RISC-V Single Cycle Processor Design

Development, Testing, and Future Enhancements

Anousha Malik | Namal University, Mianwali

Design Overview

Objective: Design and implement a functional RISC-V processor.

Approach: Software-based implementation focusing on design, simulation, and testing.

Technologies: RISC-V Assembly, Verilog HDL, C Compiler, Ripes Simulator.

Outcome: Operational processor supporting multiple instruction types.

RISC-V Architecture Fundamentals

Instruction Set Architecture (ISA)

- **R-type:** Register-based arithmetic operations
- **I-type:** Immediate value operations and loads
- **S-type:** Store operations
- **L-type:** Load operations
- **B-type:** Branch operations
- **J-type:** Jump operations

Cont…

Key Design Principles

Simplicity: Fixed-length instructions (32-bit) ensure easy decoding and execution.

Single-Cycle Execution: Each instruction completes in one clock cycle, reducing complexity.

Load-Store Architecture: Memory access that supports load (L-type) and store (S-type) instructions, improving efficiency.

Scalability: The minimal base ISA (RV32I) supports various extensions.

Assembly Language Programming

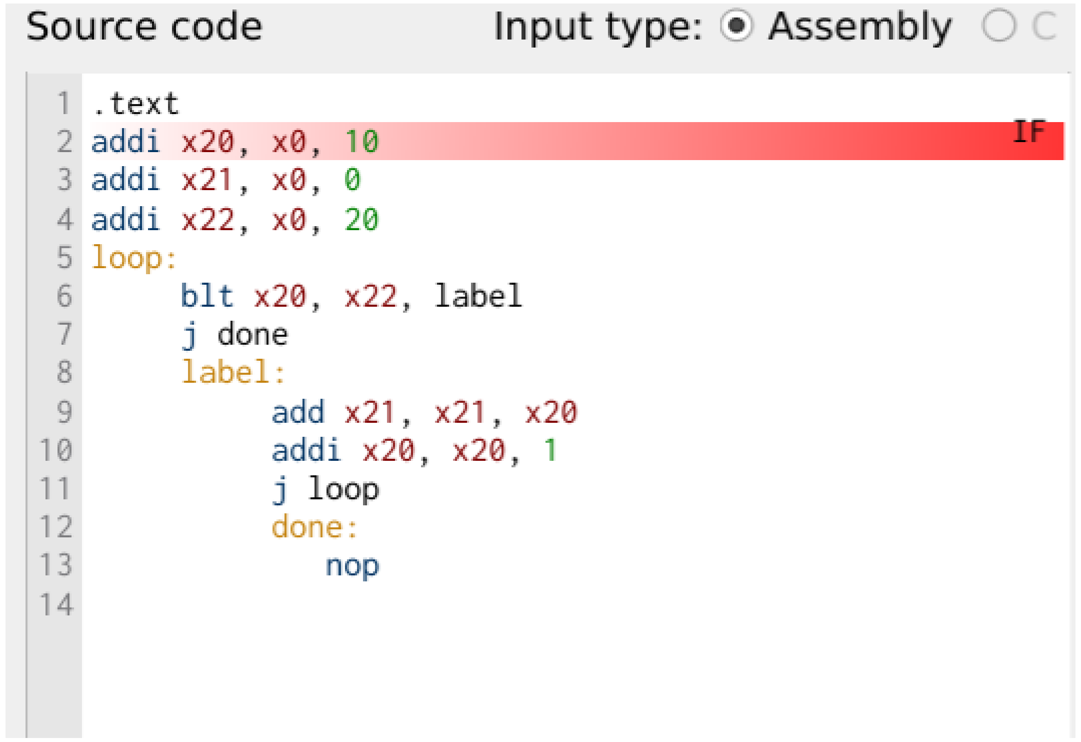
Assembly Development with Ripes Simulator

- Register initialization and manipulation
- Conditional branching implementation
- Looping constructs and arithmetic operations

Cont…

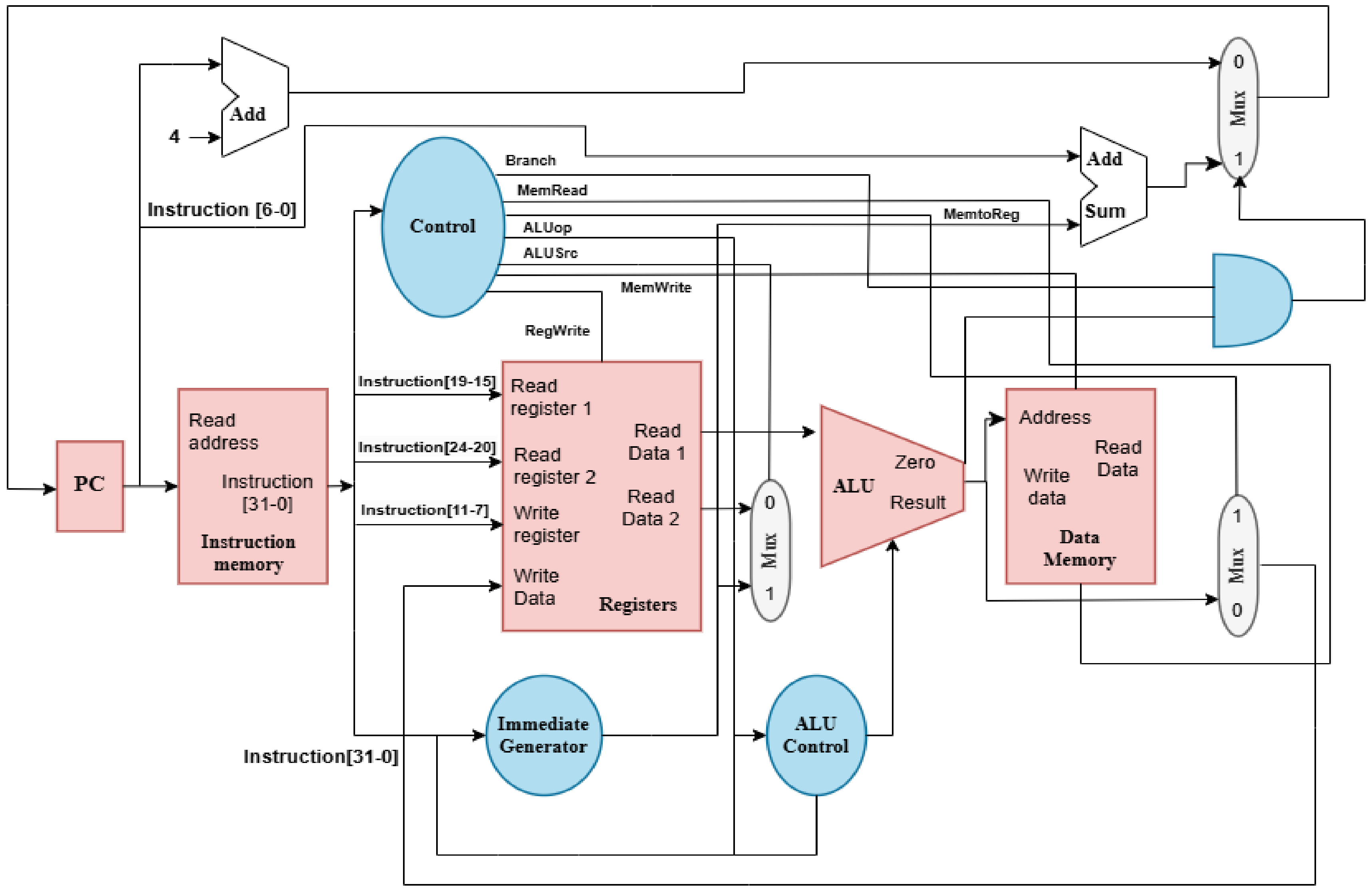
Testing Methodology

- Instruction execution verification
- Register value tracking
- Memory access validation



```
.text
addi x20, x0, 10
addi x21, x0, 0
addi x22, x0, 20
loop:
	blt x20, x22, label
	j done
	label:
		add x21, x21, x20
		addi x20, x20, 1
		j loop
		done:
		   nop
```

Processor Microarchitecture



Core Components of RISC-V Processor

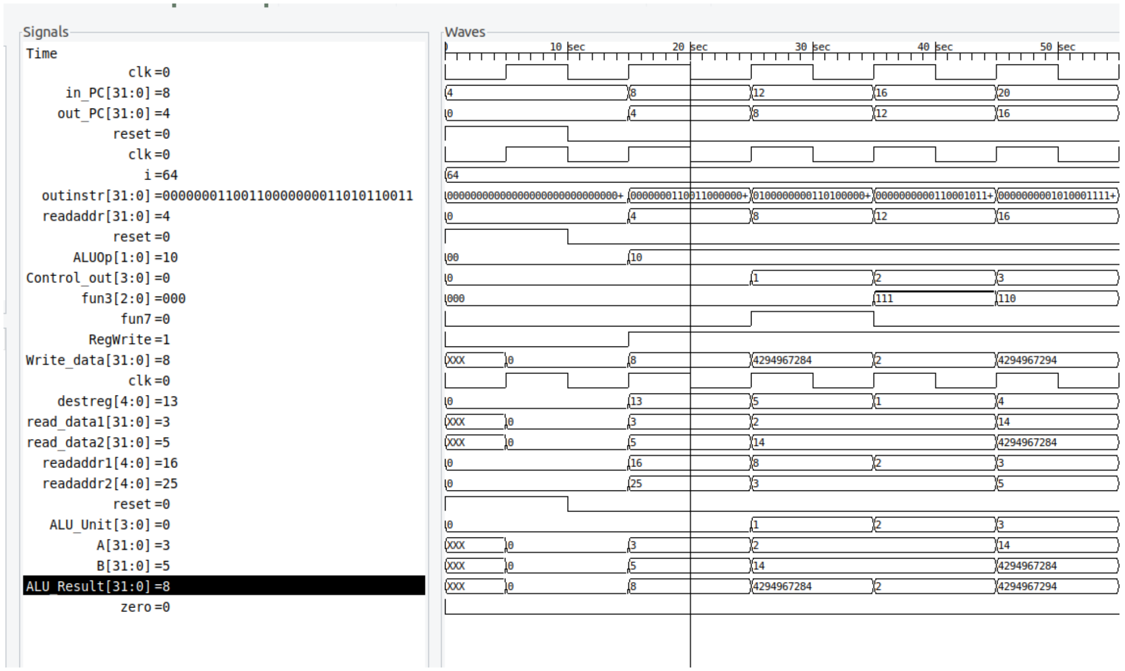
- **Program Counter (PC):** Tracks instruction execution.
- **Instruction Memory:** Stores and fetches machine code.
- **Register File:** Holds 32 general-purpose registers.
- **ALU:** Performs arithmetic and logical operations.
- **Data Memory:** Stores program data for load/store operations.
- **Control Unit:** Decodes instructions, manages execution.
- **Multiplexers (MUX):** Selects data paths for operations.
- **Immediate Generator:** Extracts constants from instructions.
- **Branch Control Logic:** Handles conditional jumps and branches.

Data Path Implementation for R-type Instructions

Example Instructions: ADD, SUB, AND, OR, XOR, SLL, SLT, SRL, SRA

Implementation Details:

- Opcode and function field decoding
- Register operand selection
- ALU operation execution
- Result writeback



```
Time
             clk=0
   in_PC[31:0]=8
  out_PC[31:0]=4
           reset=0
             clk=0
               i=64
outinstr[31:0]=00000001100110000000011010110011
readaddr[31:0]=4
           reset=0
    ALUOp[1:0]=10
Control_out[3:0]=0
     fun3[2:0]=000
            fun7=0
        RegWrite=1
Write_data[31:0]=8
             clk=0
  destreg[4:0]=13
read_data1[31:0]=3
read_data2[31:0]=5
readaddr1[4:0]=16
readaddr2[4:0]=25
           reset=0
 ALU_Unit[3:0]=0
        A[31:0]=3
        B[31:0]=5
ALU_Result[31:0]=8
            zero=0
```

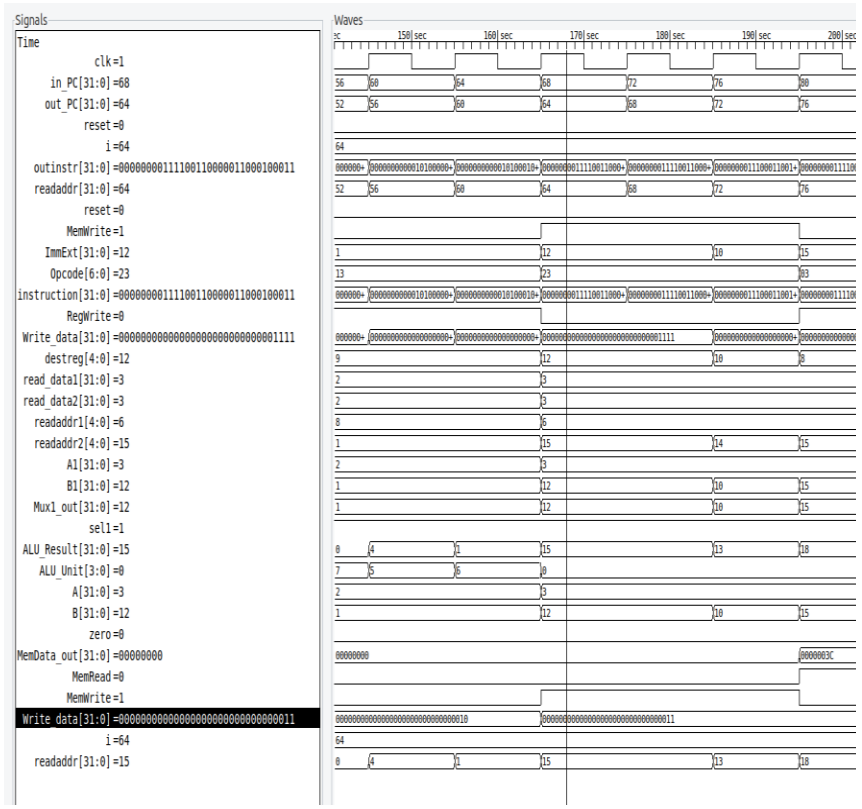
Memory Access Operations (Load/Store)

Store Instructions (S-type): SB, SH, SW

- Base address + offset calculation
- Data selection and alignment
- Memory write control

Load Instructions (L-type): LB, LH, LW

- Address generation
- Memory read operation
- Sign extension and register writeback



```
Signals
Time
               clk=1
      in_PC[31:0]=68
     out_PC[31:0]=64
             reset=0
                 i=64
   outinstr[31:0]=00000000111100110000011000100011
   readaddr[31:0]=64
             reset=0
          MemWrite=1
     ImmExt[31:0]=12
      Opcode[6:0]=23
instruction[31:0]=00000000111100110000011000100011
          RegWrite=0
 Write_data[31:0]=00000000000000000000000000001111
     destreg[4:0]=12
 read_data1[31:0]=3
 read_data2[31:0]=3
   readaddr1[4:0]=6
   readaddr2[4:0]=15
         A1[31:0]=3
         B1[31:0]=12
   Mux1_out[31:0]=12
              sel1=1
 ALU_Result[31:0]=15
    ALU_Unit[3:0]=0
          A[31:0]=3
          B[31:0]=12
              zero=0
MemData_out[31:0]=00000000
           MemRead=0
          MemWrite=1
 Write_data[31:0]=00000000000000000000000000000011
                 i=64
   readaddr[31:0]=15
```

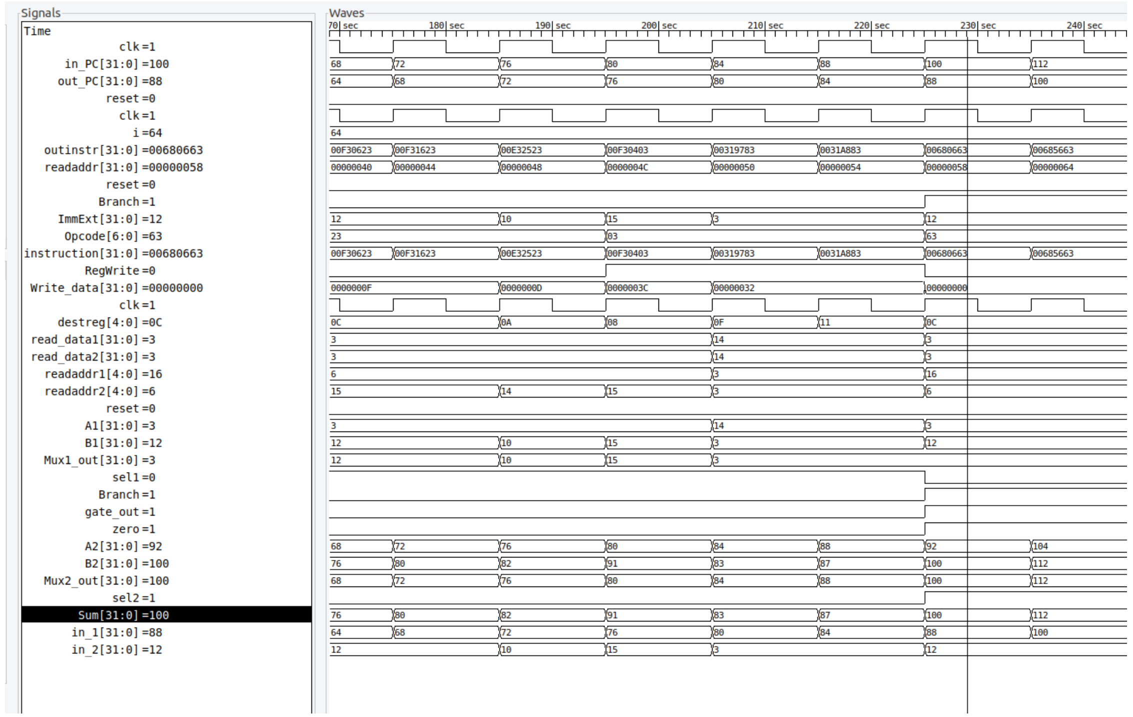
Control Flow Implementation

Branch Instructions

- Condition evaluation (equality, inequality, comparison)
- Target address calculation
- PC update mechanism

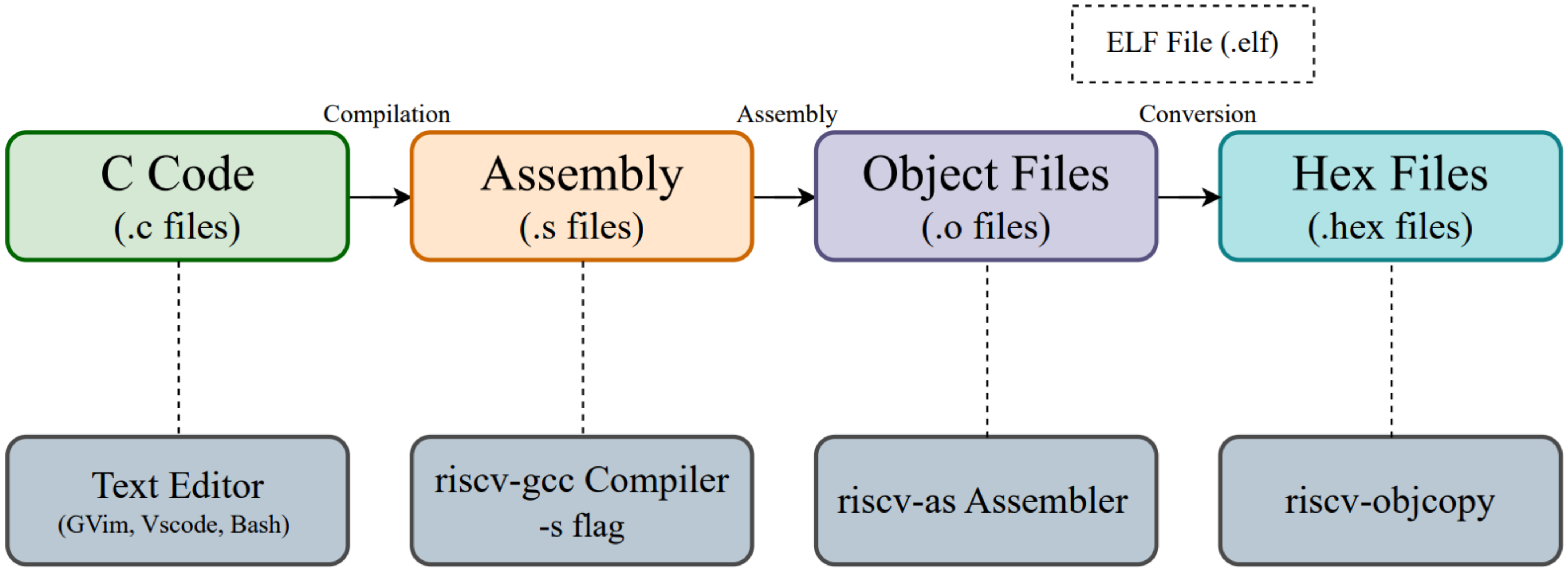
Control Unit Design

- Instruction decoding logic
- Control signal generation
- Operation sequencing



```
                clk =1
      in_PC[31:0] =100
     out_PC[31:0] =88
              reset =0
                clk =1
                  i =64
   outinstr[31:0] =00680663
   readaddr[31:0] =00000058
              reset =0
             Branch =1
     ImmExt[31:0] =12
      Opcode[6:0] =63
instruction[31:0] =00680663
           RegWrite =0
 Write_data[31:0] =00000000
                clk =1
     destreg[4:0] =0C
 read_data1[31:0] =3
 read_data2[31:0] =3
   readaddr1[4:0] =16
   readaddr2[4:0] =6
              reset =0
         A1[31:0] =3
         B1[31:0] =12
   Mux1_out[31:0] =3
               sel1 =0
             Branch =1
           gate_out =1
               zero =1
         A2[31:0] =92
         B2[31:0] =100
   Mux2_out[31:0] =100
               sel2 =1
        Sum[31:0] =100
       in_1[31:0] =88
       in_2[31:0] =12
```

C Compiler Integration



Cont…

GCC Toolchain Workflow

- C code development
- RISC-V assembly generation
- Object file creation
- ELF to hex conversion

```
$ riscv32-unknown-elf-gcc -S -march=rv32i \
  -mabi=ilp32 program.c -o program.s

$ riscv64-unknown-elf-as -march=rv32i \
  -o program.o program.s

$ riscv64-unknown-elf-objcopy -O verilog \
  program.o program.hex
```

Testing and Validation Results

1. Testing

- **Instruction Execution Testing:** Ensures correct execution of R, I, S, L, and B-type instructions.
- **Register File Validation:** Confirms that registers correctly store and update values.
- **Memory Operations Testing:** Verifies correct loading and storing of data.

Cont…

2. Simulation and Debugging

- **Software Simulation:** Uses tools like Ripes, Verilator, or ModelSim for step-by-step execution.
- **Waveform Analysis:** Observes signal transitions using GTKWave.
- **Instruction Tracing:** Tracks executed instructions and data movement.

Future Work

- Pipeline implementation to improve performance.
- Performance optimization and timing analysis.
- Porting the processor to an FPGA for hardware implementation.

References

- D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface*, 2nd ed. Cambridge, MA, USA: Morgan Kaufmann, 2020.
- D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*, 2nd ed. Amsterdam, Netherlands: Morgan Kaufmann, 2012.