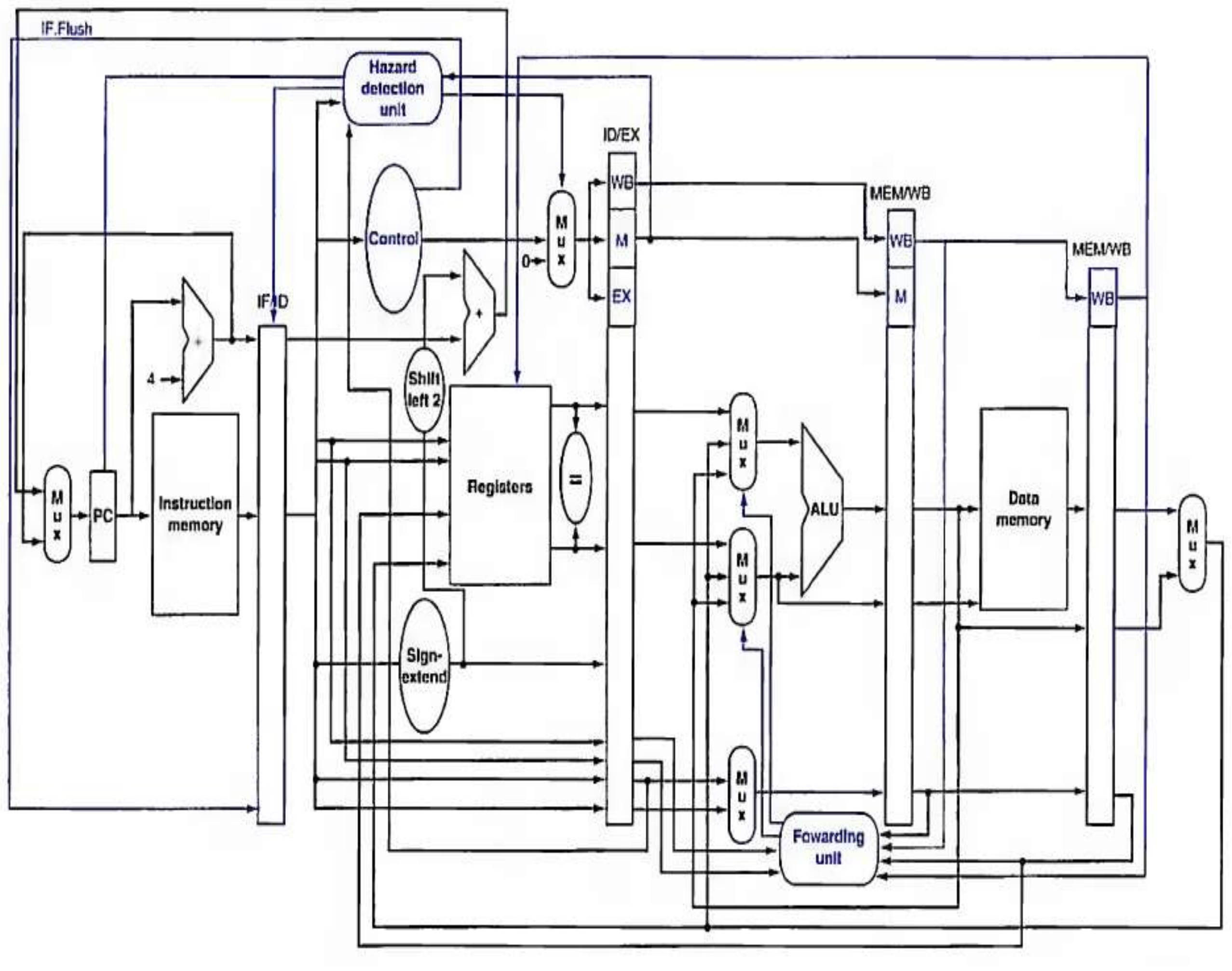
COMPUTER ARCHITECTURE
Pipelined Implementation of MIPS Architecture

Name: Mihir Vora
ID: 2018A3PS0755G

Circuit Diagram:



Different Modules and their function:

IF Stage:

1) **PC (Program Counter)** – Gives the address in the memory where the instruction to be executed is present.
2) **InstructionMemory** – In this module the instruction memory has been defined by storing some instructions in a few memory locations. It chooses the instruction to be executed depending on the address received from the Program counter.
3) **Adder (PCAdder) -** Calculates the address for the next instruction in line which may or may not be executed.
4) **Mux2x1_32Bits (nextPCMux)** – Decides whether the next instruction to be executed is PC + 4 or the instruction present at the branch address.
5) **IF_ID_reg** – This is the register that stores instruction read and PC+4 to be sent to the decoding stage in next clock cycle.

ID Stage:

1) **ControlUnit** – It takes opcode part of the instruction as input and generates all the control signals like ALUSrc, MemWrite, RegWrite, MemRead, Branch, RegDst, MemtoReg, ALUOp.
2) **RegisterFile** – Depending on the instruction it chooses appropriate register numbers and assigns the content of those registers to ReadData1 and ReadData2. Also chooses the appropriate register to write the result and does the work of storing the result into that register. Some registers have been assigned values in this module so as to use them while simulating some instructions.
3) **Mux3x1_32bits (comparatorMUX1)** – As the branch decision mechanism has been shifted to ID stage itself this mux decides whether the first operand for branch comparison should come from ReadData1 or should it be forwarded from EX or MEM stage of previous instructions.
4) **Mux3x1_32bits (comparatorMUX2)** – As the branch decision mechanism has been shifted to ID stage itself this mux decides whether the second operand for branch comparison should come from ReadData1 or should it be forwarded from EX or MEM stage of previous instructions.
5) **Comparator** – Compares the two operands from previous MUXs and sets or rested the equal flag which in turn is used to decide whether to branch or not.
6) **SignExtend** – Uses bits 0:15 from the instruction and extends that to a 32 bit number which is used as an offset in lw, sw, beq instructions.
7) **ShiftLeft2** – Used in calculation of the branch offset.
8) **Adder (branchAdder)** – Adds the correct offset with PC+4 to get the branch address.
9) **HazardDetectionUnit** - Takes the inputs as MemReadEX, MemReadMEM, rt_EX, instructionID , detect whether a hazard condition occurs and generates appropriate instructions to stall the process for 1 cycle.
10) **Mux2x1_10Bits (ID_EXRegMux)** – This takes the output from Hazard Unit and in case of a hazard makes the control signals to be sent to further stages zero.
11) **ID_EX_reg** – This is the register that holds and transfers the data like control signals, operand and destination registers, ReadData1, ReadData2, signExtendedResult to the execution stage.

EX Stage:

1) **Mux3x1_32Bits (ALUData1Mux) –** Depending on the result from the forwarding unit, chooses between ReadData1 and the results from EX and WB stage from previous instruction and sends the appropriate first operand to the ALU.
2) **Mux3x1_32Bits (ALUData2Mux_1) –** Depending on the result from the forwarding unit, chooses between ReadData1 and the results from EX and WB stage from previous instruction and sends the appropriate second operand further.
3) **Mux2x1_32Bits (ALUData2Mux_2) –** Selects between the operand received from the previous Mux and signExtended_out on the basis of the control signal ALUSrc.
4) **ALUControl –** On the basis of ALUOp control signal and the function field of the instruction generates specific ALU Control signals which will decide which operation is to be performed by the ALU.

5) **ALU32Bit** – It takes two operands and gives a particular output ALUResult. The function performed on the operands depends on the signals generated by AluControl. It also gives another output(zero) which is used to implement the beq instruction.
6) **Mux2x1_5Bits (regDstMux) –** Decides whether the destination register is present in the rt field or rd field of the instruction based on the RegDst control signal and transfers it forward.
7) **EX_MEMReg** - This is the register that holds and transfers the data like a few control signals, information about the write register, ALUResult, Write data to the Memory stage.
8) **ForwardingUnit –** This module takes inputs like EX_MemWriteReg , Mem_WbWriteReg, ID_Ex_Rs, ID_Ex_Rt, EX_MemRegwrite, Mem_WbRegwrite; decides whether there is a data hazard anywhere and generates appropriate forwarding signal to counter it.

MEM Stage:

1) **DataMemory** - Used to write to or read from a particular memory location during lw and sw instructions. The memory address to access is given by ALUResult and whether to read or write data is decided by MemRead and MemWrite control signals.
2) **Mem_WbReg** - This is the register that holds and transfers the data like a few control signals, information about the read data, ALUResult, destination register to the Memory stage.

WB Stage:

1) **Mux2x1_32Bits (writeBackMux) –** Here it is decided whether the output of ALU or the read data from memory is to be written to the register by MemtoReg control signal.

MipsPipelineTestBech : Connects all these modules and also is the test bench for simulation. For the test bench the inputs are the reset and Clock pulses.

Instruction Memory:

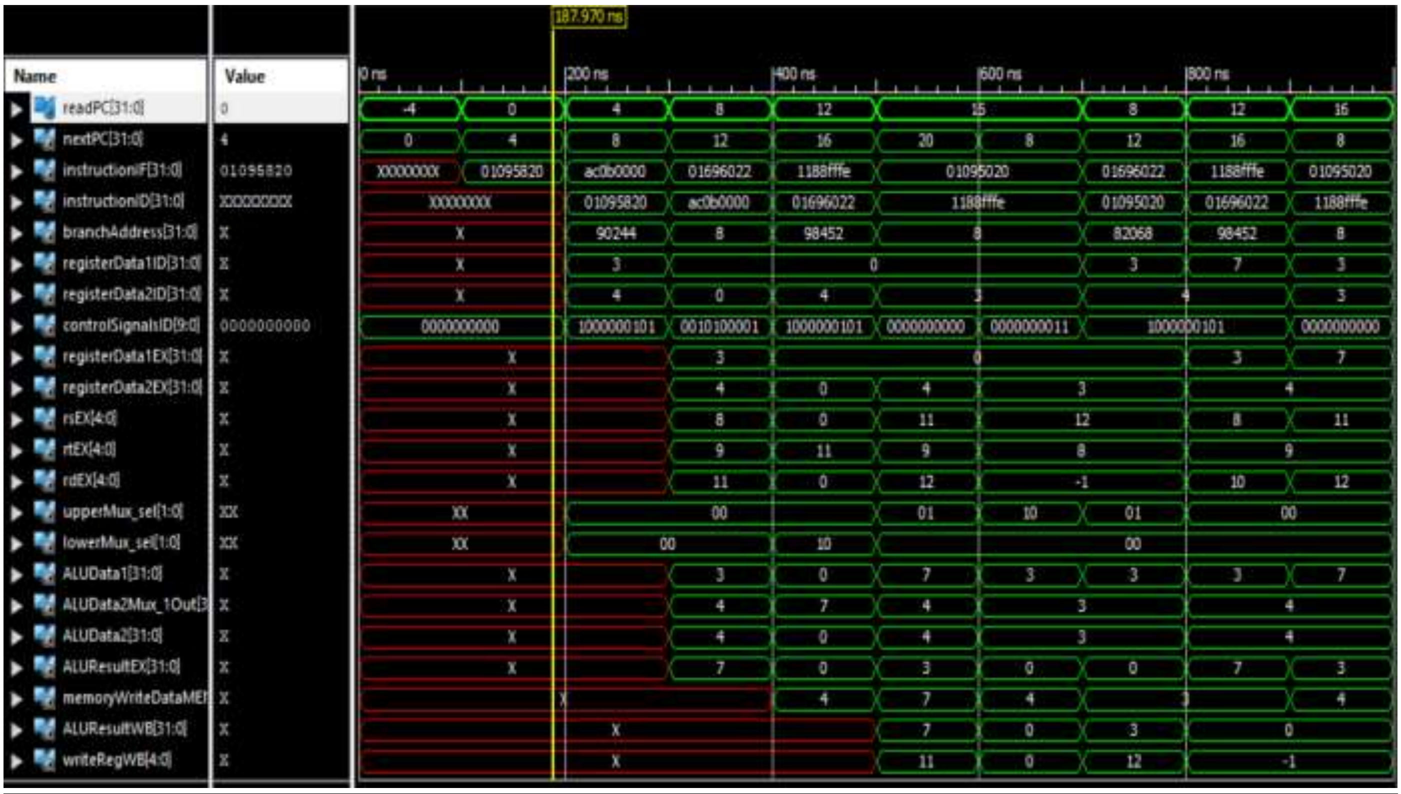
```
Imemory[0] = 32'h01095820; //add $t3 $t0 $t1

Imemory[1] = 32'hAC0B0000; //sw $t3 0($zero) //Hazard here

Imemory[2] = 32'h01696022; //sub $t4 $t3 $t1 //Hazard here

Imemory[3] = 32'h1188FFFE; //beq $t4 $t0 -2  //Hazard here

Imemory[4] = 32'h01095020; //add $t2 $t0 $t1
```

Register values:

```
memory[0] <= 32'h00000000; // $zero

memory[8] <= 32'h00000003; //$t0

memory[9] <= 32'h00000004; //$t1

memory[10] <= 32'h00000005;//$t2

memory[11] <= 32'h00000000;//$t3
```

```
            memory[12] <= 32'h00000000;//$t4
            memory[13] <= 32'h00000000;//$t5
            memory[14] <= 32'h00000000;//$t6
            memory[15] <= 32'h00000000;//$t7
```

Simulation Results:



Results Explained:

Initially a reset pulse is given which sets readPC to -4 and Control Signals to 0.

1st positive clock edge - Instruction 32'h01095820 (add $t3 $t0 $t1) is fetched and is now in the IF stage.

2nd positive clock edge - Instruction 32'hAC0B0000 (sw $t3 0($zero)) is fetched and is now in the IF stage.

add $t3 $t0 $t1 goes into ID stage. There the data from $t0 and $t1 is read which is 3 and 4. And control signals for add are generated.

3rd positive clock edge – Instruction 32'h01696022 (sub $t4 $t3 $t1) is fetched and is now in the IF stage.

sw $t3 0($zero) is in the ID stage. Data from $zero and $t3 register is taken which is 0 for both.

add $t3 $t0 $t1 goes into the EX stage. Here the two operands 3 & 4 are added and the result 7 is generated.

4th positive clock edge – Instruction 32'h1188FFFE (beq $t4 $t0 -2) is fetched and is now in IF stage.

sub $t4 $t3 $t1 is in the ID stage. It reads data from $t3 and $t1 which is 0 and 4 respectively; as add $t3 $t0 $t1 instruction hasn't written back the result 7 to $t3 yet.

sw $t3 0($zero) is in the EX stage. Here ALUData1 which is from $zero register can be seen which is zero. For ALUData2Mux1_out which should have the value to be written to memory location, it can be seen that it has 7 now which we get by forwarding the result of the add $t3 $t0 $t1 from Mem stage. ALUData2 shows zero which is the offset added to calculate address.

add $t3 $t0 $t1 is in the memory stage. The memoryWriteDataMEM shows 4 which is the value of rt register but this value won't be written as MemWrite control signal is off for R-type instructions.

5th positive clock edge – Instruction 32'h01095020 (add $t2 $t0 $t1) is fetched and is in IF stage.

beq $t4 $t0 -2 is in the decode stage. Processor detects that a hazard as occurred as the value of $t4 hasn't be obtained yet from the subtract instruction. So the processor will be stalled for next cycle. And Control signal 0's have been generated.

sub $t4 $t3 $t1 is in the EX stage. Here it will obtain the value of $t3 using the forwarding unit as the add $t3 $t0 $t1 is still in the WB stage. We can see here that ALUData1 has 7 which is obtained by forwarding and ALUData2 has 4 which is value of register $t1. Subtraction is done and ALUResult 3 is obtained.

sw $t3 0($zero) is in the memory stage. The memoryWriteDataMEM shows 7 which is the value to be written to the memory location.

add $t3 $t0 $t1 is in the WB stage and the value 7 is written in register 11 which is $t3.

6th positive clock edge – As this cycle was stalled same instruction is in IF and ID stage. Now in the ID stage the value of $t4 after sub $t4 $t3 $t1 is obtained by forwarding. This is compared with $t0 which also has 3. Branch address is calculated as 8. And the value of next PC can be seen changing to 8 instead of 24.

sub $t4 $t3 $t1 will be in the memory stage. Nothing has to be done as it as an R-type instruction.

sw $t3 0($zero) is in the WB stage where again nothing has to be done as it is sw instruction.

7th positive clock edge – As the read PC becomes 8 in this cycle 32'h01696022 (sub $t4 $t3 $t1) is fetched again and is now in the instruction decode stage.

Hence this Pipelined MIPS implementation shows execution of R-type, sw and beq instructions while giving examples of dealing with Hazards.