**Fall 2024/2025 Computer Architecture (2) Dr. Rajaa Alqudah**

# Final Project

**Project Overview:**

This project tasks groups of two students with designing and implementing a simplified RISC-V processor using Verilog HDL. The processor will support a core set of RISC-V instructions and will be enhanced to incorporate advanced features and functionalities. Students are expected to test all supported features thoroughly and document their design process.

**Requirements:**

You are required to develop the following:

1. Core Processor Design
   a. Implement a limited version of a RISC-V processor supporting the instructions in the table below.
   b. Use separate instruction and data memories: Instruction Memory (IM) at 64K x 1 byte, Data Memory (DM) at 8K x 1 byte.
   c. The processor execution should be triggered on a positive edge.
   d. To allow for Reading/Writing to memory and Reg file during the same cycle, use the positive edge of the clock to trigger reading)

2. Pipeline Management
   a. Implement EXE and MEM forwarding to manage dependencies between instructions.
   b. Data Hazard Detection: Design a mechanism to detect data hazards and implement stalls or optimized forwarding paths as needed.
   c. Handling for Control Hazards is required. (Example: need to flush pipeline if there is a jump!)

3. Performance Features
   a. Dynamic Branch Prediction: Design a simple 2-bit predictor to enhance branching accuracy.
   b. Cache Simulation: Implement a basic direct-mapped cache for instruction and data. Instruction cache is 4KB while data cache is 1KB.
   c. Loop Unrolling Optimization: Implement support for loop unrolling in the processor pipeline to reduce branching and improve performance on repetitive tasks.

**Tools:**

Use one of the following Verilog simulators (or any one of your choice)

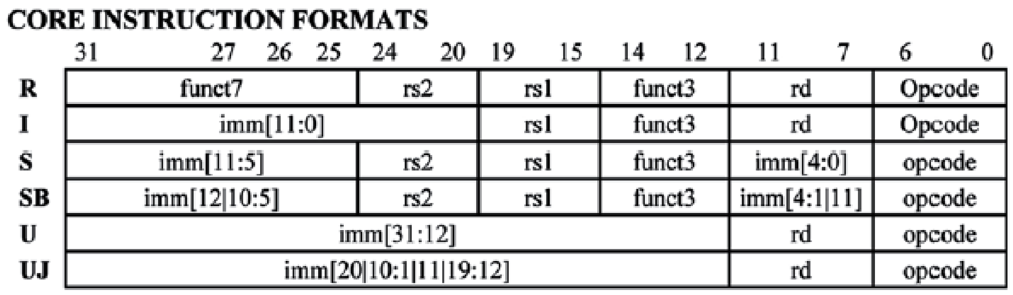
- Download Verilog from: http://bleyer.org/icarus/
- www.edaplayground.com
- Intel® Quartus®

## Core instructions needed

| No. | NAME | FORMAT | MNEMONIC | Description (in Verilog) | OPCODE/ FUNCT3/FUNCT7 or IMM in HEX |
|---|---|---|---|---|---|
| 1 | Add word | R | addw | R[rd] = R[rs1] + R[rs2] | 33/1/20 |
| 2 | Add Immediate word | I | addiw | R[rd] = R[rs1] + imm | 13/0 |
| 3 | And | R | and | R[rd] = R[rs1] & R[rs2] | 33/7/00 |
| 4 | And Immediate | I | andi | R[rd] = R[rs1] & imm | 1B/6 |
| 5 | Branch On Equal | SB | beq | if(R[rs1]==R[rs2]) PC=PC+{imm, 1b'0} | 63/0 |
| 6 | Branch On Not Equal | SB | bne | if(R[rs1]!=R[rs2]) PC=PC+{imm, 1b'0} | 63/1 |
| 7 | Jump And Link | UJ | jal | R[rd]=PC+4;PC= PC +{imm, 1b'0} | 6F |
| 8 | Jump And Link Register | I | jalr | R[rd]=PC+4;PC= R[rs1] +imm | 67/0 |
| 9 | Load Half Word (1) | I | lh | R[rd]={48'bM[](15), M[R[rs1]+imm](15:0)} | 03/2 |
| 10 | Load Upper Imm. | U | lui | R[rd] = {imm, 12'b0} | 38 |
| 11 | Load Word | I | lw | R[rd] = {M[R[rs1]+imm](31:0)} | 03/0 |
| 12 | xor | R | xor | R[rd] = R[rs1] ^ R[rs2] | 33/3/00 |
| 13 | Or | R | or | R[rd] = R[rs1] \| R[rs2] | 33/5/00 |
| 14 | Or Immediate | I | ori | R[rd] = R[rs1] \| imm | 13/7 |
| 15 | Set Less Than | R | slt | R[rd] = (R[rs1] < R[rs2]) ? 1 : 0 | 33/0/00 |
| 16 | Shift Left | R | sll | R[rd] = R[rs1] << R[rs2] | 33/4/00 |
| 17 | Shift Right | R | srl | R[rd] = R[rs1] >> R[rs2] | 33/2/00 |
| 18 | Store Byte | S | sb | M[R[rs1]+imm](7:0) = R[rs2](7:0) | 23/0 |
| 19 | Store Word | S | sw | M[R[rs1]+imm](31:0) = R[rs2](31:0) | 23/2 |
| 20 | Subtract | R | sub | R[rd] = R[rs1] - R[rs2] | 33/6/00 |

Notes:

(1) Singed Load instruction: extend the sign bit of data to fill the 32-bit register.



**CORE INSTRUCTION FORMATS**

| | 31 27 26 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|---|
| **R** | funct7 | rs2 | rs1 | funct3 | rd | Opcode |
| **I** | imm[11:0] | | rs1 | funct3 | rd | Opcode |
| **S** | imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |
| **SB** | imm[12\|10:5] | rs2 | rs1 | funct3 | imm[4:1\|11] | opcode |
| **U** | imm[31:12] | | | | rd | opcode |
| **UJ** | imm[20\|10:1\|11\|19:12] | | | | rd | opcode |

## **Project Phases:**

The project will be divided into 2 phases. You need to submit each phase on its deadline through e-learning. Below are the details of the requirements and deadline of each phase:

### **Phase 1:**

- In phase 1 you need to implement the following discrete modules in Verilog:
    - PC
    - Instruction Memory
    - Control Unit
    - Immediate Gen
    - Register File
    - ALU
    - Data Memory
- You must test all your modules using the provided testbench template.
- Mention any additional modules and include testbench.
- Implement the pipeline stages of the CPU. The stages are:
    - IF stage
    - ID stage
    - EXE stage
    - MEM stage
    - WB stage
- You must test each stage independently. Please include testbench and describe datapath in each stage.
- Phase 1 Submission should include the following:
    - Design & Test Verilog files: Your design should be modular, test each module separately
    - Run logs: the run results for each test case (text and waveforms).
    - Test CPU using sample benchmarks. (Arithmetic Operations, Load/Store).
- **Phase 1 is due by December 15th, 2024**

**Phase 2:**

- In this stage you need to enhance your CPU by doing the following:
    - Add forwarding (bypassing) to resolve data Hazard
    - Add hazard detection unit
    - Branch Prediction
    - Cache
    - Include branch determination.
- You need to test your design using the following sample benchmarks.
    - Loops
    - If statement
    - ALU-ALU Forwarding
    - Double Data Hazard
    - Load Use Hazard
    - Branches
- Phase 2 Submission should include the following:
    1. Project Report that included the following:
        - Introduction: describe the overall components and design methodology.
        - Processor Design: Detailed description of each module and its functionality.
        - **Design Block Diagram:** Preferably, standalone image or PDF file. Showing names and wire name of all components – needs to match the Verilog names.
        - Design Analysis: Discuss issues and decisions you made during the design.
        - Given Tests runs and discussion:  Show the output of the given test cases and discuss their output.
        - Additional Test runs: Discuss additional crucial test cases (Maximum 10). These are to be discussed in the report. But you can create other tests and submit them with the project along with their results.
        - Performance analysis and comparison with non-pipelined architecture.
        - Conclusion
    2. Design & Test Verilog files: Your design should be modular, test each module separately in addition to the complete design
    3. Run logs: the run results for each test case (text and waveforms).
- **Phase 2 is due by January 8th, 2024**

**General Instructions:**

- Groups of 2 students have to be formed and a list of group members has to be emailed to the course instructor by Sunday 17/11/2024.
- All submissions are through e-learning and need one submission per team only.
- Delivery past the deadline will result in **ZERO** credit.

## **<u>Grading Policy</u>**

The distribution of the grade will be as follows:

### **<u>Phase 1: 40%</u>**

It will be distributed on the following:

- The design code of the required modules
- Test benches
- Run Logs

### **<u>Phase 2: 60%</u>**

It will be distributed on the following:

- Final Report as discussed above.
- Design Functionally needed for this phase
    - Core Instructions
    - Forwarding
    - Control Hazards
    - Exception Handling
- Test benches and run logs