bits
- writeregE -> 5 bits
- writeregM -> 5 bits
- writeregW -> 5 bits
- regwriteE -> 1 bit
- regwriteM -> 1 bit
- regwriteW -> 1 bit

- memtoregE -> 2 bits
- memtoregM -> 2 bits
- branchD -> 1 bit
- bneD -> 1 bit

II. Outputs:

- forwardaD -> 1 bit
- forwardbD -> 1 bit
- forwardaE -> 2 bits
- forwardbE -> 2 bits
- stallF -> 1 bit
- stallD -> 1 bit
- flushE -> 1 bit

The datapath then calls an instance of the register file module.

B. Register file.

The mips register file has three ports:

- A1 : read data port.
- A2: read data port.
- A3: write data port.

After that the data path handles the PC logic using multiplexers

- 1st mux chooses between pc + 4 or pc branch according to the control signal pcsrc.
- 2nd mux chooses between 1st one's output and the jump target according to the jump control signal.
- 3rd mux chooses between the 2nd one's output and the content of 1st register source according to the jump register control signal.

The data path after that goes through the logic of the five stages of pipelined processor in order of fetch, decode, execute, memory, write back.

+ Fetch:
  - A register to save the value of the pc address to be fetched.
  - An adder to calculate next PC address to be fetched (pc+4).
+ Decode:
  - Registers to save the pc+4 value and the fetched instruction.
  - Dividing the instruction into its main parts:
    - Rs
    - Rt
    - Rd
    - Opcode
    - Function
    - Shift amount
    - 16 bit immediate
  - Sign extending the immediate to 32 bits.
  - Shifting the immediate by 2 bits and adding it to the current pc+4 to calculate the branch target address.
  - 2 mux to choose the data read from the register file or forwarded from memory stage to be the sources entering the execute stage.
  - assigning the flushD signal according to the instruction(branch or jump).
+ Execute:
  - Registers to save the values of the parts of the instruction divided in the decode stage, pc+4 and the instruction.
  - 2 mux to choose the sources of the alu: read from the register file, forwarded from memory stage or forwarded from write back stage.
  - 1 mux to choose the source b of the Alu to be register or the sign extended immediate according to the alusrc control signal.
  - 1 mux to choose the register destination to be rt or rd according to the instruction type (R or I).
  - 1 mux to choose the final register destination between the output of the previous mux and the register (31) ($ra) according to jal control signal.
  - Instantiating the Alu (described later).

+ Memory:

- Registers to save the values of pc+4, hi, lo, write data, aluout, instruction and writereg.

- 1 mux to choose which byte of data is going to be stored according to the first 2 bits of the instruction.

- 1 mux to choose to store the full word or only a single byte of data according to the store byte control signal.

+ Write back:

- Registers to save the values of instruction, pc+4, lo, hi, aluout, read data and write reg.

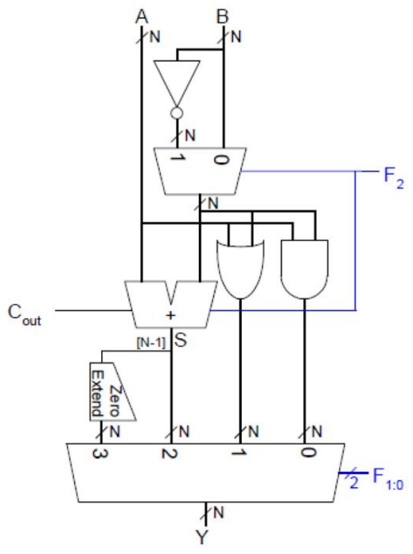
- 1 mux to choose which byte of data is going to be loaded according to the first 2 bits of the instruction.

- 1 mux to choose if the data to be written is from the memory, alu, lo or hi according to the memtoreg control signal.

- 1 mux to choose the final data to be written is the result of the previous mux of the value of pc+4 according to the jalreg control signal.

C. ALU.

Arithmetic logic unit (ALU) takes 2 inputs of 32 bits, a 3 bit function, a 5 bit shift amount and sets a 32 bit output, 2 32 bit registers lo and hi.



The functionality of the alu is extended by adding shift right, shift left, multiply and divide.

Adding instructions to pipelined processor:

1. Multiply.

- ALU:

Required adding 2 parameters in the alu (hi and lo). Hi takes the most significant 32 bits of the result of the multiplication while lo takes the least significant bits.

- Data path:

Required adding 2 registers hi and lo to save the values resulting from ALU.

- Controller:

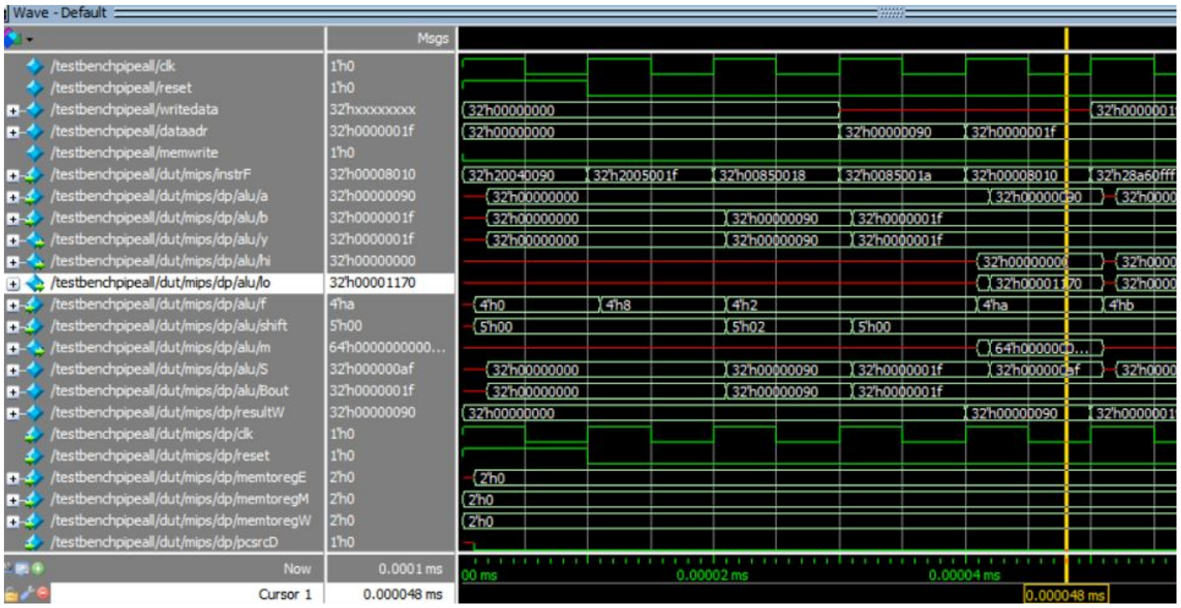
No additional changes from the normal R type control signals

Testing:

Addi a0, 0, 90

Addi a1, 0, 1f

Mult a0, a1 low = 1170 high = 0



```
Wave - Default
                                              Msgs
/testbenchpipeall/clk                         1'h0
/testbenchpipeall/reset                       1'h0
/testbenchpipeall/writedata                   32'hxxxxxxxx
/testbenchpipeall/dataadr                     32'h0000001f
/testbenchpipeall/memwrite                    1'h0
/testbenchpipeall/dut/mips/instrF             32'h00008010
/testbenchpipeall/dut/mips/dp/alu/a           32'h00000090
/testbenchpipeall/dut/mips/dp/alu/b           32'h0000001f
/testbenchpipeall/dut/mips/dp/alu/y           32'h0000001f
/testbenchpipeall/dut/mips/dp/alu/hi          32'h00000000
/testbenchpipeall/dut/mips/dp/alu/lo          32'h00001170
/testbenchpipeall/dut/mips/dp/alu/f           4'ha
/testbenchpipeall/dut/mips/dp/alu/shift       5'h00
/testbenchpipeall/dut/mips/dp/alu/m           64'h0000000000...
/testbenchpipeall/dut/mips/dp/alu/S           32'h000000af
/testbenchpipeall/dut/mips/dp/alu/Bout        32'h0000001f
/testbenchpipeall/dut/mips/dp/resultW         32'h00000090
/testbenchpipeall/dut/mips/dp/clk             1'h0
/testbenchpipeall/dut/mips/dp/reset           1'h0
/testbenchpipeall/dut/mips/dp/memtoregE       2'h0
/testbenchpipeall/dut/mips/dp/memtoregM       2'h0
/testbenchpipeall/dut/mips/dp/memtoregW       2'h0
/testbenchpipeall/dut/mips/dp/pcsrcD          1'h0
                                   Now        0.0001 ms
                              Cursor 1        0.000048 ms
```

2. Divide.

- ALU:

Required adding 2 parameters in the alu (hi and lo). Hi takes the remainder of the division while lo takes the quotient.

- Data path:

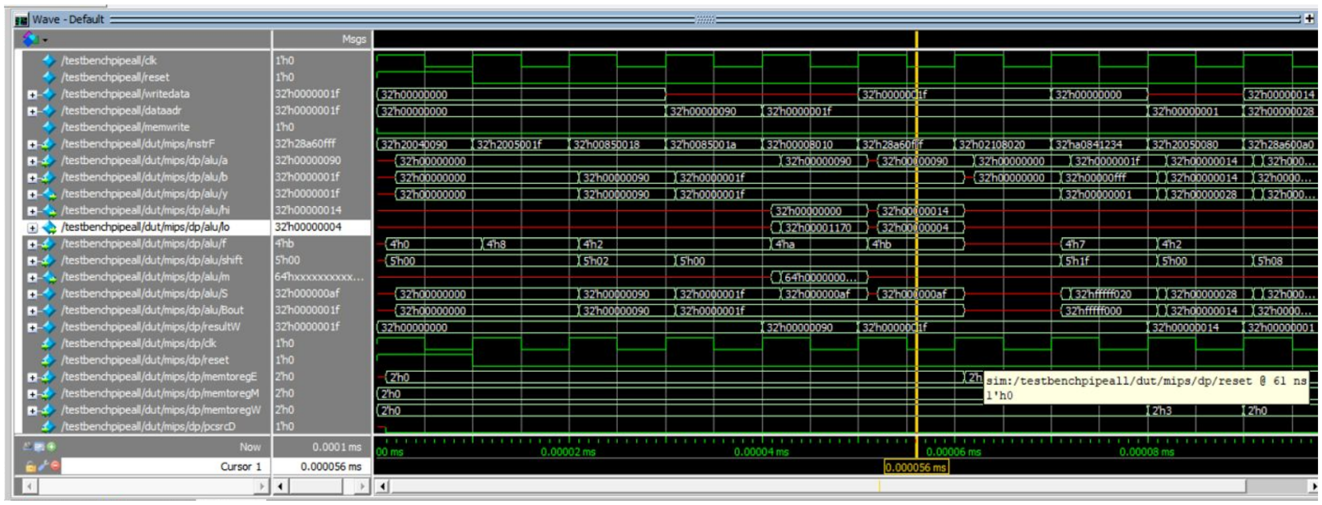
Same as Multiply.

- Controller:

Same as any R type.

Testing:

Div a0, a1 low = 4 high = 14



```
Wave - Default
                                              Msgs
/testbenchpipeall/clk                         1'h0
/testbenchpipeall/reset                       1'h0
/testbenchpipeall/writedata                   32'h0000001f
/testbenchpipeall/dataadr                     32'h0000001f
/testbenchpipeall/memwrite                    1'h0
/testbenchpipeall/dut/mips/instrF             32'h28a60fff
/testbenchpipeall/dut/mips/dp/alu/a           32'h00000090
/testbenchpipeall/dut/mips/dp/alu/b           32'h0000001f
/testbenchpipeall/dut/mips/dp/alu/y           32'h0000001f
/testbenchpipeall/dut/mips/dp/alu/hi          32'h00000014
/testbenchpipeall/dut/mips/dp/alu/lo          32'h00000004
/testbenchpipeall/dut/mips/dp/alu/f           4'hb
/testbenchpipeall/dut/mips/dp/alu/shift       5'h00
/testbenchpipeall/dut/mips/dp/alu/m           64'hxxxxxxxxxxx...
/testbenchpipeall/dut/mips/dp/alu/S           32'h000000af
/testbenchpipeall/dut/mips/dp/alu/Bout        32'h0000001f
/testbenchpipeall/dut/mips/dp/resultW         32'h0000001f
/testbenchpipeall/dut/mips/dp/clk             1'h0
/testbenchpipeall/dut/mips/dp/reset           1'h0
/testbenchpipeall/dut/mips/dp/memtoregE       2'h0
/testbenchpipeall/dut/mips/dp/memtoregM       2'h0
/testbenchpipeall/dut/mips/dp/memtoregW       2'h0
/testbenchpipeall/dut/mips/dp/pcsrcD          1'h0
                                    Now       0.0001 ms
                               Cursor 1       0.000056 ms

sim:/testbenchpipeall/dut/mips/dp/reset @ 61 ns
1'h0
```

3. Move from high.

4. Move from low.

- ALU:

No additionals changes.

- Data path:

Saving the values of the lo and hi registers through the stages till write back.

Increasing the memtoreg mux to hold 4 values instead of 2

0- alu result

1- data read from memory

2- lo register

3- hi register

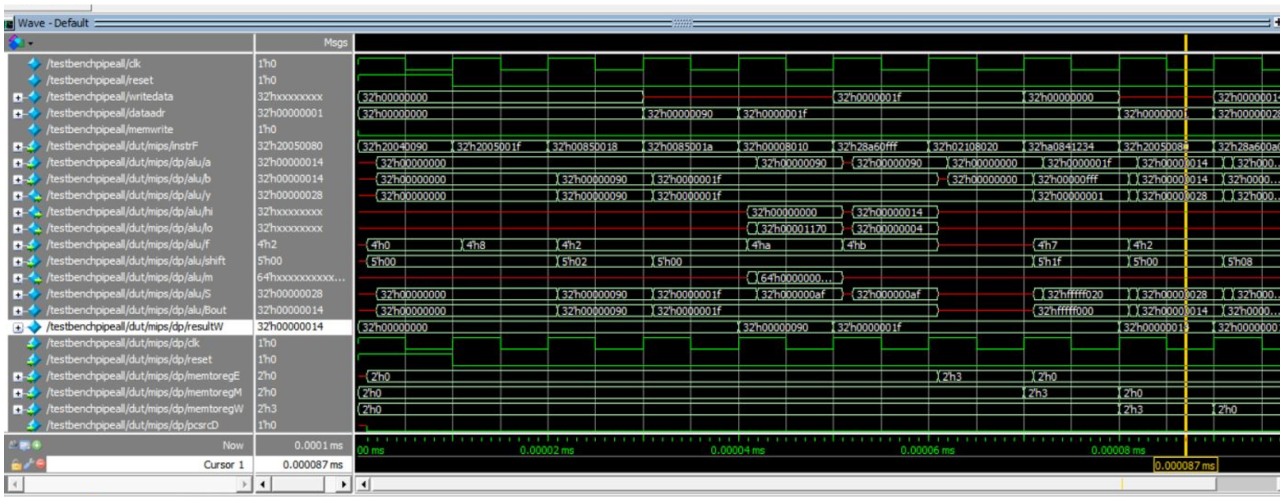
- Controller:

Memtoreg changes to 2 bits instead of one.

Same as R type except for the memtoreg control signal (10 mflo - 11 mfhi).

Testing:

Mfhi a0 resultw (to be written in register file) = 14



	Msgs
/testbenchpipeall/clk	1'h0
/testbenchpipeall/reset	1'h0
/testbenchpipeall/writedata	32'hxxxxxxxx
/testbenchpipeall/dataadr	32'h00000001
/testbenchpipeall/memwrite	1'h0
/testbenchpipeall/dut/mips/instrF	32'h20050080
/testbenchpipeall/dut/mips/dp/alu/a	32'h00000014
/testbenchpipeall/dut/mips/dp/alu/b	32'h00000014
/testbenchpipeall/dut/mips/dp/alu/y	32'h00000028
/testbenchpipeall/dut/mips/dp/alu/hi	32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/alu/lo	32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/alu/f	4'h2
/testbenchpipeall/dut/mips/dp/alu/shift	5'h00
/testbenchpipeall/dut/mips/dp/alu/m	64'hxxxxxxxxxxxx...
/testbenchpipeall/dut/mips/dp/alu/S	32'h00000028
/testbenchpipeall/dut/mips/dp/alu/Bout	32'h00000014
/testbenchpipeall/dut/mips/dp/resultW	32'h00000014
/testbenchpipeall/dut/mips/dp/clk	1'h0
/testbenchpipeall/dut/mips/dp/reset	1'h0
/testbenchpipeall/dut/mips/dp/memtoregE	2'h0
/testbenchpipeall/dut/mips/dp/memtoregM	2'h0
/testbenchpipeall/dut/mips/dp/memtoregW	2'h3
/testbenchpipeall/dut/mips/dp/pcsrcD	1'h0
Now	0.0001 ms
Cursor 1	0.000087 ms

5. Set less than immediate.

- ALU:

No additionals changes.

- Data path:

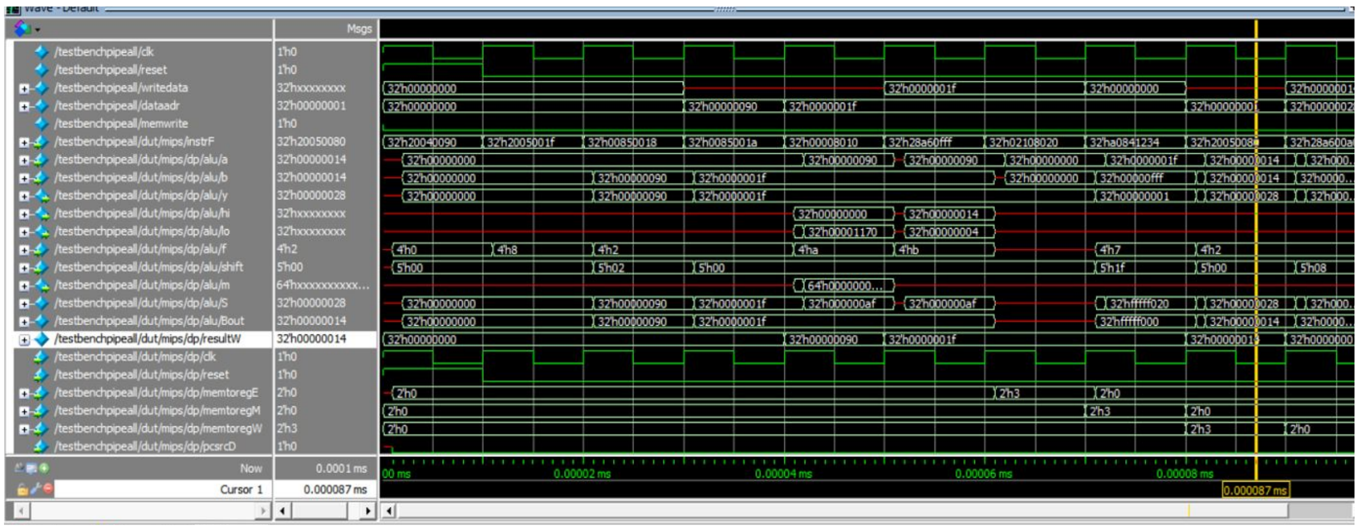
No additionals changes.

- Controller:

Same controls as addi except for the aluop (11 -> alu control = slt)

Testing:

Slti a3, a0, 0fff resultw (to be written in register file) = 1



	Msgs
/testbenchpipeall/clk	1'h0
/testbenchpipeall/reset	1'h0
/testbenchpipeall/writedata	32'hxxxxxxxx
/testbenchpipeall/dataadr	32'h00000001
/testbenchpipeall/memwrite	1'h0
/testbenchpipeall/dut/mips/instrF	32'h20050080
/testbenchpipeall/dut/mips/dp/alu/a	32'h00000014
/testbenchpipeall/dut/mips/dp/alu/b	32'h00000014
/testbenchpipeall/dut/mips/dp/alu/y	32'h00000028
/testbenchpipeall/dut/mips/dp/alu/hi	32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/alu/lo	32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/alu/f	4'h2
/testbenchpipeall/dut/mips/dp/alu/shift	5'h00
/testbenchpipeall/dut/mips/dp/alu/m	64'hxxxxxxxxxxxx...
/testbenchpipeall/dut/mips/dp/alu/S	32'h00000028
/testbenchpipeall/dut/mips/dp/alu/Bout	32'h00000014
/testbenchpipeall/dut/mips/dp/resultW	32'h00000014
/testbenchpipeall/dut/mips/dp/clk	1'h0
/testbenchpipeall/dut/mips/dp/reset	1'h0
/testbenchpipeall/dut/mips/dp/memtoregE	2'h0
/testbenchpipeall/dut/mips/dp/memtoregM	2'h0
/testbenchpipeall/dut/mips/dp/memtoregW	2'h3
/testbenchpipeall/dut/mips/dp/pcsrcD	1'h0
Now	0.0001 ms
Cursor 1	0.000087 ms

6. Branch if not equal.

- ALU:

No additionals changes.

- Data path:

No additionals changes.

- Controller:

Extra control signal bne.

The pcsrc value changes to: (branch & zero) | (bne & ~zero).

Control signals same as branch except branch and bne (01 instead of 10).

7. Load byte.

- ALU:

No additionals changes.

- Data path:

A 4 mux to choose which byte of data to be loaded and one more mux to choose to load the full word or only a byte.

- Controller:

An extra control signal lb.

Same controls as load word except lb = 1.

8. Store byte.

- ALU:

No additionals changes.

- Data path:

A 4 mux to choose which byte of data to be stored and one more mux to choose to store the full word or only a byte.

- Controller:

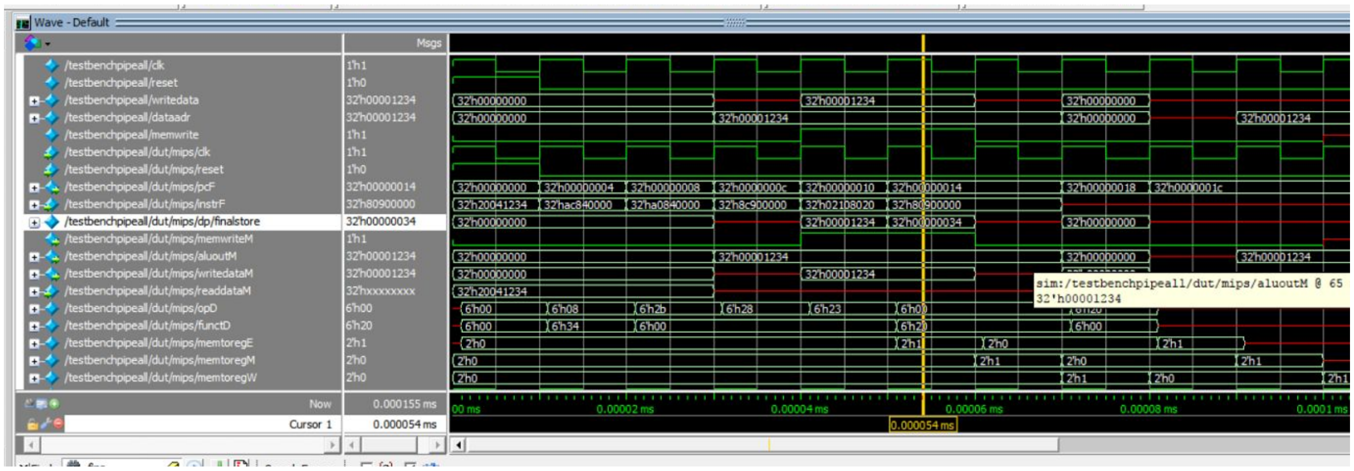
An extra control signal sb.

Same controls as store word except sb = 1.

Testing:

Addi a0 $0 0x1234

Sb a0 $0 0x0000 final data to be stored is 00000034



Wave - Default

	Msgs
/testbenchpipeall/clk	1'h1
/testbenchpipeall/reset	1'h0
/testbenchpipeall/writedata	32'h00001234
/testbenchpipeall/dataadr	32'h00001234
/testbenchpipeall/memwrite	1'h1
/testbenchpipeall/dut/mips/clk	1'h1
/testbenchpipeall/dut/mips/reset	1'h0
/testbenchpipeall/dut/mips/pcF	32'h00000014
/testbenchpipeall/dut/mips/instrF	32'h80900000
/testbenchpipeall/dut/mips/dp/finalstore	32'h00000034
/testbenchpipeall/dut/mips/memwriteM	1'h1
/testbenchpipeall/dut/mips/aluoutM	32'h00001234
/testbenchpipeall/dut/mips/writedataM	32'h00001234
/testbenchpipeall/dut/mips/readdataM	32'hxxxxxxxx
/testbenchpipeall/dut/mips/opD	6'h00
/testbenchpipeall/dut/mips/functD	6'h20
/testbenchpipeall/dut/mips/memtoregE	2'h1
/testbenchpipeall/dut/mips/memtoregM	2'h0
/testbenchpipeall/dut/mips/memtoregW	2'h0
Now	0.000155 ms
Cursor 1	0.000054 ms

```
sim:/testbenchpipeall/dut/mips/aluoutM @ 65
32'h00001234
```

9. Logical shift left.

10. Logical shift right.

- ALU:

Adding 1 extra parameter (shift amount) as an input.

Adding 2 extra functions shift left and right.

- Data path:

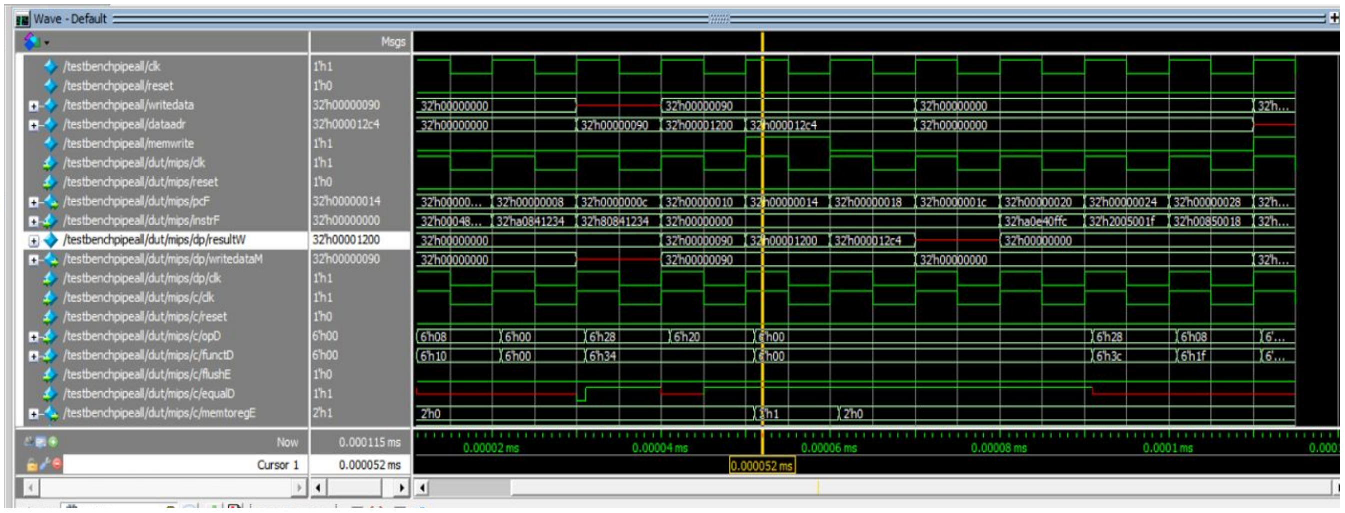
No additionals changes.

- Controller:

Same as any R type.

Testing:

Sll s0 a0 5 resultw to be written (shift 90 by 5 = 1200)


Wave - Default

	Msgs
/testbenchpipeall/clk	1'h1
/testbenchpipeall/reset	1'h0
/testbenchpipeall/writedata	32'h00000090
/testbenchpipeall/dataadr	32'h000012c4
/testbenchpipeall/memwrite	1'h1
/testbenchpipeall/dut/mips/clk	1'h1
/testbenchpipeall/dut/mips/reset	1'h0
/testbenchpipeall/dut/mips/pcF	32'h00000014
/testbenchpipeall/dut/mips/instrF	32'h00000000
/testbenchpipeall/dut/mips/dp/resultW	32'h00001200
/testbenchpipeall/dut/mips/dp/writedataM	32'h00000090
/testbenchpipeall/dut/mips/dp/clk	1'h1
/testbenchpipeall/dut/mips/c/clk	1'h1
/testbenchpipeall/dut/mips/c/reset	1'h0
/testbenchpipeall/dut/mips/c/opD	6'h00
/testbenchpipeall/dut/mips/c/functD	6'h00
/testbenchpipeall/dut/mips/c/flushE	1'h0
/testbenchpipeall/dut/mips/c/equalD	1'h1
/testbenchpipeall/dut/mips/c/memtoregE	2'h1
Now	0.000115 ms
Cursor 1	0.000052 ms

11. Jump register.

- ALU:

No additionals changes.

- Data path:

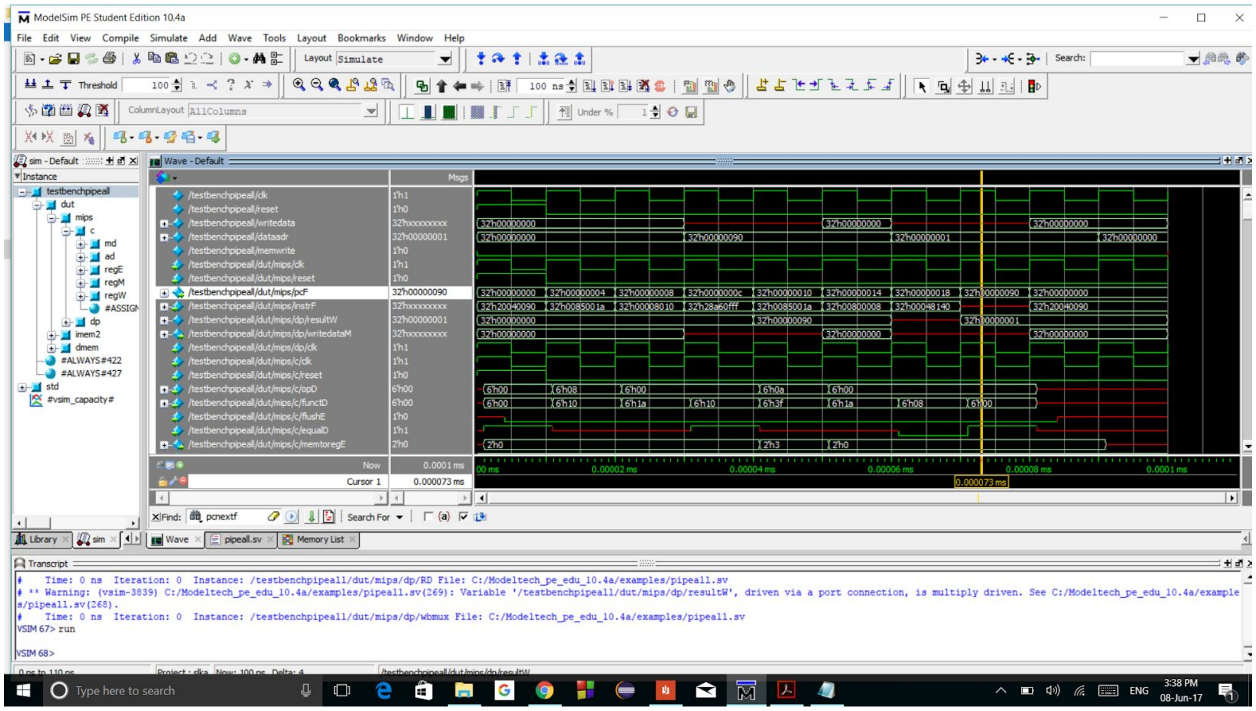
Connecting the source read from the register file to the pc.

- Controller:

An extra control signa jr to choose what value to be written in the pc.

Testing:

Jr a0 pc value to be fetched is 90



	Msgs
/testbenchpipeall/clk	1'h1
/testbenchpipeall/reset	1'h0
/testbenchpipeall/writedata	32'hxxxxxxxx
/testbenchpipeall/dataadr	32'h00000001
/testbenchpipeall/memwrite	1'h0
/testbenchpipeall/dut/mips/clk	1'h1
/testbenchpipeall/dut/mips/reset	1'h0
/testbenchpipeall/dut/mips/pcF	32'h00000090
/testbenchpipeall/dut/mips/instrF	32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/resultW	32'h00000001
/testbenchpipeall/dut/mips/dp/writedataM	32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/clk	1'h1
/testbenchpipeall/dut/mips/c/clk	1'h1
/testbenchpipeall/dut/mips/c/reset	1'h0
/testbenchpipeall/dut/mips/c/opD	6'h00
/testbenchpipeall/dut/mips/c/functD	6'h00
/testbenchpipeall/dut/mips/c/flushE	1'h0
/testbenchpipeall/dut/mips/c/equalD	1'h1
/testbenchpipeall/dut/mips/c/memtoregE	2'h0
Now	0.0001 ms
Cursor 1	0.000073 ms

```
#    Time: 0 ns  Iteration: 0  Instance: /testbenchpipeall/dut/mips/dp/RD File: C:/Modeltech_pe_edu_10.4a/examples/pipeall.sv
# ** Warning: (vsim-3839) C:/Modeltech_pe_edu_10.4a/examples/pipeall.sv(269): Variable '/testbenchpipeall/dut/mips/dp/resultW', driven via a port connection, is multiply driven. See C:/Modeltech_pe_edu_10.4a/example
s/pipeall.sv(268).
#    Time: 0 ns  Iteration: 0  Instance: /testbenchpipeall/dut/mips/dp/wbmux File: C:/Modeltech_pe_edu_10.4a/examples/pipeall.sv
VSIM 67> run

VSIM 68>
```

12. Jump and link.

- ALU:

No additionals changes.

- Data path:

Adding a mux to choose to write into destination register in regfile or in register 31.

Adding a mux to choose what result to be written in register file the result or pc+4.

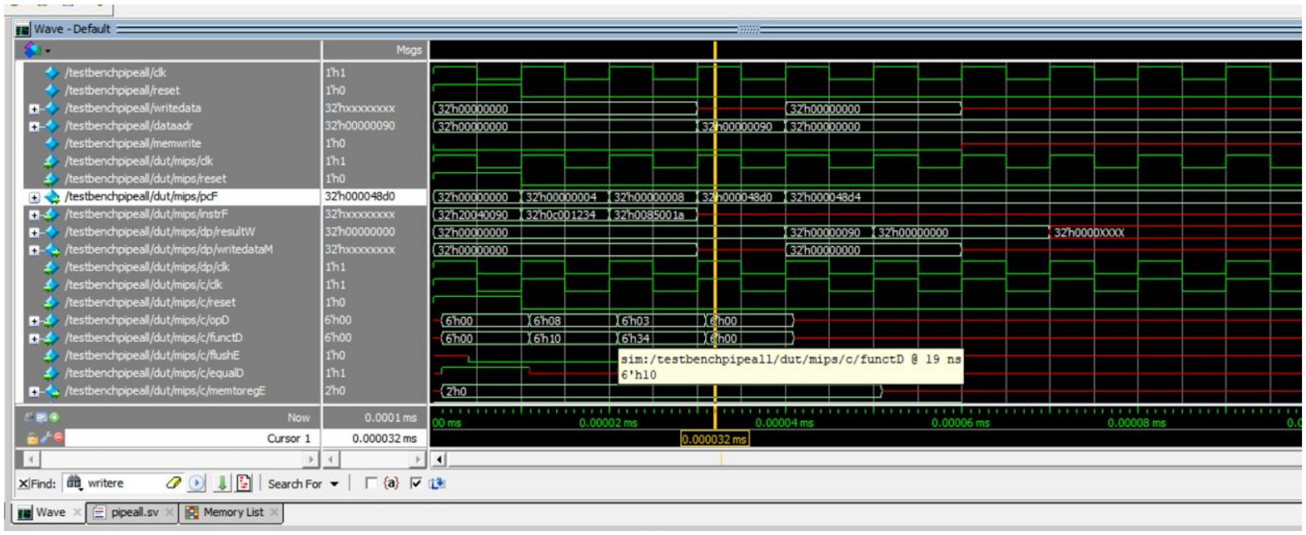
- Controller:

Two extra control signals 1 to choose what result is going to be written and 1 to choose what register is the result going to be written into.

Same control signals as jump except for jal and jalreg both are 1.

Testing:

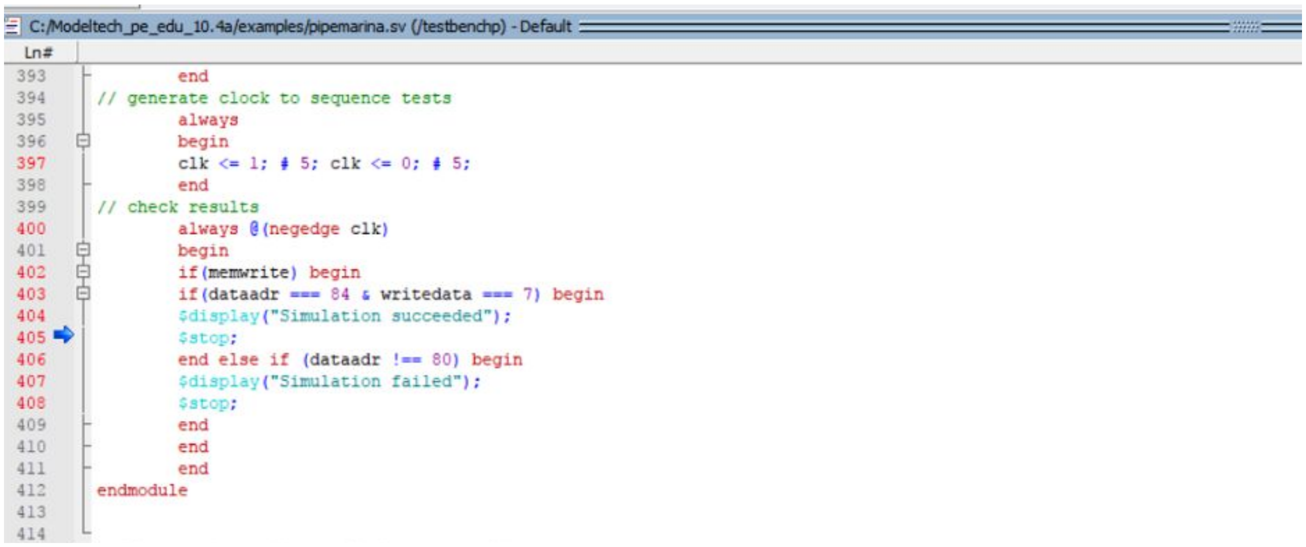
Jal 48d0



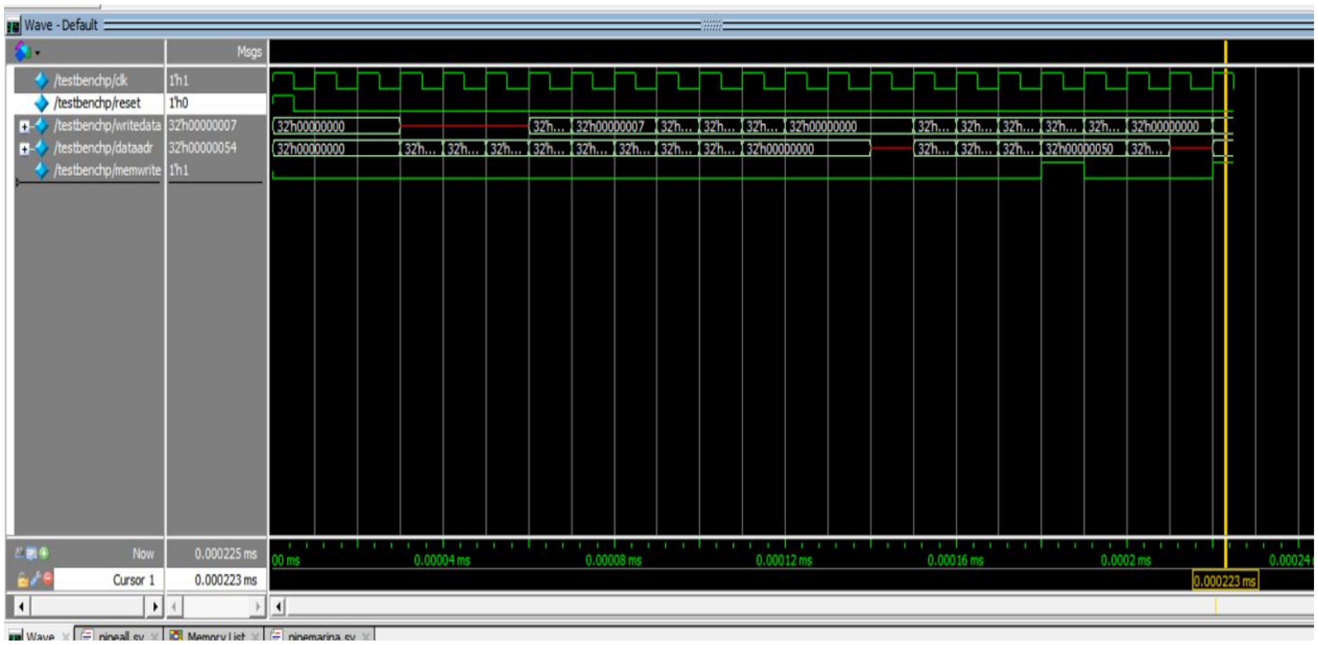
```
Wave - Default                                   Msgs
/testbenchpipeall/clk                            1'h1
/testbenchpipeall/reset                          1'h0
/testbenchpipeall/writedata                      32'hxxxxxxxx
/testbenchpipeall/dataadr                        32'h00000090
/testbenchpipeall/memwrite                       1'h0
/testbenchpipeall/dut/mips/clk                   1'h1
/testbenchpipeall/dut/mips/reset                 1'h0
/testbenchpipeall/dut/mips/pcF                   32'h000048d0
/testbenchpipeall/dut/mips/instrF                32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/resultW            32'h00000000
/testbenchpipeall/dut/mips/dp/writedataM         32'hxxxxxxxx
/testbenchpipeall/dut/mips/dp/clk                1'h1
/testbenchpipeall/dut/mips/c/clk                 1'h1
/testbenchpipeall/dut/mips/c/reset               1'h0
/testbenchpipeall/dut/mips/c/opD                 6'h00
/testbenchpipeall/dut/mips/c/functD              6'h00
/testbenchpipeall/dut/mips/c/flushE              1'h0
/testbenchpipeall/dut/mips/c/equalD              1'h1
/testbenchpipeall/dut/mips/c/memtoregE           2'h0
                                     Now         0.0001 ms
                                     Cursor 1    0.000032 ms
```

```
sim:/testbenchpipeall/dut/mips/c/functD @ 19 ns
6'h10
```

Testing memfile program:


C:/Modeltech_pe_edu_10.4a/examples/pipemarina.sv (/testbenchp) - Default

```
        end
// generate clock to sequence tests
        always
        begin
        clk <= 1; # 5; clk <= 0; # 5;
        end
// check results
        always @(negedge clk)
        begin
        if(memwrite) begin
        if(dataadr === 84 & writedata === 7) begin
        $display("Simulation succeeded");
        $stop;
        end else if (dataadr !== 80) begin
        $display("Simulation failed");
        $stop;
        end
        end
        end
endmodule
```


Wave - Default

```
                      Msgs
/testbenchp/clk       1'h1
/testbenchp/reset     1'h0
/testbenchp/writedata 32'h00000007
/testbenchp/dataadr   32'h00000054
/testbenchp/memwrite  1'h1

Now       0.000225 ms
Cursor 1  0.000223 ms
```