**Report**

1. **Architecture Diagrams**

Show your ALU control (if you have one), main control and single-cycle processor design by "Schematic" tool in Vivado, or draw them by yourself.

And briefly explain them.

* **ALU control**
* A diagram of a machine

  Description automatically generated
* ALUOp specifies the type of operation to be performed (e.g., addition, subtraction, bitwise AND, etc.)
* Operation2 = alu\_op0 + funct1 ‧ alu\_op1, which is bit2 in the figure
* Operation1 = alu\_op1’ + funct2’, which is bit1 in the figure
* Operation0 = (func0 + funct3) ‧ alu\_op1, which is bit0 in the figure
* A table with numbers and symbols

  Description automatically generated
* **Main control**
* A diagram of a circuit

  Description automatically generated
* It takes the instruction opcode as input and outputs control signals such as RegDst, MemRead, MemWrite, RegWrite, and Branch.
* RegDst specifies the destination register for the result of an ALU operation.
* MemRead and MemWrite control memory access (read or write).
* RegWrite enables writing the result of an ALU operation back to a register.
* Branch controls branching behavior based on conditional instructions.
* It will be implement on the following figure
* A table with numbers and symbols

  Description automatically generated
* **Single-cycle processor**
* A computer generated diagram of a circuit board

  Description automatically generated
* The instruction memory stores program instructions, and the program counter (PC) fetches instructions from memory.
* The instruction is decoded by the main control unit, which generates control signals for various components based on the opcode.
* The ALU control unit generates control signals for the ALU based on the instruction opcode.
* The ALU performs arithmetic and logic operations, and the result is either written back to the register file or stored in memory based on the instruction type.
* The data path facilitates the flow of data between components, including registers, the ALU, and memory.

1. **Experimental Result**
   1. A screen shot of a computer

      Description automatically generatedShow the waveform screenshot of the test we provided.
   2. What other cases you've tested? Why you choose them?
2. **Answer the following Questions**
   1. When does write to register/memory happen during the clock cycle? How about read?

During a clock cycle, writes to register/memory typically occur at the rising or falling edge of the clock signal. Reads can happen before or after writes, depending on system timing and instruction sequencing.

* 1. Translate the "branch" pseudo instructions (blt, bgt, ble, bge) in the Green Card into real instructions. Only at register can be modified, and other common registers should not be modified.
     + blt $a0, $a1, Label is translated into the following.

slt $at, $a0, $a1 // $at = 1 if $a0 < $a1

bne $at, $0, Label // Branch if $at != 0

* + - bgt $a0, $a1, Label is translated into the following.

slt $at, $a1, $a0 // $at = 1 if $a1 < $a0

bne $at, $0, Label // Branch if $at != 0

* + - ble $a0, $a1, Label is translated into the following.

slt $at, $a1, $a0 // $at = 1 if $a1 < $a0

beq $at, $0, Label // Branch if $at = 0

* + - bge $a0, $a1, Label is translated into the following.

slt $at, $a0, $a1 // $at = 1 if $a0 < $a1

beq $at, $0, Label // Branch if $at = 0

* 1. Give a single beq assembly instruction that causes infinite loop. (consider that there's no delay slot)

Loop: beq $0, $0, loop

* 1. The j instruction can only jump to instructions within the "block" defined by "(PC+4) [31:28]". Design a method to allow j to jump to the next block (block number + 1) using another j.

# Calculate the Address of the Next Block

addi $next\_block, $PC, 4 # Increment PC by 4 to get the address of the next instruction

srl $next\_block, $next\_block, 28 # Extract the upper 4 bits (block number)

addi $next\_block, $next\_block, 1 # Increment the block number to get the next block number

# Use Conditional Jump to Jump to the Next Block

load $current\_block, $PC\_block\_number # Load the block number of the current instruction

bne $current\_block, $next\_block, not\_next\_block # Compare with the block number of the next block

j next\_block\_label # Jump to the next block if the block numbers are equal

not\_next\_block: # Otherwise, continue executing instructions in the current block

* 1. Why a Single-Cycle Implementation Is Not Used Today?

Each instruction must be completed within a single clock cycle, which often leads to inefficient use of hardware resources. For complex instructions that require multiple stages to execute, dedicating resources for each stage in every cycle can be wasteful.