CSCE3301 - Computer Architecture

RISC-V Processor

Names: Jana Elfeky, Kareem Elnaghy

Dr. Cherif Salama

1. Introduction

This project aims to design and implement a processor supporting the RV32I base integer instruction set. The final implementation will be tested on the Nexys A7 trainer kit, allowing it to execute various programs by adhering to RISC-V specifications. The project includes handling hazards in a pipelined design, using single-ported memory, and addressing structural hazards with every-other-cycle instruction issuing.

For this milestone, we focused on building a single-cycle datapath that supports the RV32I base integer instruction set. This processor is designed to execute all 42 user-level RV32I instructions, except ECALL, EBREAK, PAUSE, FENCE, and FENCE.TSO which are instead interpreted as halting the program or no operations.

We created a block diagram detailing the single-cycle datapath, outlining components such as the program counter (PC), control unit, register file, ALU, ALU Control, Immediate Generator, and separate memories for data and instructions. After that, we implemented the design using Verilog, of which the structure of the implementation is explained below in detail.

2. Structure

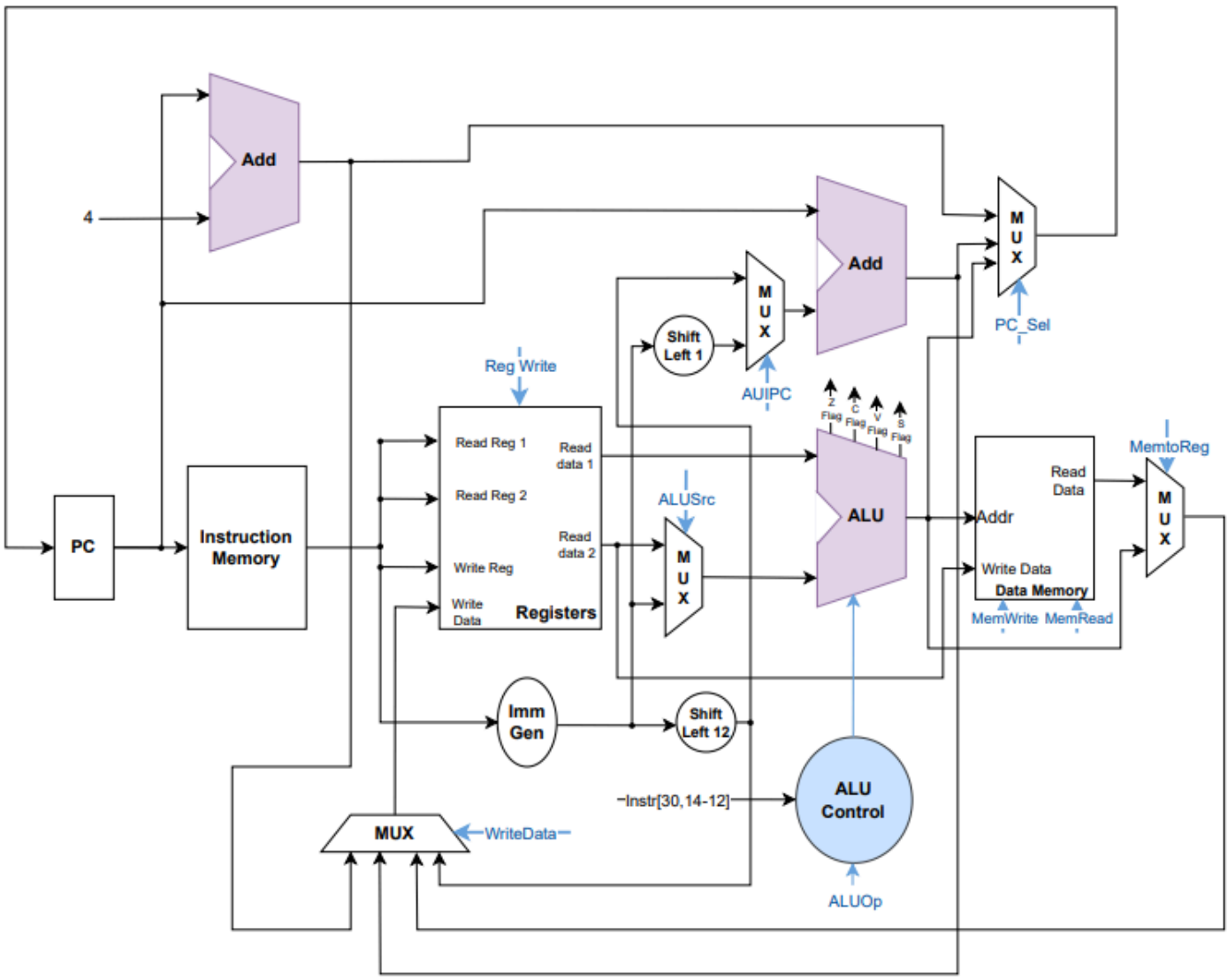
The **riscVSSD** module is a top module which connects a single-cycle RISC-V CPU with a seven-segment display driver, enabling real-time output display. The CPU processes data using `clock`, and its output is sent to LEDs and the display via `ledSel` and `ssdSel`. A 13-bit wire (`ssd`) links the CPU to the display driver, which refreshes segments using `SSD_clock` to show updated values on the seven-segment display. The single cycle cpu module implements are datapath as shown in the diagram below.

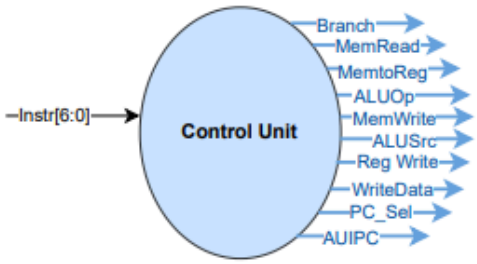
Single Cycle CPU
The `singleCycleCPU` module is a single-cycle RISC-V processor where each submodule serves a distinct role in processing instructions:

1. **register #(32) PC:** A 32-bit Program Counter that holds the current instruction address.
2. **RCA #(32) rca:** An adder that increments the PC by 4 to point to the next instruction.
3. **instr_mem:** Instruction Memory to fetch the current instruction based on the PC.
4. **control_unit CU:** Generates control signals to guide data flow and operation based on the instruction opcode.
5. **registerFile #(32) RF:** A Register File providing read and write access to CPU registers.
6. **ImmGen immgen:** Immediate Generator, producing the immediate value from the instruction.
7. **N_bit_mux #(32) aluMUX:** A multiplexer selecting between register data and the immediate value as input to the ALU.
8. **ALU_control aluCU:** Determines the ALU operation type based on control signals and instruction type.
9. **N_bit_ALU #(32) alu:** Executes arithmetic and logical operations.
10. **data_mem dataMem:** Data Memory for reading/writing data to memory based on the instruction.
11. **N_bit_mux #(32) aluDataMUX:** MUX choosing between ALU result and data memory output for register writes.
12. **N_bit_shift12 shiftBy12:** Shifts the immediate value by 12 bits, typically for jump/branch calculations.
13. **N_bit_mux_4x1 writeDataMUX:** Selects data to be written back to the register file, based on control signals.
14. **branch branchRes:** Calculates branch decisions using flags from ALU outputs.
15. **N_bit_mux_4x1 #(32) pcMUX:** Determines the next PC value, depending on branch decisions or jumps.
16. **RCA #(32) rca1:** Calculates the target address for branches.
17. **N_bit_mux #(32) branchAdderMUX:** Selects between two different immediate values for branch calculation.

The `ledSel` and `ssdSel` inputs control the outputs on `leds` and `ssd`, allowing various CPU states or data to be displayed.

3. Datapath


Add
4
Add
M
U
X
PC_Sel
Shift
Left 1
M
U
X
AUIPC
Reg Write
Z Flag
C Flag
V Flag
S Flag
Read Reg 1
Read Reg 2
Write Reg
Write Data
Registers
Read data 1
Read data 2
ALUSrc
M
U
X
ALU
Addr
Read Data
Write Data
Data Memory
MemWrite
MemRead
MemtoReg
M
U
X
PC
Instruction Memory
Imm Gen
Shift Left 12
Instr[30,14-12]
ALU Control
MUX
WriteData
ALUOp

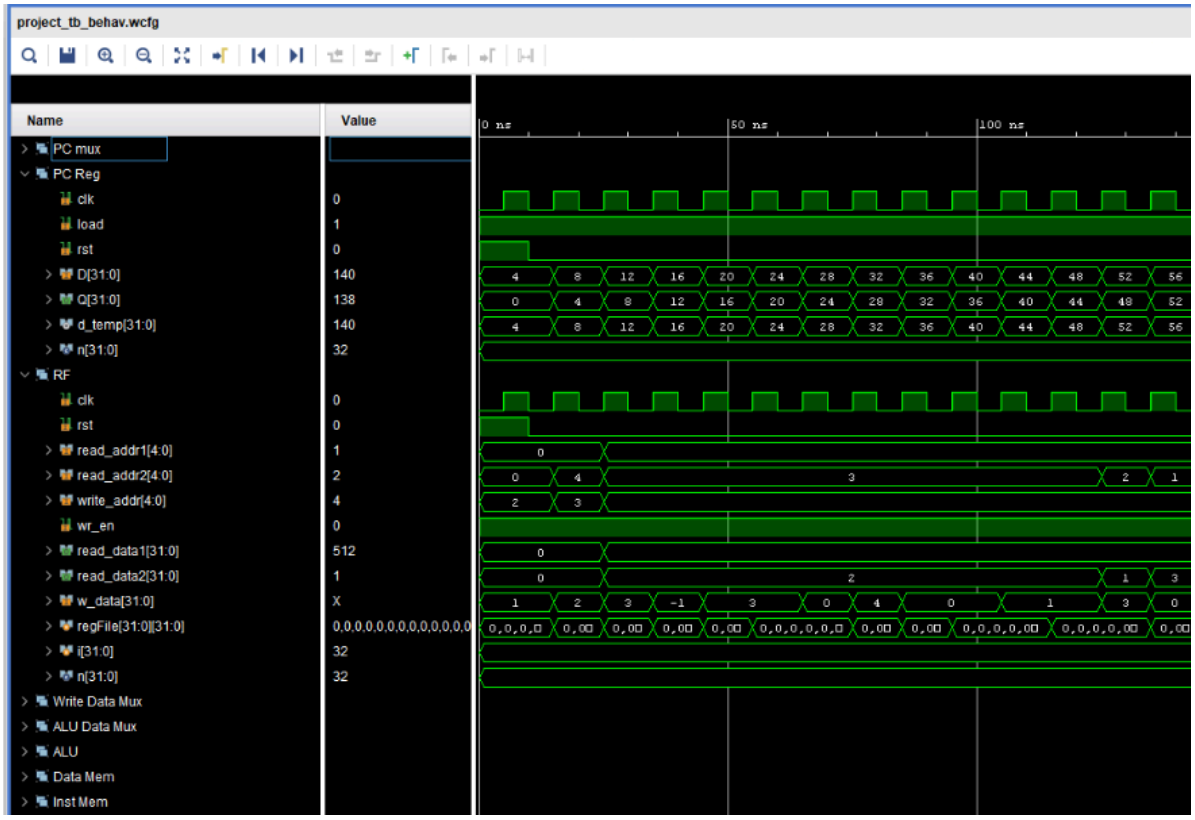

Instr[6:0]
Control Unit
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
Reg Write
WriteData
PC_Sel
AUIPC

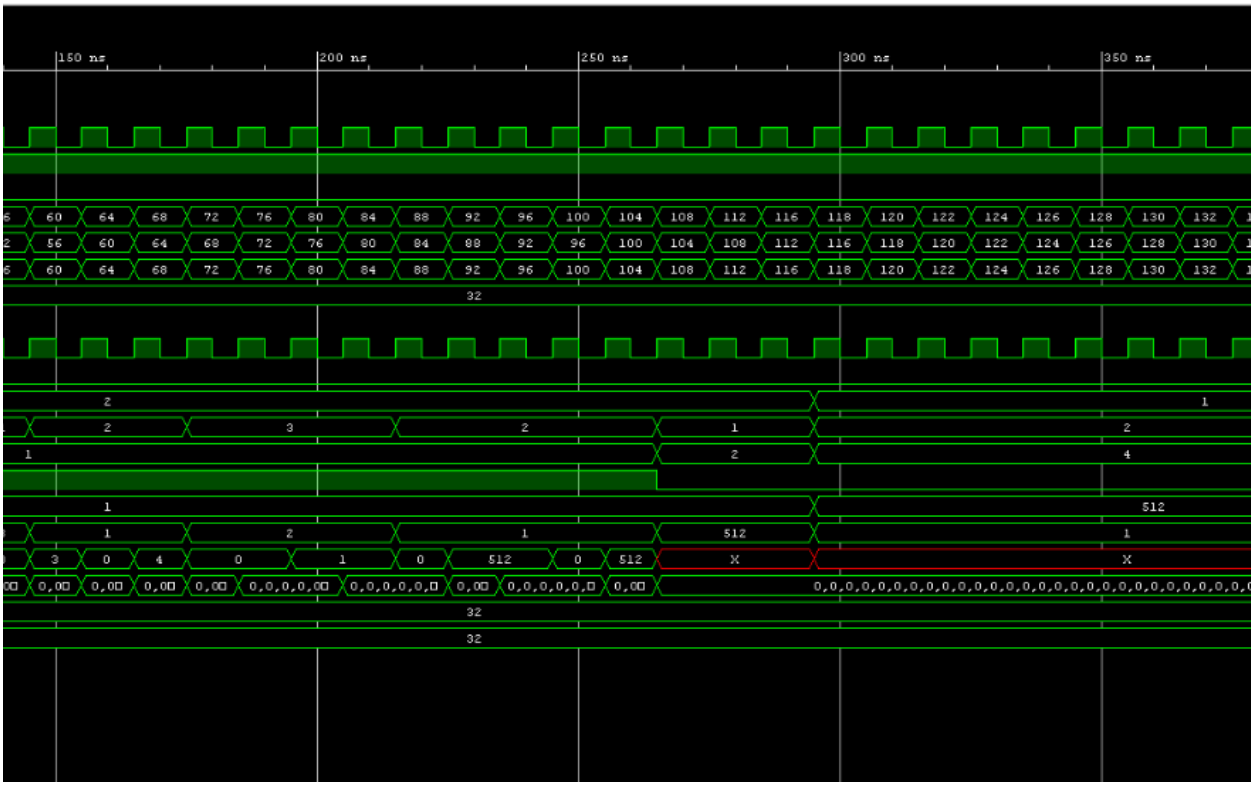
The single-cycle datapath for the RISC-V processor is structured to complete each instruction in a single clock cycle, ensuring that all components work simultaneously within that cycle. Key components include:

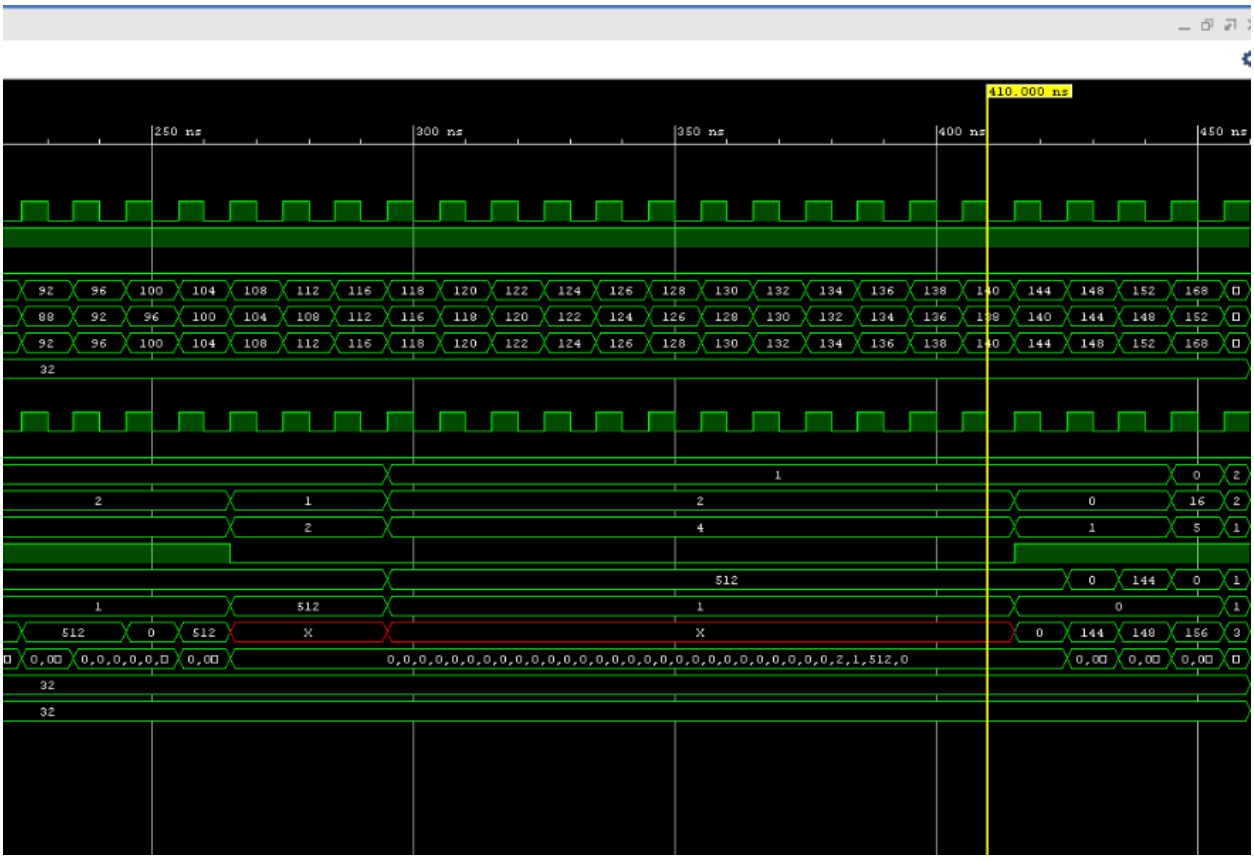
1. **Instruction Memory**: Holds the program instructions, with the program counter (PC) directing which instruction is fetched on each cycle.
2. **Program Counter (PC)**: Determines the current instruction address. After each instruction is executed, the PC increments to fetch the next instruction or halts if ECALL is encountered.
3. **Control Unit**: Decodes the fetched instruction and generates control signals to direct the flow of data, select ALU operations, and coordinate memory and register interactions.
4. **ALU (Arithmetic Logic Unit)**: Performs arithmetic and logical operations based on the control unit's signals. Inputs come from the register file or the immediate generator, depending on the instruction type.
5. **ALU Control**: Receives signal from the control unit and further decodes it to specify the exact operation the ALU should perform.
6. **Immediate Generator**: Extracts and sign-extends immediate values from the instruction for operations needing constants, such as load/store and branch instructions.
7. **Register File**: Stores all general-purpose registers. It reads the required registers for an instruction and writes back results after ALU or memory operations.
8. **Data Memory**: Used for load and store instructions, where data is either read into a register or written from a register.
9. **Multiplexers (MUXes)**: These guide data flow by selecting between inputs (e.g., PC + 4 or branch target for the next PC, or ALU vs. immediate input to the ALU).

This datapath enables all 42 user-level RV32I instructions (except ECALL, EBREAK, PAUSE, FENCE, and FENCE.TSO) in a single cycle by fetching, decoding, executing, and writing back data within one clock pulse.

4. Waveforms


project_tb_behav.wcfg
Name
Value
PC mux
PC Reg
clk
load
rst
D[31:0]
Q[31:0]
d_temp[31:0]
n[31:0]
RF
clk
rst
read_addr1[4:0]
read_addr2[4:0]
write_addr[4:0]
wr_en
read_data1[31:0]
read_data2[31:0]
w_data[31:0]
regFile[31:0][31:0]
i[31:0]
n[31:0]
Write Data Mux
ALU Data Mux
ALU
Data Mem
Inst Mem
0 ns
50 ns
100 ns


150 ns
200 ns
250 ns
300 ns
350 ns



We have included simulation waveforms that verify our processor's functionality by running a comprehensive test program that executes every RV32I instruction. These screenshots demonstrate correct operation across all implemented instructions, showing accurate state changes and outputs for each cycle. This verification confirms that our processor meets the RV32I specifications and performs as expected in a single-cycle implementation.

5. Release Notes

This single-cycle RISC-V processor strictly adheres to the RV32I specification, with no additional assumptions or deviations from the standard. All RV32I instructions have been implemented and thoroughly verified, ensuring correct execution and state updates within a single clock cycle across registers and memory. Certain instructions—specifically `ebreak`, `pause`, `fence`, and `fence.tso` are currently defined as no-operations (NOPs) and have not yet been implemented. The development and testing processes revealed no issues, resulting in a fully operational and reliable implementation.