Name: Suraj Salian ID:2018AAPS0253G

MIPS PIPELINED ARCHITECTURE

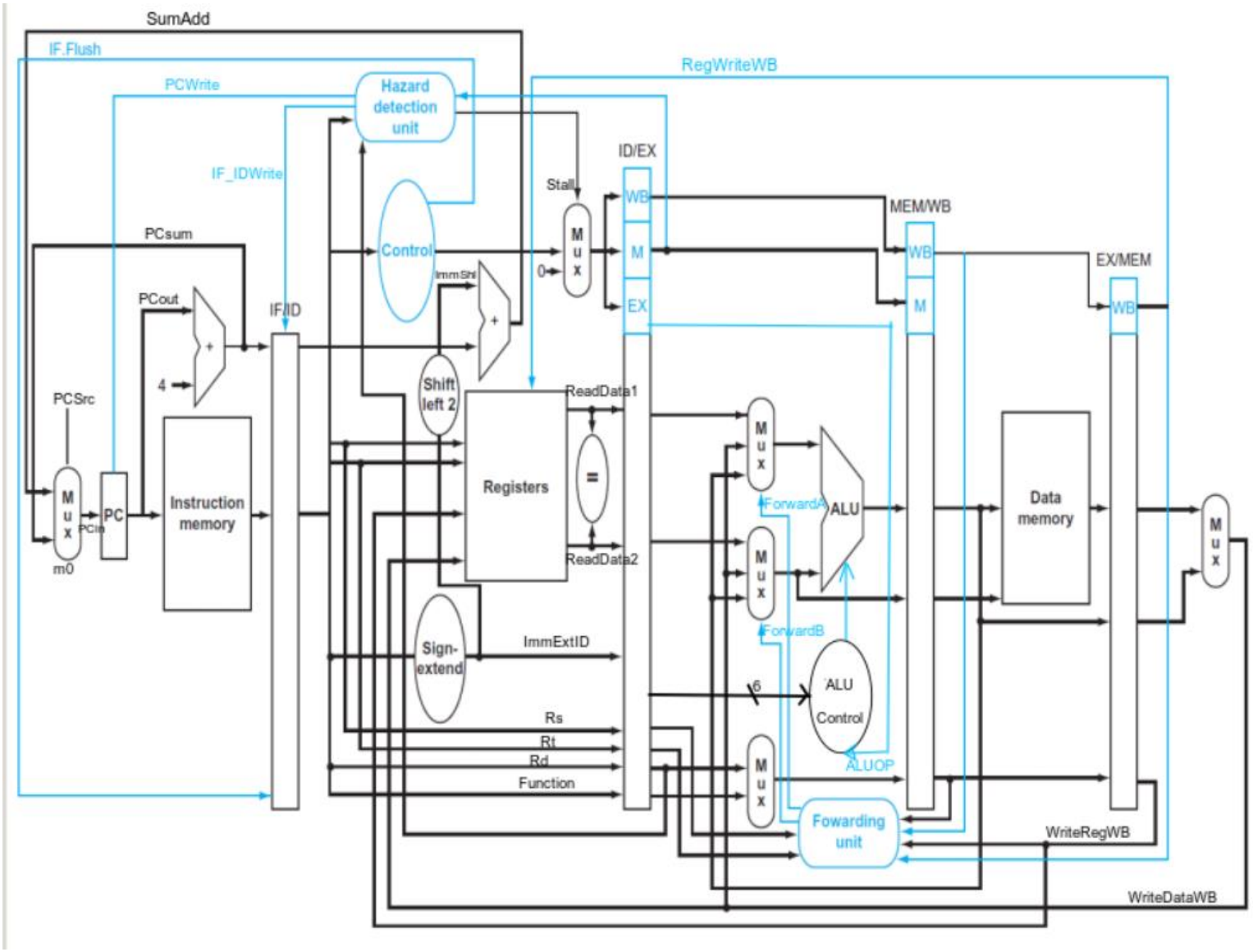
Modules Developed:

1. 32-bit Adder
2. ALU Control Unit
3. ALU
4. Branch Comparator
5. Control Signal Mux
6. Main Control Unit
7. Data Memory Unit
8. Instruction Memory Unit
9. Parameterized Mux
10. Program Counter Unit
11. Forwarding Unit
12. Hazard Detection Unit
13. Register File
14. Shift Left By 2
15. Sign Extension Unit
16. IF/ID Register
17. ID/EX Register
18. EX/MEM Register
19. MEM/WB Register
20. MIPS Core
21. MIPS Core Testbench

Assumptions:

1. A 5-stage Pipeline of MIPS architecture is implemented which consists of Instruction Fetch, Instruction Decode, Execution, Memory and Write-Back Stage.
2. Only 11 out of the complete MIPS ISA. These instructions include-
    a. NOP
    b. **R-Type***: AND, OR, ADD, SUB, SLT, NOR, XOR, LW, SW*
    c. **Branch:** *BEQ*
3. On boot, the Global Reset Signal is ON for one clock cycle to initialize the system (PC = 0).
4. Instruction Memory has 128 Memory Locations each memory location is 8-bit wide. Making the memory byte organized.
5. Data Memory has 512 Memory Locations each memory location is 8-bit wide.
6. Whenever the the Program Counter is out of bound i.e greater than 128 bytes. No Instruction fetching takes place, to go back to initial state Reset should be made ON.
7. ***For running the simulation is necessary to place the ‘Data.mem’, ‘REGINP.mem' and ‘Instruction.mem’ files in the project directory/ Simulation Directory.
8. The Type of Branch Prediction Implemented here is Static Prediction of Branch Not taken. In case when branch is taken the pipeline is flushed and execution starts from branch address
9. Forwarding Unit has been implemented and used to forward data from MEM/WB stage to Execution stage.
10. Load Use Case Hazard is handled by inserting one bubble. In order to get the correct data from Memory Instructions.
11. Data Dependent Branch Instruction is not implemented in this design. (i.e) Separate forwarding unit is not implemented for branch instructions.

Pipelined Datapath Implementation:



Modules:

1. **32-bit Adder:** Performs the Add Operation.
   First Instance**: 'a0'** Used for incrementing the Program Counter by 4 in the IF stage.
   Second Instance**: 'a1'** Used while executing branch instruction to calculate the Program Counter from the relative address in the ID stage.

2. **ALU Control Unit:** Generates the ALU Control Signals required by the ALU for execution.

3. **ALU:** Performs all important task- R-type Instruction in the EX-stage. Instructions implemented here depend on the control signal receive from ALU Control Unit.

4. **Branch Comparator:** This unit is present in the ID stage and used to check the equality of two register contents for branch instructions.

5. **Main Control Unit:** Generates the required control signals from the instruction received from the Instruction Memory. Control signals generated by this unit are- RegDst, Branch, ALUsrc, MemRead, MemWrite, MemtoReg, ALUOP, RegWrite

6. **Control Signal Mux:** This Mux is implemented in the decode stage. The control signal to this mux is the stall signal. On detection of hazard "nop' control signals are directed by this mux.

7. **Data Memory:** This Unit contains 512 bytes RAM for storing data during the operation of CPU. The memory is used for Load word and Store word Instructions. The ROM is byte organized. In this design data is written in "Data.mem" file which is used by the Data memory unit at the time of simulation.

8. **Instruction Memory:** This unit contains 128 bytes ROM which contains the instruction to be executed by the CPU.In this design instructions are written in "Instruction.mem" file which is used by the Instruction memory unit at the time of simulation.

9. **Parameterized Mux:** This single parameterized MUX is instantiated according to required width as per the block diagram.

10. **Program Counter:** Global Clock and Global Reset are given to this unit. On reset the PC is made zero. Else on every clock edge the calculated correct value of Program Counter is written on to the PC.

11. **Forwarding Unit:** This unit is used handle data dependency between any two instructions. This units is implemented in the execution stage and generate two control Signals ForwardA and ForwardB, which used to select the correct forwarded data.

EX Hazard Logic:

if (EX/MEM.RegWrite) && (EX/MEM.RegisterRd ≠ 0) && (EX/MEM.RegisterRd = ID/EX.RegisterRs))

ForwardA = 10

if (EX/MEM.RegWrite && (EX/MEM.RegisterRd ≠ 0) && (EX/MEM.RegisterRd = ID/EX.RegisterRt))

ForwardB = 10

MEM Hazard Logic:

if (MEM/WB.RegWrite) && (MEM/WB.RegisterRd ≠ 0) && !(EX/MEM.RegWrite && (EX/MEM.RegisterRd ≠ 0) && (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)) && (MEM/WB.RegisterRd = ID/EX.RegisterRs))

ForwardA = 01

if (MEM/WB.RegWrite) && (MEM/WB.RegisterRd ≠ 0) && !(EX/MEM.RegWrite && (EX/MEM.RegisterRd ≠ 0) && (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt)) && (MEM/WB.RegisterRd = ID/EX.RegisterRt))

ForwardB = 01

Else //In all other cases

ForwardA = 00; ForwardB = 00

12. **Hazard Detection Unit :** This unit is used load use case dependency between instructions. This units is implemented in the decode stage and generate three control Signals PCWrite, IF/IDWrite and Stall. When a load use case dependency is detected PCWrite, = 0, IF/IDWrite = 0 and Stall = 1,in this way a bubble is inserted in the pipeline. While all other cases PCWrite, = 1, IF/IDWrite = 1 and Stall = 0.

    Hazard Detection Logic:

    if (ID/EX.MemRead) && ((ID/EX.RegisterRt = IF/ID.RegisterRs) || (ID/EX.RegisterRt = IF/ID.RegisterRt)))

        PCWrite, = 0;, IF/IDWrite = 0; Stall = 1;

    Else

        PCWrite, = 1;, IF/IDWrite = 1; Stall = 0;

13. **Register File:** This unit contains two register files one for normal instruction execution and one register file for floating point instructions. Each Register file contains 32 registers. The Register file is used to read the registers and perform write operation on clock-edge.

14. **Shift Left by 2:** This unit is used to shift the immediate data left by 2. (Used by the branch instructions)

15. **Sign Extension Unit:** This unit sign extends the 16-bit immediate data which is the part of Instruction to 32 bit. (Used by LW, SW, Branch and other Immediate Instructions)

16. **IF/ID:** This Pipeline register is present at the junction of Fetch and Decode stage. At clock edge data from fetch stage is directed to the Decode stage.

17. **ID/EX:** This Pipeline register is present at the junction of Decode and Execute stage. At clock edge data from decode stage is directed to the execute stage.

18. **EX/MEM:** This Pipeline register is present at the junction of Execute and Memory stage. At clock edge data from Execute stage is directed to the Memory stage.

19. **MEM/WB:** This Pipeline register is present at the junction of Memory and Writeback stage. At clock edge data from Memory stage is directed to the Write stage.

20. **MIPS Core:** The main module of Single Cycle architecture. This unit completes the required connection with different blocks.

21. **MIPS Core Testbench:** This module provides the Clock of time period 100ns to the MIPS core. This also provides the global clock to the MIPS core.

Control Signal Analysis:

1. **RegDst:** Given as control input to MUX 'm2'. This signal is used to select the destination register from decoded instruction. Inputs to Mux 'm0' are 'Rt' and 'Rd' respectively.
2. **RegWrite:** This control signal is high only when the data is to written in the destination register.
3. **Branch:** Given as control input to and gate. This used to select the generate the PCSrc signal. This signal is active only during Branch Instruction.
4. **MemRead:** The given control signal is active only during memory access instruction (Memory Read Instruction) Eg- LW.
5. **MemWrite:** The given control signal is active only during memory write operation. (Eg- SW)
6. **MemtoReg:** Given as control input to MUX 'm3'. This selects the data to be written in register file. This signal is active only during memory access Instruction. i.e When this signal in ON the data from memory is written onto the destination register
7. **ALUsrc:** Given as control input to MUX 'm1'. This selects the second input given to ALU. This signal is active only during LW, SW Instruction.
8. **ALUOP:** 2-bit control signal given to ALU Control for given execution of Instructions. ALUOP for given ALU instructions
    a. R-Type 2'b10
    b. LW 2'b00
    c. SW 2'b00
    d. BEQ 2'b10

Other Control Signals:

1. **ALUCT:** -4-bit control signal given to ALU for given execution of Instructions. ALUCT for given ALU instructions
    a. AND 4'b0000
    b. ADD 4'b0010
    c. OR 4'b0001
    d. SUB 4'b0110
    e. SLT 4'b0111
    f. NOR 4'b1100
    g. XOR 4'b1101

Instruction Analysis:

1) **NOP:** Processor remains in the same state.
   Instruction: 32'h0
   Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 0
Memtoreg = 0	ALUsrc = 0
RegWrite = 0	ALUOP = 00
ALUCT = 00	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

2) **AND:**

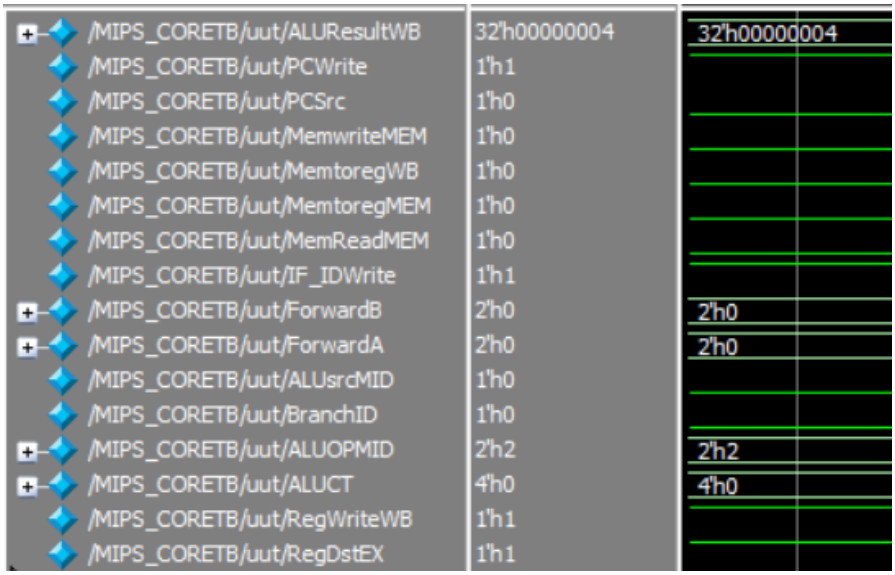
```
and rd, rs, rt
```

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x24
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 2'b10
ALUCT = 0000	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 00A63824 // Instructions - AND $a3 $a1 $a2
$a1 = 0x05 $a2 = 0x06 $a3 =ALUResult= 0x5 & 0x6 = 0x4



3) **OR:**

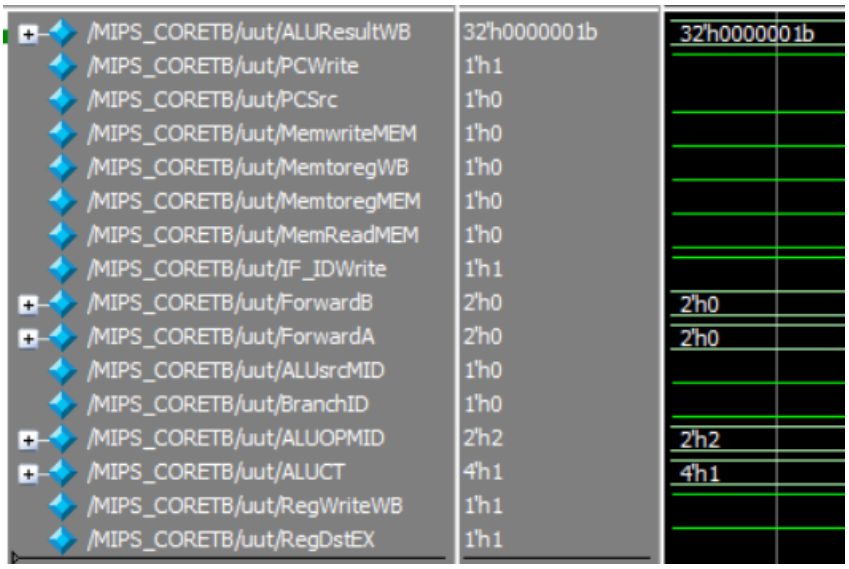
```
or rd, rs, rt
```

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x25
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 2'b10
ALUCT = 0001	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 00E84825 // Instructions - OR $t1 $a3 $t0
$a3 = 0x3 $t0 = 0x18 $t1 = ALUResult = 0x1B



4) ADD:

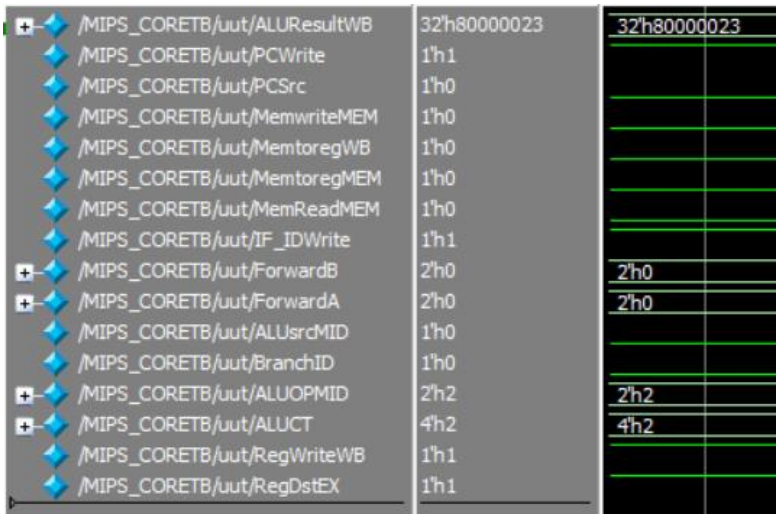
```
add rd, rs, rt
```

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x20
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 10
ALUCT = 0010	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 02138820 // Instructions - ADD $s1 $s0 $s3
$s0 = 0x10 $s3 = 0x80000013 $s1 =ALUResult= 0x10 + 0x80000013= 0x80000023



5) SUB:

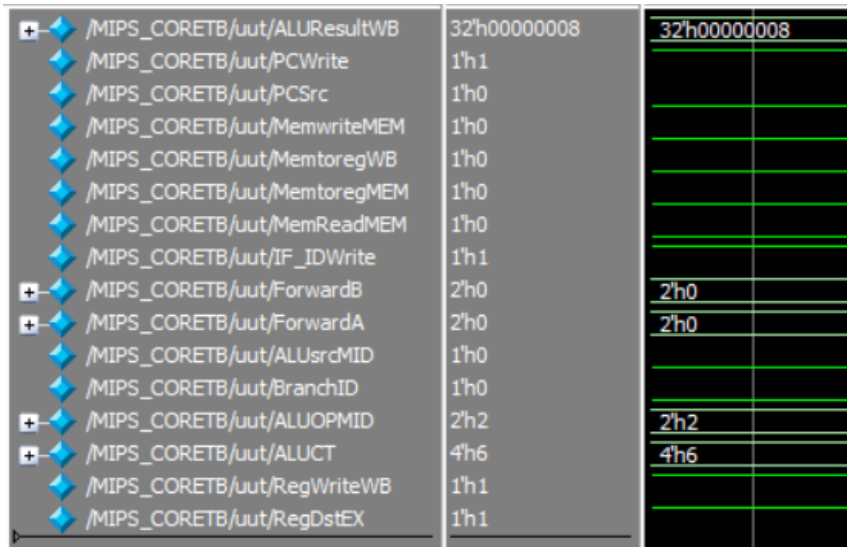
`sub rd, rs, rt`

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x22
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 10
ALUCT = 0110	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 01108822 // Instructions - sub $s1 $t0 $s0
$t0 = 0x18 $s0 = 0x10 $s1 =ALUResult= 0x08



6) SLT:

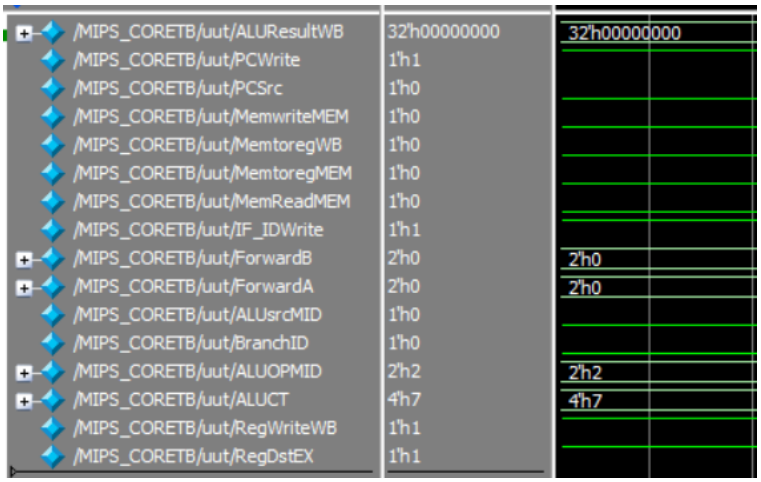
```
slt rd, rs, rt
```

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x2a
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 10
ALUCT = 0111	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 0253A02A // Instructions - slt $s4 $s2 $s3
$s2 = 0x12 $s3 = 0x80000013 $s4 =ALUResult= 0x0F < 0x 80000013 ? = 0x00



7) NOR:

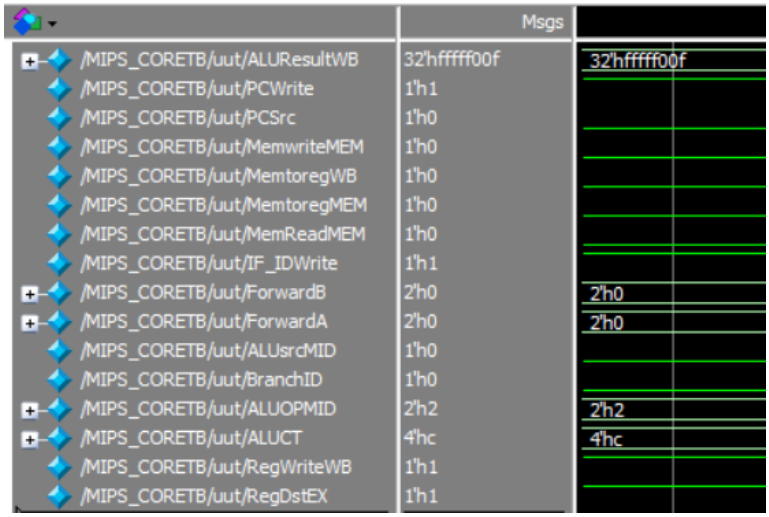
```
nor rd, rs, rt
```

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x27
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 10
ALUCT = 1100	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 01495827 // Instructions - NOR $t3 $t2 $t1
$t2 = 0x F00 $t1 = 0x F0 $t3 = ALUResult = 0x FFFFFFFFFFFFF00F



8) XOR:

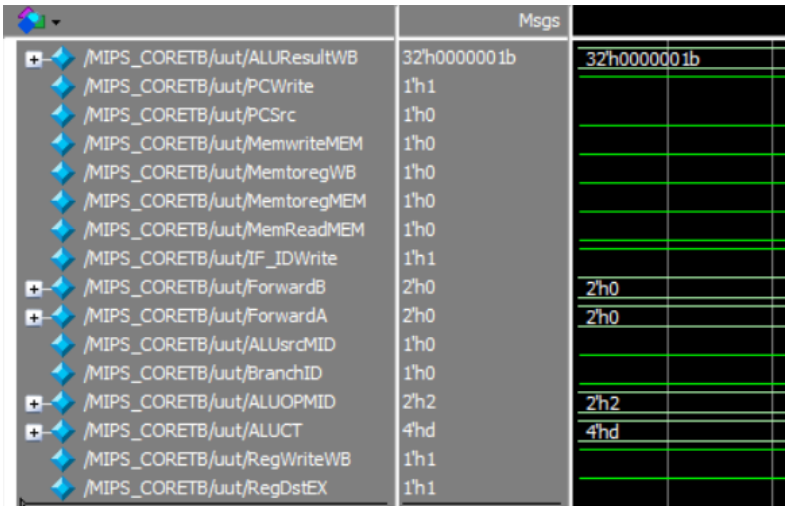
```
xor rd, rs, rt
```

Opcode	Rs	Rt	Rd	Shamt	Function
0	rs	rt	rd	0	0x26
6	5	5	5	5	6

Control Signals Generated:

Memread = 0	Branch = 0
Memwrite = 0	RegDst = 1
Memtoreg = 0	ALUsrc = 0
RegWrite = 1	ALUOP = 10
ALUCT = 1101	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 00E84826 // Instructions - XOR $t1 $a3 $t0
$a3 = 0x3 $t0 = 0x18 $t1 = ALUResult = 0x1B



9) LW:

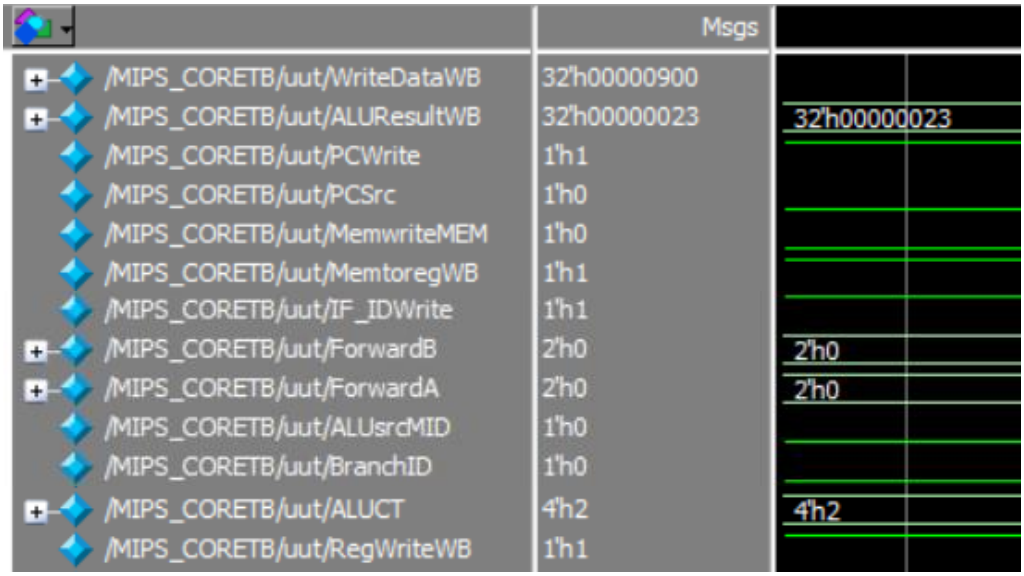
```
lw rt, address
```

0x23	rs	rt	Offset
6	5	5	16

Control Signals Generated:

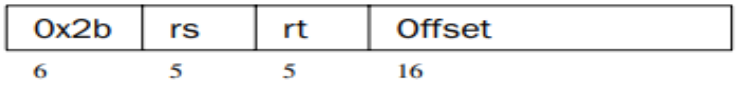
Memread = 1	Branch = 0
Memwrite = 0	RegDst = 0
Memtoreg = 1	ALUsrc = 0
RegWrite = 1	ALUOP = 00
ALUCT = 0010	PCWrite = 0/1
IF/IDWrite = 0/1	Stall = 0/1
PCSrc = 0	IF.FLUSH = 0

Eg: Hex- 8C620020 // Instructions - lw $v0 0x0020 $v1
$v1 = 0x3, ; Load Address= ALU Result = 0x3 + 0x20 = 0x23
Data @ 0x23 = 0x900
MemReadData= WriteData: 0x900



10) SW:

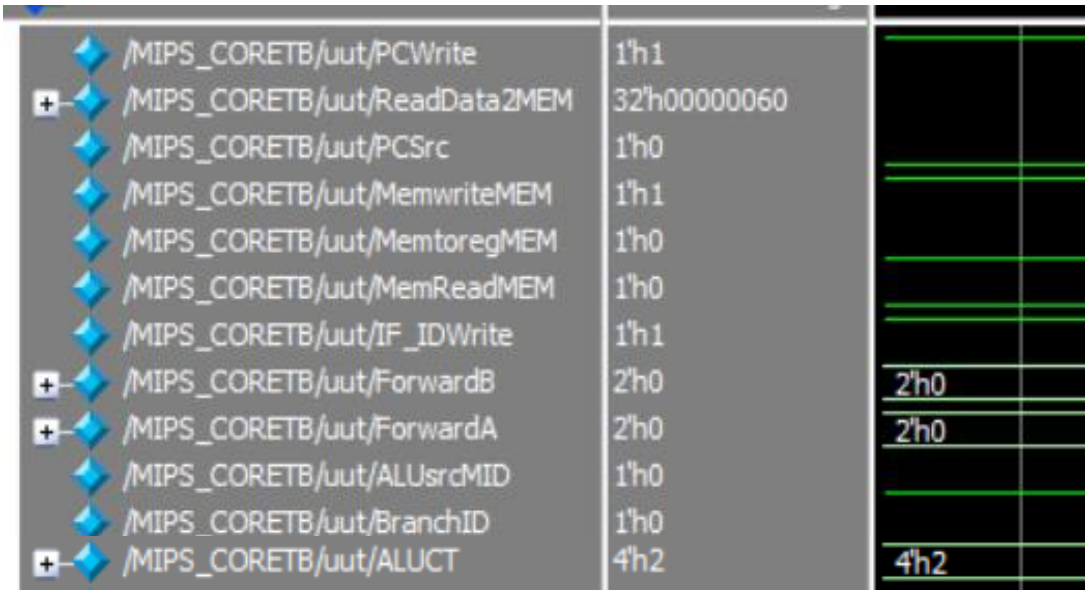
```
sw rt, address
```



Control Signals Generated:

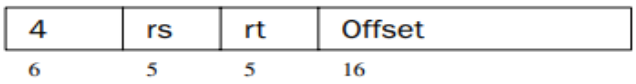
Memread = 0	Branch = 0
Memwrite = 1	RegDst = 0
Memtoreg = 0	ALUsrc = 1
RegWrite = 0	ALUOP = 00
ALUCT = 0010	PCWrite = 1
IF/IDWrite = 1	Stall = 0
PCSrc = 1	IF.FLUSH = 0

Eg: Hex- AFEF0004 Instruction - SW $t7 0x0004 $ra
$ra = 0x1F, $t7 = 0x60 ; Store Address= ALU Result = 0x1F + 0x04 = 0x23
Read Data2= Write Data: 0x60



11) BEQ:

```
beq rs, rt, label
```



Control Signals Generated:

Memread = 0	Branch = 1
Memwrite = 0	RegDst = 0
Memtoreg = 0	ALUsrc = 0
RegWrite = 0	ALUOP = 00
ALUCT = xxxx	PCWrite = 0/1
IF/IDWrite = 0/1	Stall = 0
PCSrc = 0/1	IF.FLUSH = 0/1

Eg: Hex- 10670016 // Instructions - BEQ $v1 $a3 0x0016
$a3 = 0x3 $v1=0x3 PCin = 0x00+0x4+(0x16 << 2)= 0x5C