LAB # 10
Designing Microarchitecture-IV

Objective:

- To design RISC-V microarchitecture for R-type instruction.

System Module (Hardware/Software)

- Visual Studio Code

Theoretical Background:

Introduction

The name R-type is short for register-type. R-type instructions use three registers as operands: two as sources, and one as a destination. Figure below shows the R-type machine instruction format. The 32-bit instruction has six fields: op, rs1, rs2, rd, funct3, and funct7. Each field is five or seven bits, as indicated. The operation the instruction performs is encoded in the three fields highlighted in blue: op (also called opcode or operation code) and funct3 and funct7 (also called the function). All R-type instructions have an opcode of 33. The specific R-type operation is determined by the funct field. The operands are encoded in the three fields: rs1, rs2, and rd. The first two registers, rs1 and rs2, are the source registers; rd is the destination register.

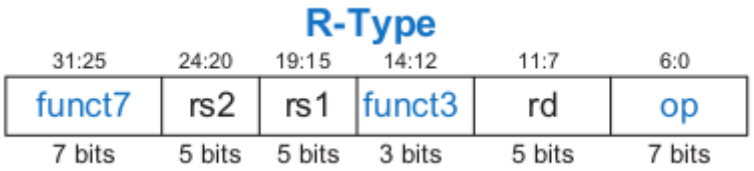


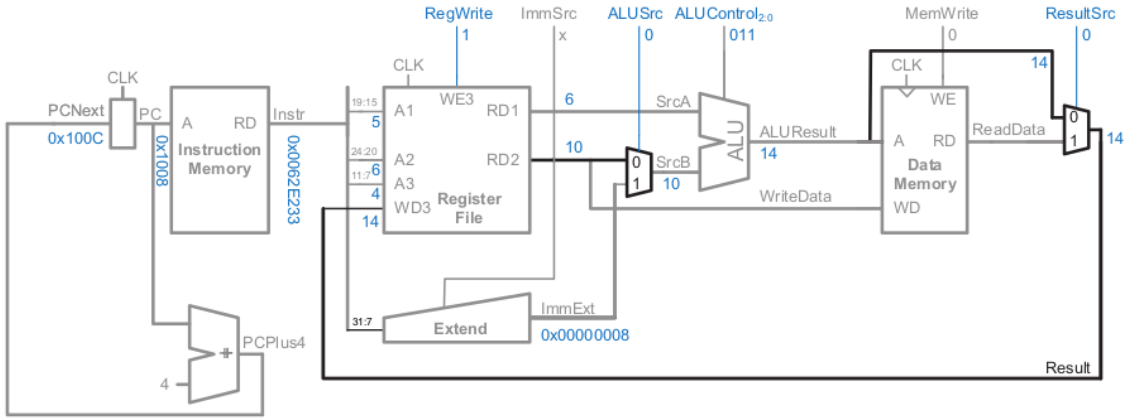
Figure below shows an example for R-Type instruction represented in assembly code and machine code.

Address	Instruction	Type	Fields						Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Data Path for R-Type instruction

In this section we will extend the data path to handle R-type instructions add, sub, and, or, and slt. All of these instructions read two registers from the register file, perform some ALU operation on them, and write the result back to a third register in the register file. They differ only in the specific ALU operation. Hence, they can all be handled with the same hardware, using different ALUControl signals.

Figure below shows the enhanced datapath handling R-type instructions. The register file reads two registers. The ALU performs an operation on these two registers. In previous labs we saw that the ALU always received its SrcB operand from the sign-extended immediate (ImmExt). Now, we add a multiplexer to choose SrcB from either the register file RD2 port or ImmExt. The multiplexer is controlled by a new signal, ALUSrc. ALUSrc is 0 for R-type instructions to choose SrcB from the register file; it is 1 for Load and store to choose ImmExt.



Address	Instruction	Type	Fields					Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

This principle of enhancing the datapath's capabilities by adding a multiplexer to choose inputs from several possibilities is extremely useful. Indeed, we will apply it twice more to complete the handling of R-type instructions. In lab 9 , the register file always got its write data from the data memory. However, R-type instructions write the ALUResult to the register file. Therefore, we add another multiplexer to choose between ReadData and ALUResult. We call its output Result. This multiplexer is controlled by another new signal, ResultSrc. ResultSrc is 0 for R-type instructions to choose Result from the ALUResult; it is 1 for Load to choose ReadData. We don't care about the value of ResultSrc for store, because store does not write to the register file.

Procedure:

Open Visual Studio Code and perform the procedure mentioned in lab #01 in order to make a Verilog module and test bench.

Lab Tasks:

1. **Write a complete data path for R-type instruction in Verilog by instantiating all the module blocks. Attach the code.**

Conclusion:

What have you learnt from this lab?

Learning Outcomes:

Upon successful completion of the lab, students will be able to:

LO1: Design data path for R-type instruction in Verilog.