

**COMPLETE RISC – V SINGLE -CYCLE PROCESSOR**

**RAO MUHAMMAD UMER**

**TC -21073**



# 

# Control Unit

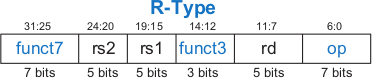
## Introduction

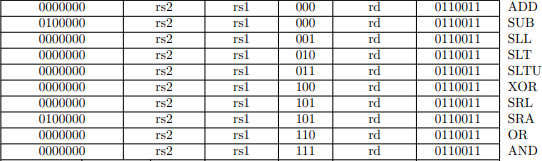
RISC-V uses 32-bit instructions. RISC-V consists of defining the following instruction formats: R-type, I-type, S-Type, B-Type, U-type, and J-type. R-type instructions operate on three registers. I-type, S-type and B-type instructions operate on two registers and a 12-bit immediate. U-type and J-type (jump) instructions operate on one 20-bit immediate.

### R-Type Instructions

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. The 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**.

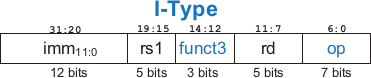
The specific R-type operation is determined by the function fields. The operands are encoded in the three fields: **rs1**, **rs2**, and **rd**. The first two registers, rs1, and rs2 are the source registers; rd is the destination register.

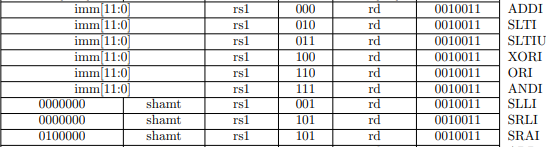
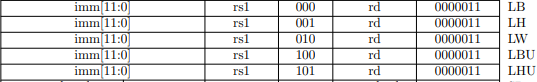


Instruction Format for some of the R-type instructions is shown below:

### I-Type Instructions

The name I-type is short for immediate-type. I-type instructions use two register operands and one immediate operand. The figure below shows the I-type machine instruction format. The 32-bit instruction has five fields: op, rs1, rd, funct3, and imm. The first three fields, op, rs1, and rd, are like those of R-type instructions. The imm field holds the 12-bit immediate. The funct3 holds the operation to be performed. The operation is determined by the opcode and funct3, highlighted in blue. The operands are specified in the two fields rs1, and imm. rs1 and imm are always used as source operands. rd is used as a destination.



Instruction format for some of the I-type instructions are shown below:

### S-Type Instructions

The name S-type is short for store-type. S-type instructions use two register operands and one immediate operand. Figure below shows the S-type machine instruction format. The 32-bit instruction has five fields: op, rs1, rs2, funct3 and imm. The first three fields op, rs1, and rs2 are like those of R-type instructions. The imm fields hold the 12-bit immediate. The 12-bit immediate is split into two sets as shown below. The operation is determined by the opcode and funct3, highlighted in blue. The operands are specified in the three fields rs1, rs2 and imm.

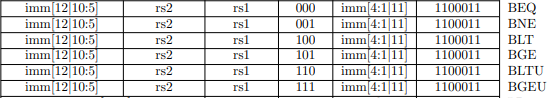


Instruction format for some of the S-type instructions are shown below:

### B-Type Instructions

The name B-type is short for branch-type. This format is used only with branch instructions. B-type instructions use two register operands and one immediate operand. Figure below shows the B-type machine instruction format. The 32-bit instruction has five fields: op, rs1, rs2, funct3 and imm. The first three fields op, rs1, and rs2 are like those of R-type instructions. The imm fields hold the 12-bit immediate. The 12-bit immediate is split into two sets as shown below. The operation is determined by the opcode and funct3, highlighted in blue. The operands are specified in the three fields rs1, rs2 and imm.



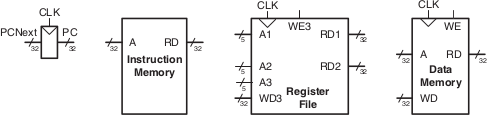
Instruction format for some of the B-type instructions are shown below:

To interpret machine language, one must decipher the fields of each 32-bit instruction word. Different instructions use different formats, but all formats start with a 7-bit opcode field. Thus, the best place to begin is to look at the opcode.

## State Elements

A good way to design a complex system is to start with hardware containing the state elements. These elements include the memories and the architectural state (the program counter and registers). Then, add blocks of combinational logic between the state elements to compute the new state based on the current state. The instruction is read from part of memory; load and store instructions then read or write data from another part of memory.

Hence, it is often convenient to partition the overall memory into two smaller memories, one containing instructions and the other containing data. Figure below shows a block diagram with the four state elements: the program counter, register file, and instruction and data memories.



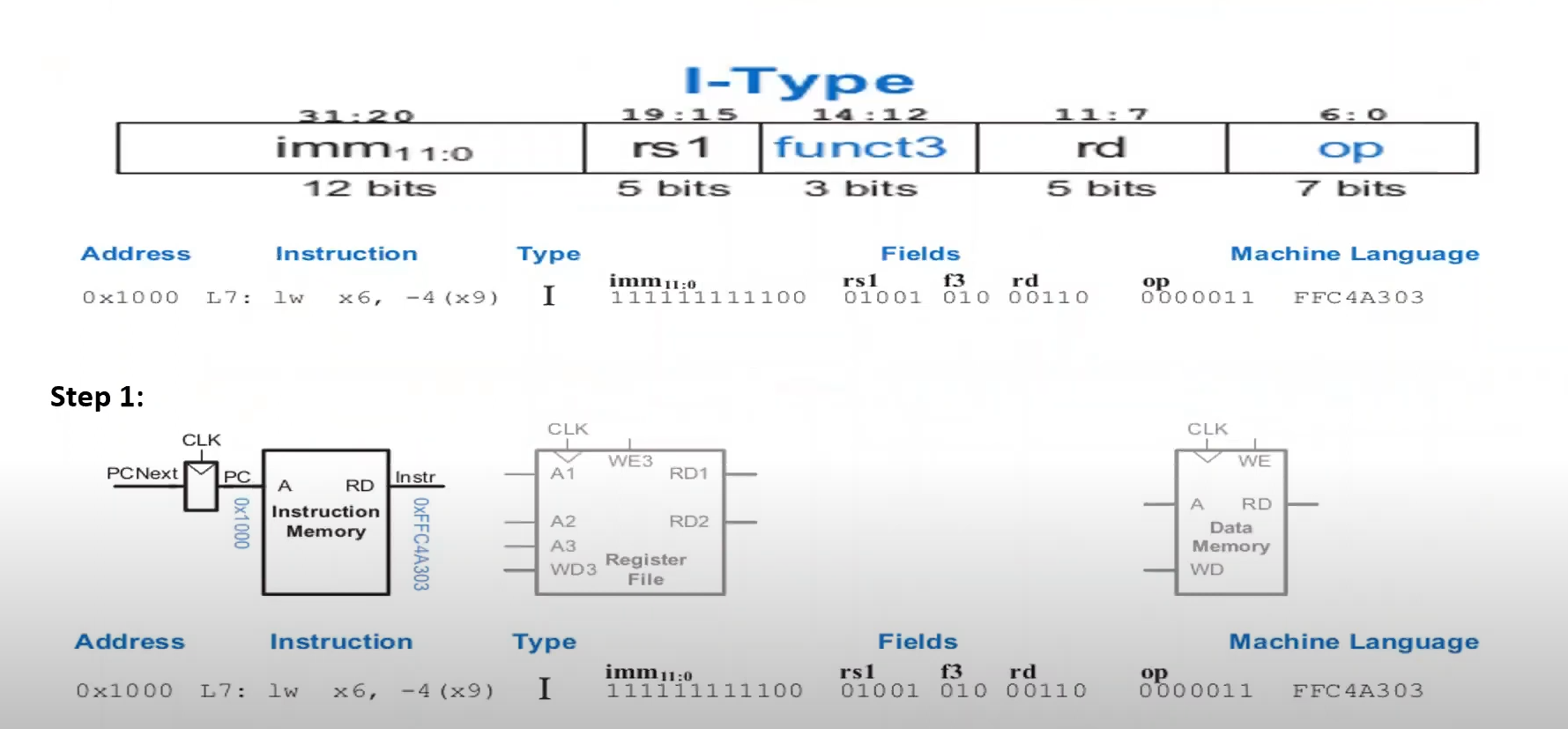
In above Figure heavy lines are used to indicate 32-bit data busses. Medium lines are used to indicate narrower busses, such as the 5-bit address busses on the register file. Narrow blue lines are used to indicate control signals, such as the register file write enable. We will use this convention throughout the chapter to avoid cluttering diagrams with bus widths. Also, state elements usually have a reset input to put them into a known state at start-up. Again, to save clutter, this reset is not shown.

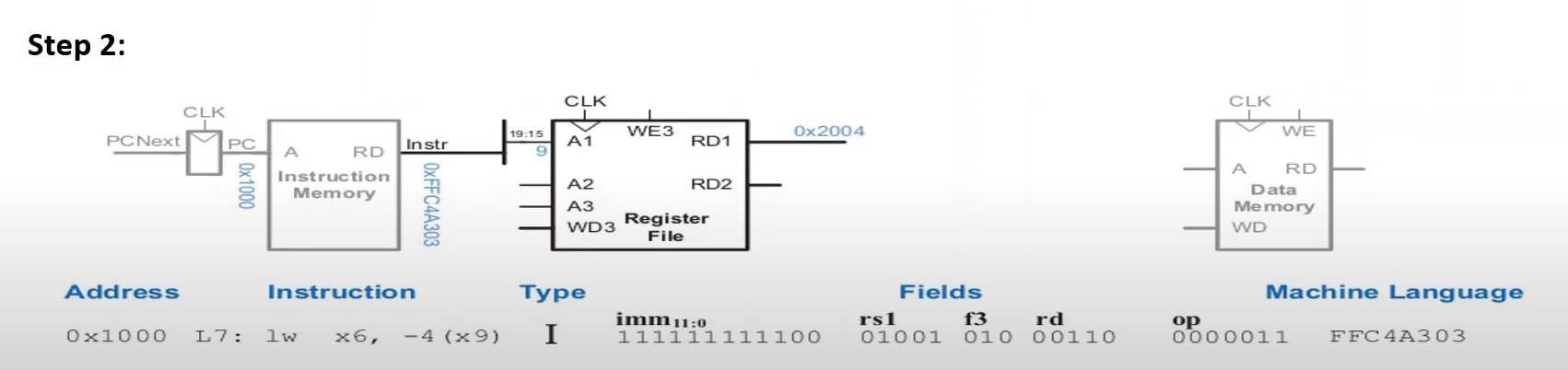
The **program counter** is an ordinary 32-bit register. Its output, PC, points to the current instruction. Its input, PCNext, indicates the address of the next instruction.

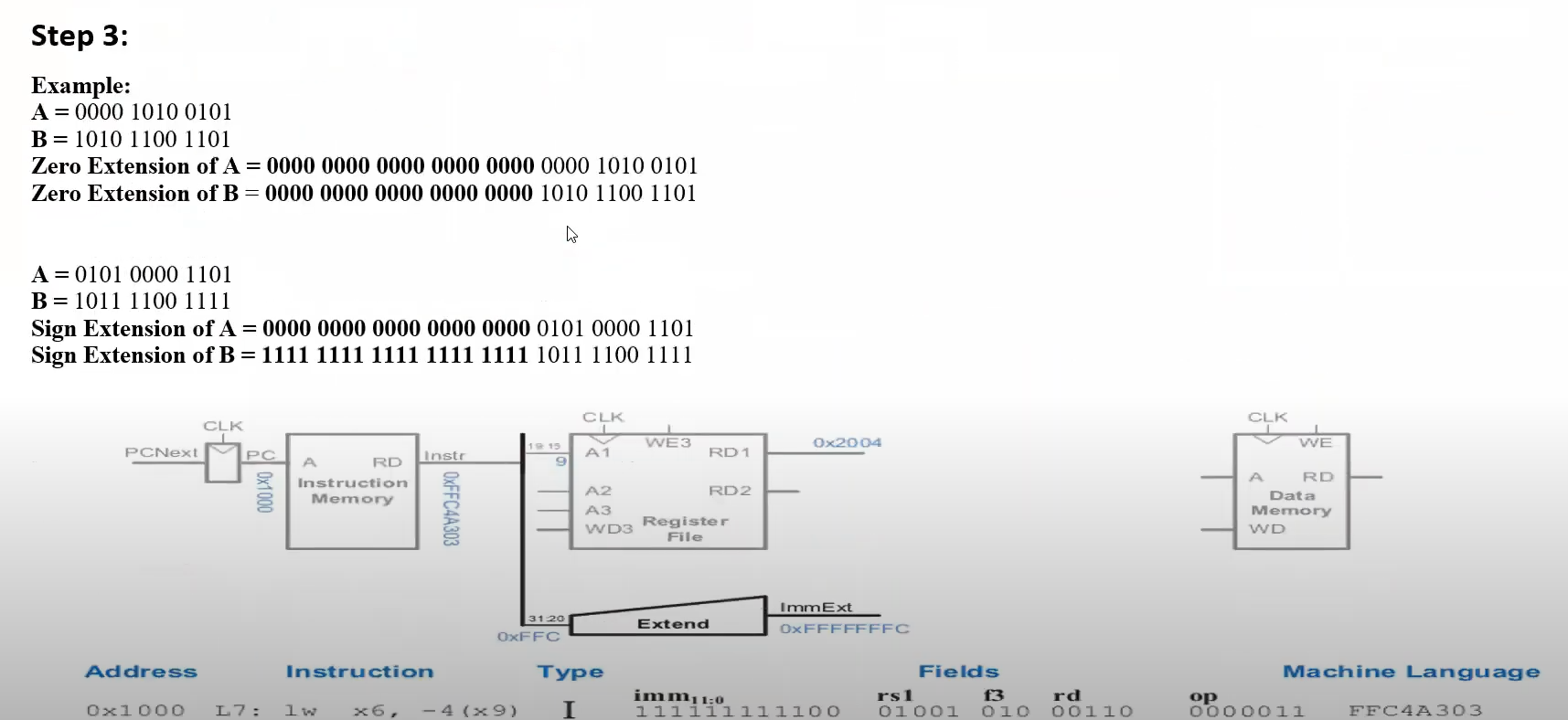
The **instruction memory** has a single read port. It takes a 32-bit instruction address input, A, and reads the 32-bit data (i.e., instruction) from that address onto the read data output, RD.

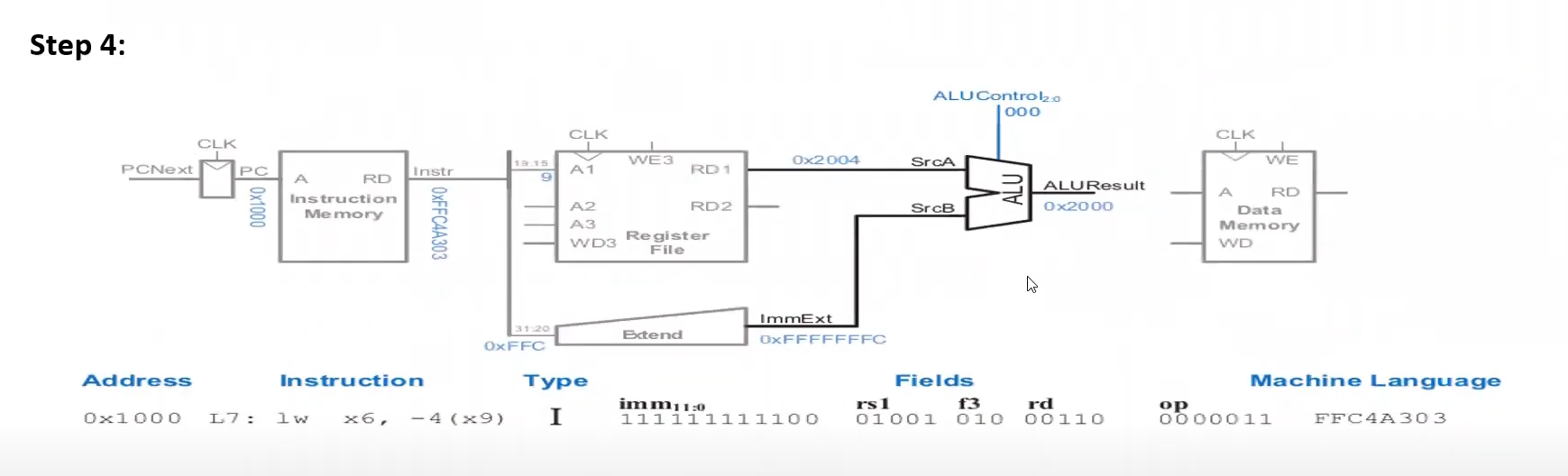
The 32-element × 32-bit **register file** has two read ports and one write port. The read ports take 5-bit address inputs, A1 and A2, each specifying one of 5 = 32 registers as source operands. They read the 32-bit register values onto read data outputs RD1 and RD2, respectively. The write port takes a 5-bit address input, A3; a 32-bit write data input, WD; a write enable input, WE3; and a clock. If the write enable is 1, the register file writes the data into the specified register on the rising edge of the clock.

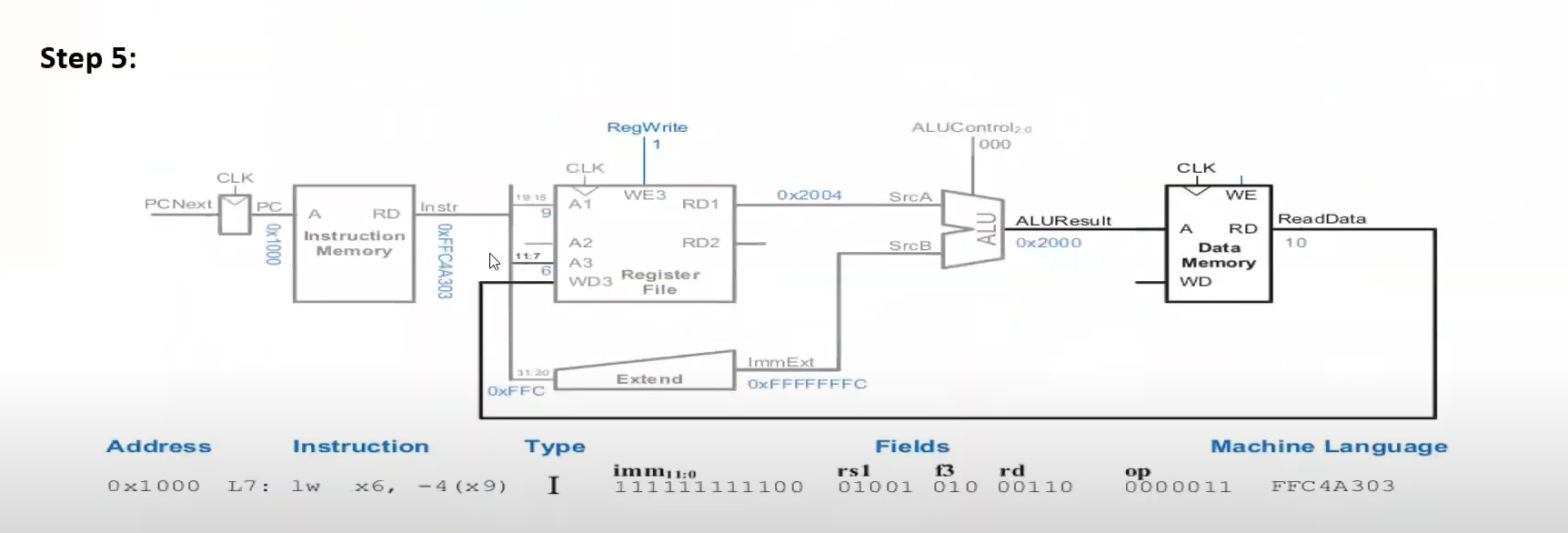
2

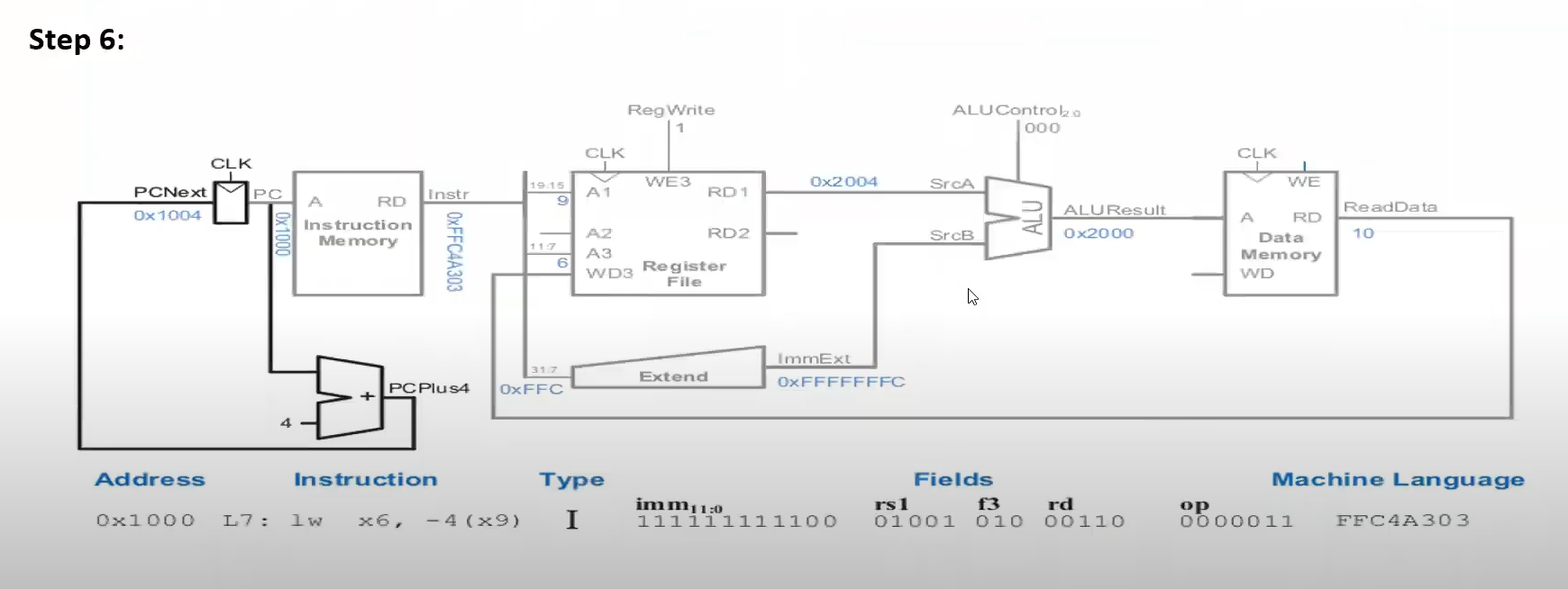


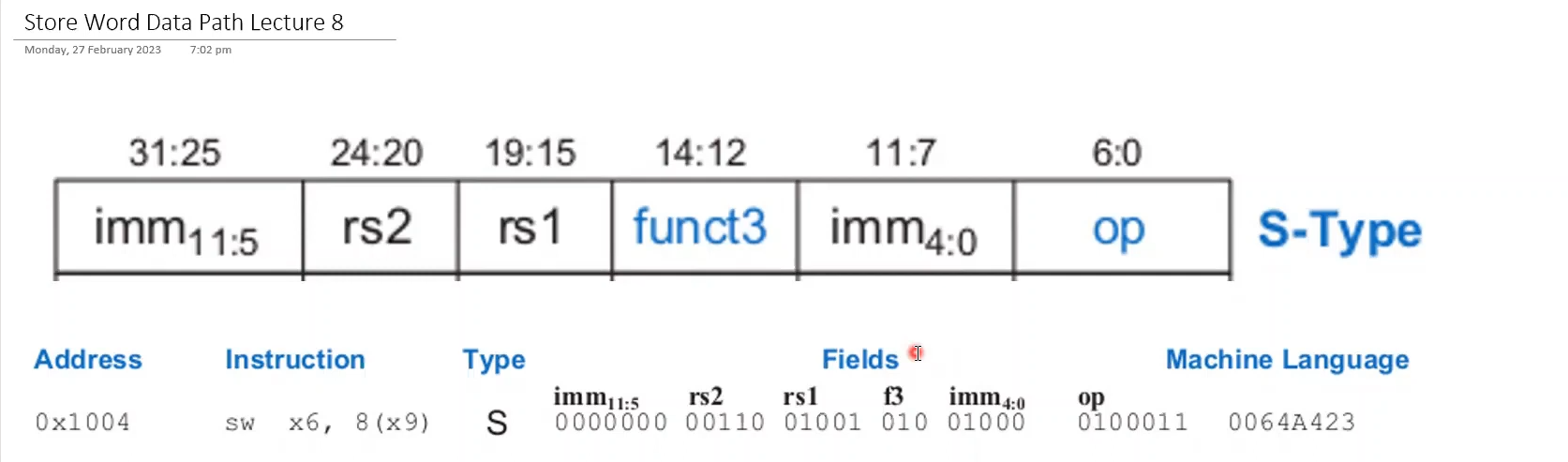


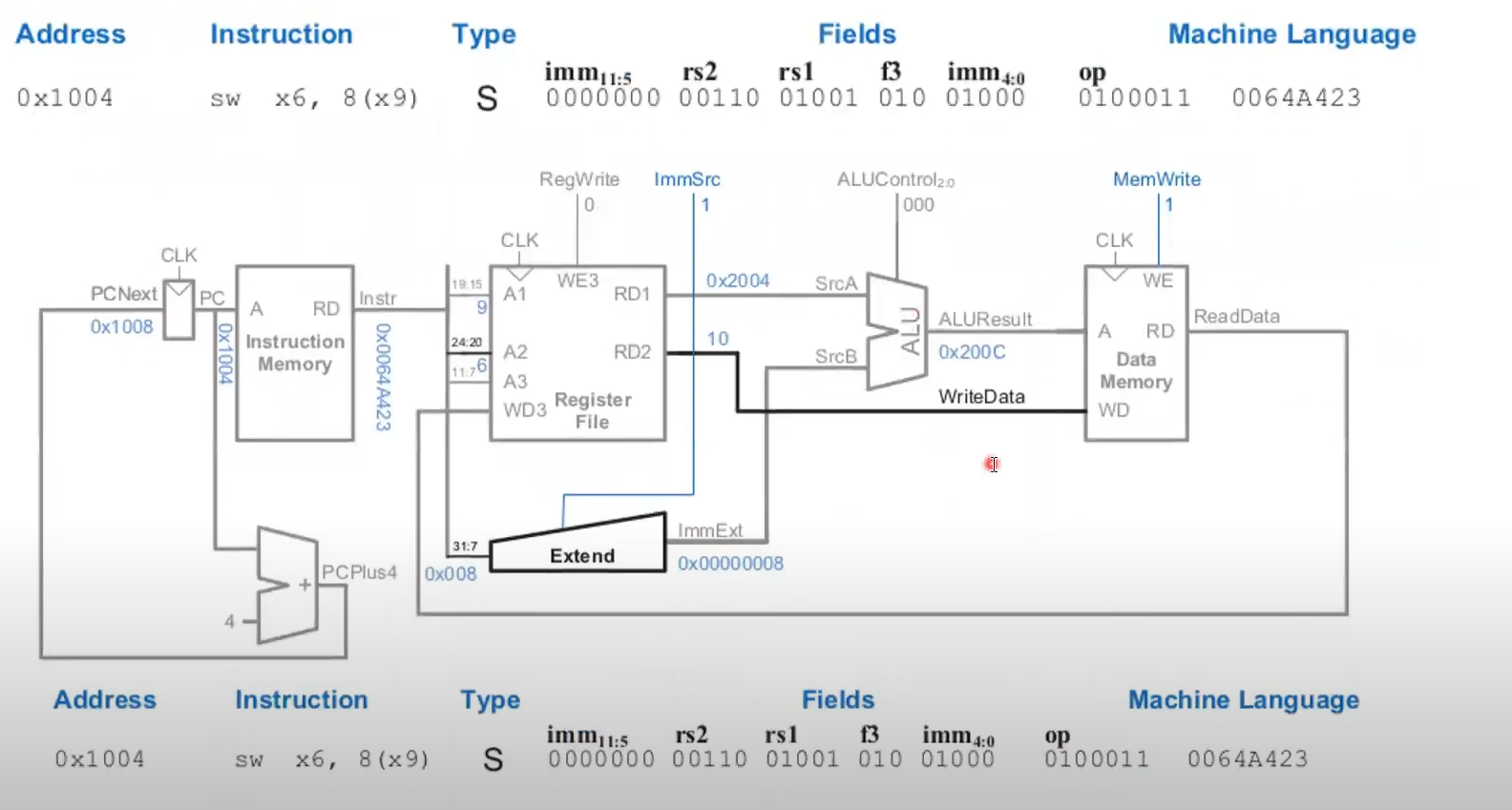




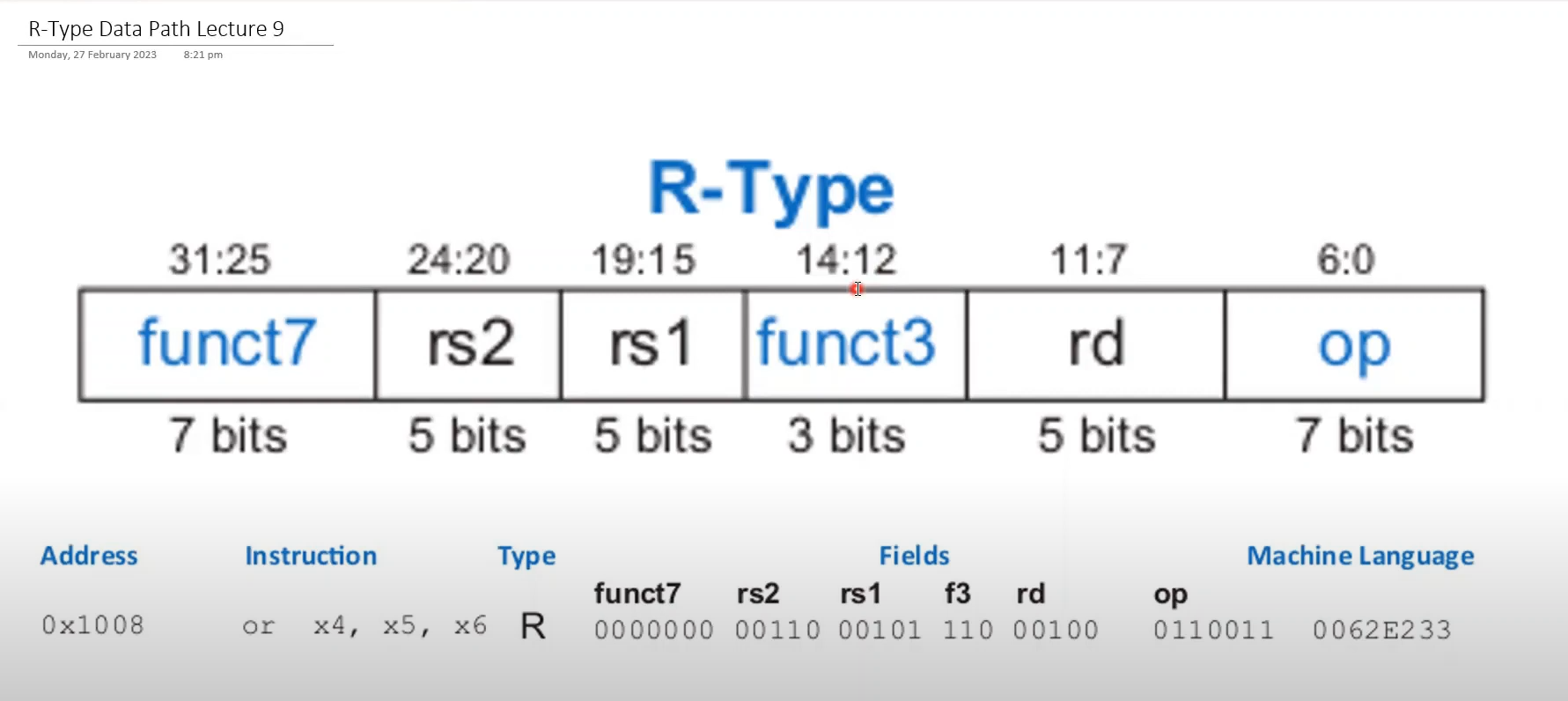


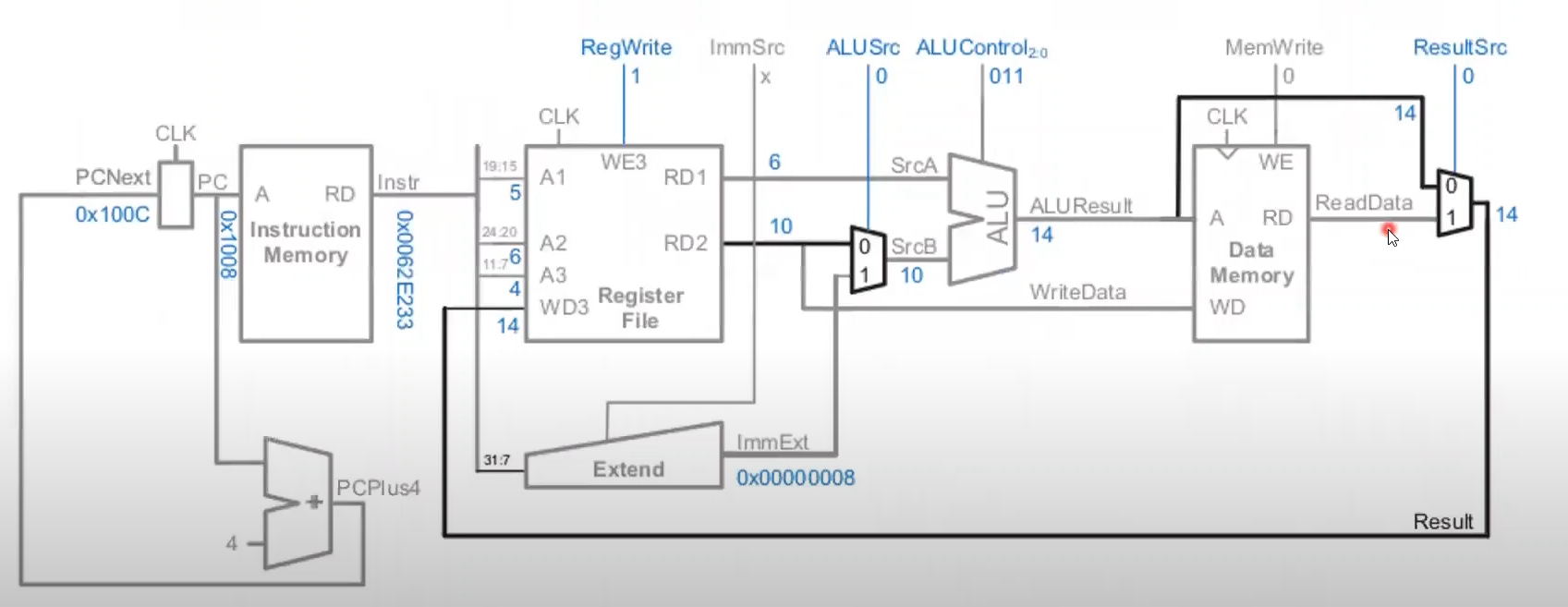




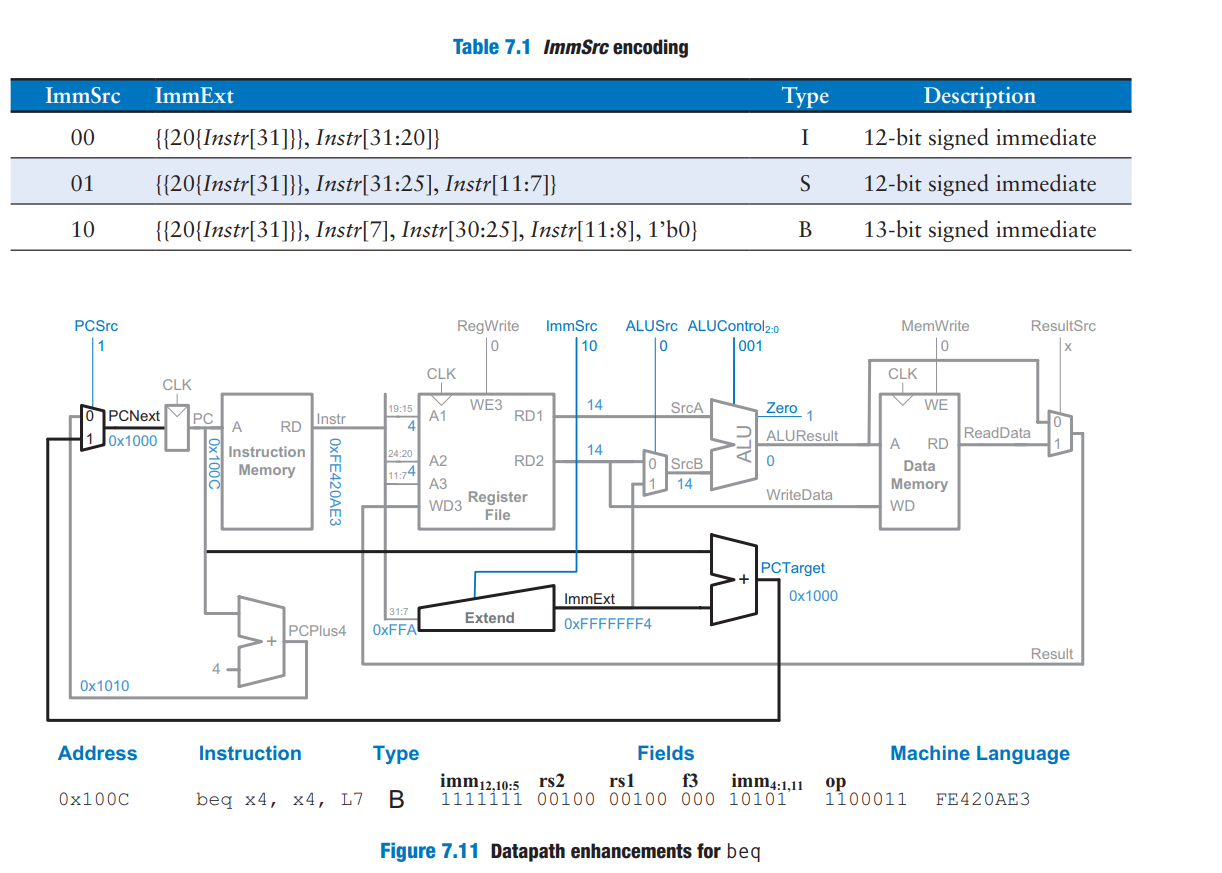


**Designing DataPath of R-Type RISC-V Instruction**

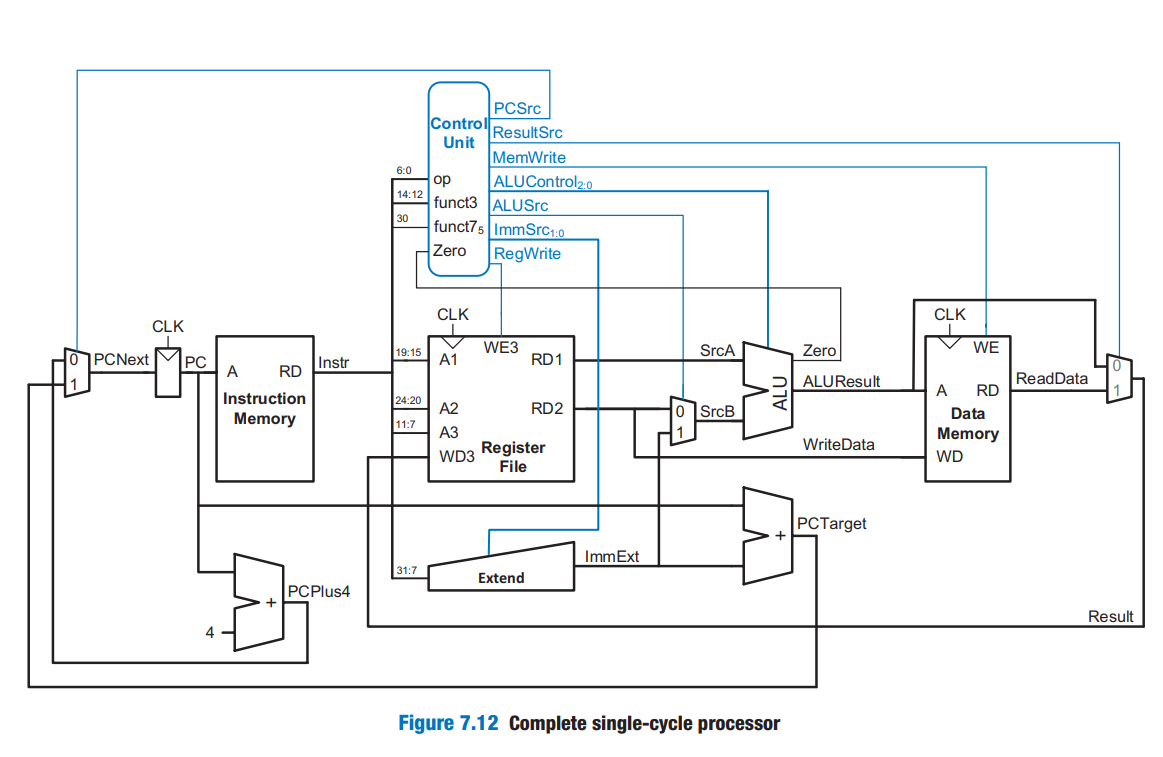




**Designing DataPath of B-Type RISC-V Instruction**



**Complete single-cycle processor**

****

**GitHub Repository:** You can explore the project in detail at my GitHub repository:

[**https://github.com/RAOUMERTC73/RISC-V-Single-Cycle-Core-in-Verilog**](https://github.com/RAOUMERTC73/RISC-V-Single-Cycle-Core-in-Verilog)