Computer Architecture

MS2

Iman Yehia 900171249

Ramy ElGendi 900170269

Sara Ahmed 900170870

28th October, 2019

Instructions

Description of how each instruction was implemented.

- **LUI: RF[rd] = (Imm[32:12]<<12)**

Inside the ALU, the first input inside the ALU is concatenated with 11'b0 to perform the shift. Then, the opcode inside the Control Unit was added with everything equal to 0 except Regwrite,ALUSrc = 1 & ALUOp = 11 (which is shift inside the ALU).

- **AUIPC: RF[rd] = PC+(Imm[32:12]<<12)**

Inside the ALU, we use the same ALUOp=11 to shift 12 bits, then outside there is a multiplexer before the Write Back of the Register File that picks whether its normal WB, or save the immediate of the jump, or the result AUIPC.

- **JAL: TF[rd]=PC+4**

The output of the Instruction Memory is shifted then added to the current PC, then it is linked to the Multiplexer before Write Back and used when jump=1.

- JALR: Here, I added an extra adder that adds the immediate with the contents of rs1. In addition to that, I added an extra MUX that selects between the regular Jump, and the JALR. This way it is sure to select the correct instruction.
- BEQ: In the BEQ, I added subtracted the values of the two inputs of the ALU. If they were 0(equal) it selects the FinalFlag(zeroflag) to be 1. If not, the FinalFlag is 0.
- BNE: In the BNE, I inverted the result of the BEQ, as the two instructions are complete opposites
- BLT: We converted it to signed assuming that the original instruction is unsigned. In the BLT, I checked through the ALU whether the instruction would yield a less than or not. The FinalFlag would generate a 1 if the output was 1, and a 0 if the output was 0.
- BGE: In the BGE, I checked through the ALU, if the instruction would yield a greater than or equal, and turned it to signed, assuming that the original instruction is unsigned. The FinalFlag would generate a 1 if the output was 1, and a 0 if the output was 0.
- BLTU: Assuming that the original instruction is unsigned, I checked through the ALU if the instruction would yield a less than or equal. The FinalFlag would generate a 1 if the output was 1, and a 0 if the output was 0.
- BGEU: Assuming that the original instruction is unsigned, I checked through the ALU if the instruction would yield a less than or equal. The FinalFlag would generate a 1 if the output was 1, and a 0 if the output was 0.
- **Load instruction:**

In order to do the load instructions, i had to change the data memory to be byte addressable. Hence, I had added function3 as an input to datamemory. In addition, I checked whether we are loading or not (reading). If we are loading I checked which load instructions was it. Therefore, if we are loading I had to check for which load instructions we are doing this was done by func3. In addition, if we are loading Control unit sends to the ALU to add. Finally, if the instruction was lw it loaded the 32 bits. Otherwise, if it was lhu or lh we loaded half word i.e 16 bits but the difference is lhu signed extended the rest of the bits with zeros, meanwhile lh extended the bits with the sign bit. Finally, if it was lb or lbu it loaded lower 8 bits, lb extended the bits with the sign bit, and lbu signed extended the rest of the bits with zeros.

- SB: To store a byte , we made the data memory byte addressable. And if we are writing in the memory we checked whether it is SB or not, this was done by function3. If we are storing byte we made the memory to take data_in and writes first 8 bits
- SH: To store a half , we made the data memory byte addressable. And if we are writing in the memory we checked whether it is SH or not, this was done by function3. If we are storing half we store the data in the memory by taking data_in and writes first 16 bits
- SW:To store a word , we made the data memory byte addressable. And if we are writing in the memory we checked whether it is SW or not, this was done by function3. If we are storing a word it stores all the input to the memory.
- ADDI: we faced a problem in addi we as we didnt think about if the immediate value is negative value and hence the instruction would think what I entered should be subtracted hence it subtracted the negative value hence it changed to addition. So, I handled this problem by if it was I format it should add only and it ignored inst[30].
- SLTI: Set less than immediate. ALU checks if register is less than the immediate if yes it set the aluresult with 1 otherwise it set it with zero. This was done for signed value. So we add $signed to identify it is signed value.
- SLTIU:Set less than immediate unsigned. ALU checks if register is less than the immediate if yes it set the aluresult with 1 otherwise it set it with zero. This was done for unsigned values, hence nothing should be added.
- XORI: The ALU xor the register with the immediate and set the output to ALUresult.
- ORI: The ALU or the register with the immediate and set the output to ALUresult.
- ANDI: The ALU and the register with the immediate and set the output to ALUresult.
- SLLI: Shift left logical immediate. ALU shifts the register to the left by taking in the immediate. The result is set to ALUresult.
- SRLI:Shift right logical immediate. ALU shifts the register to the right by taking in the immediate. The result is set to ALUresult.
- SRAI:Shift right arithmetic immediate. ALU shifts the register to the right by taking in the immediate. But we identified that the input is signed value. Hence, the result is set to ALUresult.
- ADD: The ALU took both register and added them together and the output was set to ALUresult.
- SUB:The ALU took both register and subtracted them together and the output was set to ALUresult.
- SLL:Shift left logical immediate. ALU shifts the register to the left by taking in the second register. The result is set to ALUresult.

- SLT:Set less than immediate. ALU checks if first register is less than the second register if yes it set the aluresult with 1 otherwise it set it with zero. This was done for signed value. So i add $signed to identify it is signed value.
- SLTU:Set less than unsigned. ALU checks if the first register is less than the second register if yes it set the aluresult with 1 otherwise it set it with zero. This was done for unsigned values, hence nothing should be added.
- XOR:The ALU xor the first register with the second register and set the output to ALUresult.
- SRL:Shift right logical. ALU shifts the first register to the right by taking in the second register. The result is set to ALUresult.
- SRA:Shift right arithmetic. ALU shifts the first register to the right by taking in the second register. But we identified that the input is signed value. Hence, the result is set to ALUresult.
- OR: The ALU or the first register with the second register and set the output to ALUresult.
- AND:The ALU and the first register with the second register and set the output to ALUresult.
- Ecall: If CU finds ecall it creates all its output with zero. Which results into nop
- Ebreak: It stops the pc from counting any next instruction. This happened by adding a mux that takes pc_out and pc_current and if it is ebreak it chooses pc_current. Hence, it doesn’t update pc.

Schematic Design