PROJECT-3 REPORT

The design flow for the 32-bit MIPS. However, at first, the instruction set of the MIPS Processor is as follows:

1. ADD rd, rs, rt: Reg[rd] = Reg[rs] + Reg[rt].

2. BNE rs, rt, imm16: if (Reg[rs] != Reg[rt]) PC = PC + 4 + Sign_ext(Imm16)<<2 else PC = PC + 4

3. J target: PC = { PC[31:28], target, 00 }.

4. JR rs: PC = Reg[rs].

5. LW rt, imm16(rs): Reg[rt] = Mem[Reg[rs] + Sign_ext(Imm16)].

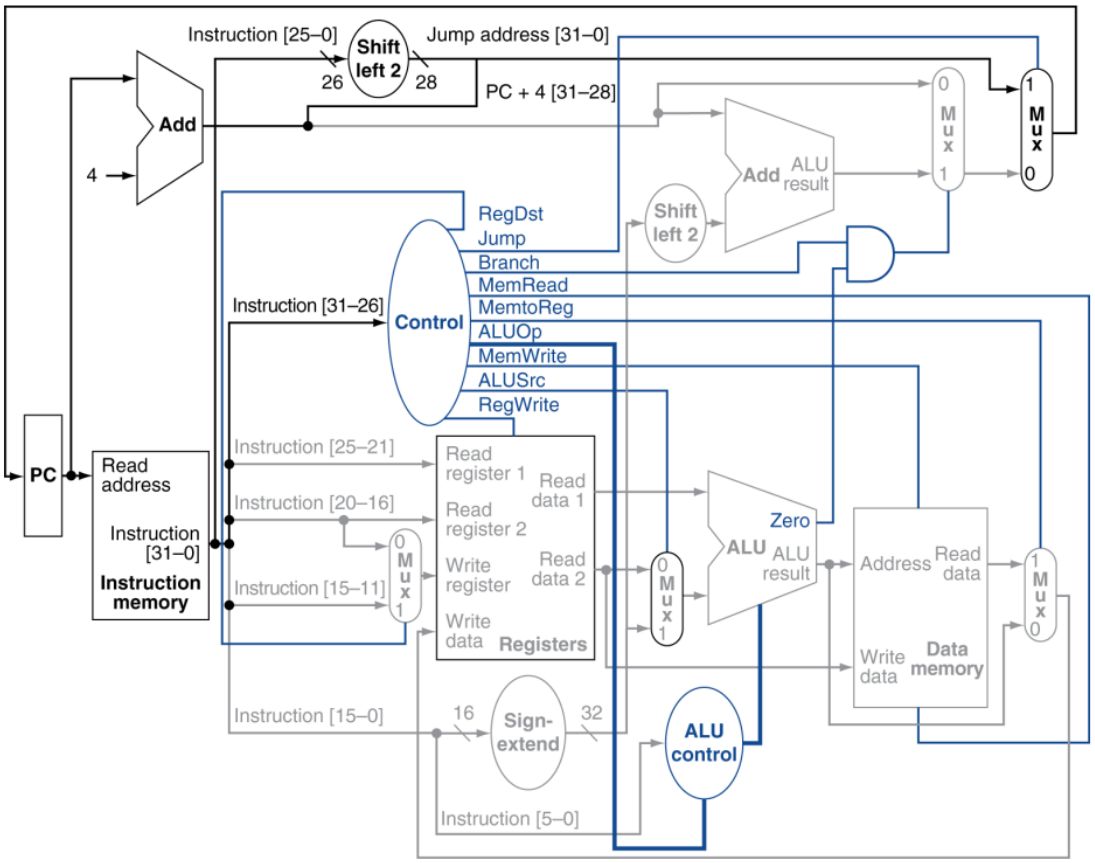
6. SLT rd, rs, rt: If (Reg[rs] < Reg[rt]) Reg[rd] = 00000001 else Reg[rd] = 00000000.

7. SUB rd, rs, rt: Reg[rd] = Reg[rs] – Reg[rt].

8. SW rt, imm16(rs): Mem[Reg[rs] + Sign_ext(Imm16)] = Reg[rt].

9. XORI rt, rs, imm16: Reg[rt] = Reg[rs] XOR Zero_ext(Imm16).

32-bit MIPS architecture datapath



Module designing

1. I started with making instruction memory module that stores and retrieves instructions based on the provided address, and a test stimulus module to verify its functionality by generating memory addresses and observing the fetched instructions.
   I named that module as “InstructionMem.v”
2. Created a register file module with 32 registers. It includes a decoder, multiplexers, and D flip-flops to perform read and write operations on the registers based on control signals.
   I named that module as “regfile.v”
3. After that I created a 32bit Adder as module “Add.v”
4. designed 32-bit Arithmetic Logic Unit (ALU) module. It performs various arithmetic and logical operations based on the ALU control signals. The ALU consists of 32 instances of a 1-bit ALU module, where each instance performs calculations for a corresponding bit position. The outputs include the ALU result, carry-out, zero flag, overflow flag, and negative flag.
   I named that module as “alu.v”
5. Designed a data memory module that stores and retrieves 32-bit data values. It uses a 128-bit array to represent the memory, with each element storing 32 bits of data. The module supports write and read operations based on the control signals and clock signal.
   I named that module as “dataMem”
6. Control Unit is designed for a 32-bit 5-stage Pipelined MIPS Processor

- It takes a 6-bit opcode (Opcode) as input and generates various control signals for the different stages of the processor pipeline.
- The control signals include:

  RegDst: Determines the destination register for register write operations.
  ALUSrc: Selects the second ALU operand (either a register or an immediate value).
  MemtoReg: Specifies whether the data read from memory should be written to a register.
  RegWrite: Enables register write operation.
  MemRead: Enables memory read operation.
  MemWrite: Enables memory write operation.
  Branch: Enables branching based on the comparison result.
  ALUOp: Determines the ALU operation based on the opcode.
  Jump: Enables a jump instruction.
  SignZero: Specifies whether the immediate value should be sign-extended or zero-extended.

- The control signals are assigned values based on the opcode using a casex statement, which matches the opcode with specific cases and assigns the corresponding values to the control signals.

- If the opcode doesn't match any specified cases, the default values are assigned to the control signals.
- The module provides the assigned values of the control signals as outputs.

7. Forwarding Unit:
   Forwarding Unit is designed to solve the data hazards in pipelined MIPS Processor. The correct data at the output of the ALU is forwarded to the input of the ALU when data hazards are detected. Data hazards are detected when the source register (EX_rs or EX_rt) of the current instruction is the same as the destination register (MEM_WriteRegister or EX_WriteRegister) of the previous instruction.
   I named that module as “ForwardingUnit.v”
8. After adding the forwarding unit to solve the data hazard, the 2x32 to 32 multiplexers at the input of the ALU become 3x32 to 32 multiplexers.
   So designed “mux3x32to32.v” module.
9. Finally designed my main module as “mips_32.v”
   - 32-MIPS Processor implemented in Verilog. It consists of modules and components that perform various tasks required for executing MIPS instructions.

   - The processor includes stages such as Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Each stage has its own set of registers and operations.

   - modules for key components of the processor, such as the Control Unit, Register File, ALU (Arithmetic Logic Unit), Data Memory, and Instruction Memory. These modules are interconnected to enable the flow of instructions and data through the pipeline.

   - Control signals are used to control the flow of data and operations within the processor. These signals include RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp, Jump, and SignZero. They are generated based on the opcode of the current instruction.

   - Module also includes additional features such as forwarding units for resolving data hazards, handling branch instructions, jumping to different locations in the program, and implementing the JR (Jump Register) instruction.

10. Designed a testbench to test my mips_32 module with clk = 0; rst = 0; #2 rst = 1;

No	Check sheet item	Answer/Done?
1.	How many instructions are meaningful?	Y
2.	Decode the necessary instructions only	Y
3.	Illustrate the instruction order	Y
4.	Final values that are stored in the RF and Data Memory	Y
5.	Parameterized MUX design	Y
6.	Implemented 30% of the correct result	Y
7.	Implemented 60% of the correct result	Y
8.	Implemented 100% of the correct result	Y
9.	Synthesizable?	Y
10.	Early submission (TBA)	N

QOR

```
****************************************
Report : qor
Design : mips_32
Version: S-2021.06-SP4
Date   : Tue Jun 20 23:00:51 2023
****************************************


  Cell Count
  ----------------------------------
  Hierarchical Cell Count:       5426
  Hierarchical Port Count:      39439
  Leaf Cell Count:              55340
  Buf/Inv Cell Count:            9478
  Buf Cell Count:                5202
  Inv Cell Count:                4276
  CT Buf/Inv Cell Count:            0
  Combinational Cell Count:     45713
  Sequential Cell Count:         9627
  Macro Count:                      0
  ----------------------------------


  Area
  ----------------------------------
  Combinational Area:   116106.195988
  Noncombinational Area: 64347.483747
  Buf/Inv Area:          16473.106336
  Total Buffer Area:         10660.83
  Total Inverter Area:        5812.27
  Macro/Black Box Area:      0.000000
  Net Area:              86054.761408
  ----------------------------------
  Cell Area:            180453.679735
  Design Area:          266508.441143


  Design Rules
  ----------------------------------
  Total Number of Nets:         55435
  Nets With Violations:            23
  Max Trans Violations:             0
  Max Cap Violations:              23
  ----------------------------------


  Hostname: knuee-srv3

  Compile CPU Statistics
  ----------------------------------------
  Resource Sharing:                  50.92
  Logic Optimization:               133.99
  Mapping Optimization:             224.68
  ----------------------------------------
  Overall Compile Time:             439.41
  Overall Compile Wall Clock Time:  442.06
```