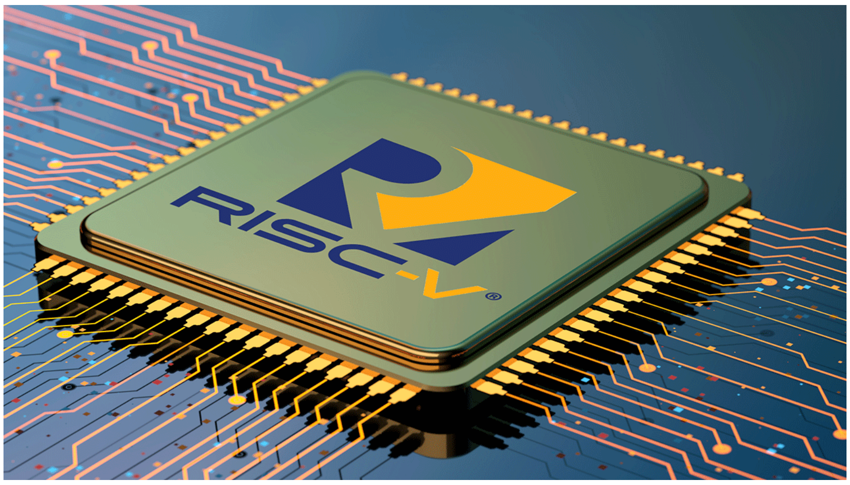
METU EE446
Computer Architecture
Laboratory



Laboratory Project - Single Cycle RISC-V Processor

Objectives

This project aims to explore a novel, license-free, open-source instruction set architecture that is becoming increasingly popular in the industry. You will construct the datapath and the control unit of a single-cycle 32-bit RISC-V processor. For this project, the instruction set has been extended by one more instruction. The designed processor will be able to execute all instructions in the **extended** instruction set. Finally, you will embed your design into the FPGA of the DE1-SoC board and demonstrate your design.

This project will be done in groups of 2 students unless you choose to do the project by yourself. You can choose your partner. Each partner is expected to contribute in equal amounts. If the work is divided too unevenly, the student who did more will be generously graded, while the student who did less will be penalized. The most uneven work division acceptable is 60-40. If you wish to do the project in a group but cannot find a partner, we will match you with another student (if possible).

The project needs to be done by groups individually. **In other words, inter-group cooperation will be considered as cheating and further action will be taken as explained in the EE446 Laboratory Manual.**

Contents

1 Introduction

RISC-V is an innovative instruction-set architecture (ISA) that was initially developed to support research and education in computer architecture at UC Berkeley. Its design aspirations have since expanded, aiming to become a standard, free, and open architecture for industry implementations. Characterized by its flexibility and generality, RISC-V is not tailored to specific microarchitectural styles or technologies, making it suitable for a wide range of hardware implementations, from custom chips to multicore processors. The architecture includes a modular structure with a base integer ISA and optional extensions supporting 32-bit and 64-bit address spaces. RISC-V is designed to facilitate efficient implementations and is fully virtualizable, simplifying hypervisor development. This open and versatile ISA holds significant potential to influence academic research and industrial applications broadly.

1.1 Reading Assignment

In this project, a part of RISC-V ISA will be implemented. To get familiar with it, read the RISC-V specification given in the link below. Chapters 2, 24 and 25 are of most interest. Note that this project manual doesn't contain the low-level details needed to implement the processor fully. Therefore, you'll need to seek additional information from relevant parts of the RISC-V specification.

RISC-V Specification 20191213.pdf

Additionally, Chapter 7.3 of Harris & Harris Risc-v book contains an implementation of part of the ISA, which you can consider:

Sarah Harris, David Harris - Digital Design and Computer Architecture RISC-V Edition (2021)

Since RISC-V is an open standard, many materials are freely available online.

As usual, taking any code without citing its origin or taking large sections of code from any source is Plagiarism and will result in a 0x0 (zero) grade, whereas disciplinary action may be taken.

1.2 Comparison with ARM

ARM 32-bit had 16 architectural registers, with PC=R15. On the other hand, RISC-V has 32 registers (x0 to x31), and PC is not one of these registers. Also, register0 is hardwired to zero value.

ARM had conditional instructions, but RISC-V only has conditional branches. Therefore, ALU flags don't need to be saved to a register. RISC-V does not have shifted operands, but it has shift instructions.

RISC-V has six types of instructions as seen in Figure 1. Instructions can be read from the rs1/rs2 registers and written to the rd register. Instruction encodings may look complicated, especially the immediate encodings. However, these encodings reduce the number of multiplexers and are efficient to implement in hardware.

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

Figure 1: RISC-V instruction types

2 Project Preliminary Work (20%)

2.1 ISA to be implemented

RISC-V consists of base ISA and extensions. In this project, the Unprivileged RV32I Base Integer Instruction Set will be implemented. Additionally, one new instruction will be added as an extension.

FENCE, ECALL, EBREAK, and HINT instructions will not be implemented, as they are irrelevant.

The list of instructions to be implemented is given in Table 1. For more information about the instructions, refer to the RISC-V specification.

Arithmetic instructions:	ADD[I], SUB
Logic instructions:	AND[I], OR[I], XOR[I]
Shift instructions:	SLL[I], SRL[I], SRA[I]
Set if less than:	SLT[I][U]
Conditional branch:	BEQ, BNE, BLT[U], BGE[U]
Unconditional jump:	JAL, JALR (Return-address stack push/pop functionality will not be implemented)
Load:	LW, LH[U], LB[U]
Store:	SW, SH, SB
Others:	LUI, AUIPC
Extra instruction:	XORID

Table 1: List of Instructions

2.1.1 Extra instruction

The extra instruction is XORID. It will take the xor of rs1 with an embedded constant and write the result to rd. It has the following format:

XORID rd, rs1 $\quad rd \leftarrow rs1 \oplus (studentId1 \oplus studentId2)$

Instruction encoding details are given below:

- opcode[6:0]=0001011
- I type instruction, but immediate value will not be used
- funct3[2:0]=100

2.2 Datapath (25%)

2.2.1 Datapath Design

Design a datapath that will support all of the instructions listed above. In your report, explain your design with appropriate visuals. It can be based on the datapath shown in Figure 2.

2.2.2 Datapath Implementation

Implement your design in Verilog HDL. Datapath must consist of submodules but should not contain ”always blocks” or direct logic. You can use the modules from laboratories or create your own.
Datapath needs to have a synchronous reset.
Show the synthesized RTL view of your datapath in your report. Explain how each instruction type is executed in your datapath.

2.3 Controller (25%)

2.3.1 Controller Design

Design a controller that matches your datapath. In your report, explain your controller with appropriate visuals. Explain any submodules the controller has.

2.3.2 Controller Implementation

Implement your design in Verilog HDL. The controller can be written in a single module, or it can have submodules.
Show the synthesized RTL view of your controller in your report. Explain how each instruction type is executed in your controller.

2.4 Top Level (5%)

In this part, the controller and datapath are assembled in a top module. The top module needs to have a synchronous reset.

For debugging purposes, connect five switches to the register files debug port select. Connect debug register output and PC register output to 7-segment displays as in previous labs.

2.5 Testbench (45%)

It is required to verify your Computer's operation by writing a testbench.

- The testbench should read the instructions from a hex file like the HDL computer.
- It should execute all the instructions by itself and then compare its register file and PC values to the HDL design.
- The testbench should be able to execute arbitrary RISC-V code consisting of the given RV32I instructions.
- A proper testbench is automated, so it should indicate when something fails in the design without needing any manual work.

Your testbench should be very similar in form to the supplied testbenches on ODTUClass. You can use the single-cycle ARM computer testbench that was provided and modify it appropriately. For debugging purposes, you can use prints/logs in your testbench.

Explain how you implemented your testbench and its details in your report. You'll need to write RISC-V program(s) that cover all instructions with their special cases. Explain your programs and how they cover all instructions and special cases.

2.6 Important Considerations

Your codes should be clean, easily readable, and understandable. Check your indentations so that they are obvious where a block ends. Put comments when and where needed.

Since you have almost graduated, we expect a more professional report for the project compared to some of your laboratory reports. Your report should be proofread and structured well, and the texts in it should not be pictures of handwriting.

If you used some online/offline tools to compile or assemble RISC-V codes, specify them in your report.

You can lose grades if your code/report is unprofessional, hard to read, etc.

Deliverables:

- PDF Report
- Python testbench codes
- HDL codes

3 Project Demonstration (80%)

In the demonstration, you will be given a code segment for your computer; you will first use the code in your testbench and demonstrate it to your teachers. Then, load your processor designed in Part 2 to the DE1-SoC board with the given instructions.

Additionally, the teachers will ask questions about the parts each student has contributed. You should be very familiar with your work to be able to answer any questions. If a student cannot answer the questions to the satisfaction of the teachers, you will lose grades.

You should also bring your own test codes incase the computer cannot execute the given instructions

A Useful Materials

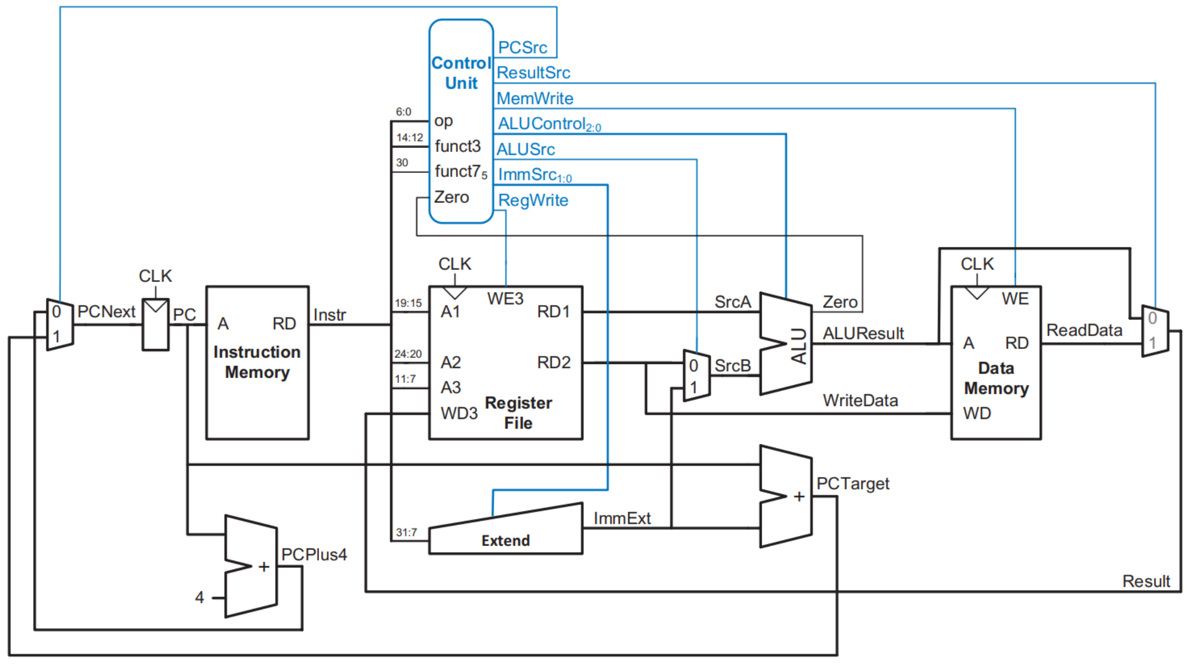


Figure 2: RISC-V Single Cycle Computer as Described in Harris&Harris