LAB # 7
Designing Microarchitecture-II

Objective:

- To design RISC-V Microarchitecture for Load Instruction.

System Module (Hardware/Software)

- Visual Studio Code

Theoretical Background:

Introduction

Load word (lw) is an I-type instruction. The **LW instruction** loads data from the data memory through a specified address, with a possible offset, to the destination register. The name I-type is short for immediate-type. I-type instructions use two register operands and one immediate operand. 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 operation is determined by the opcode and funct3, highlighted in blue. The operands are specified in the two fields rs1, rd, and imm. rs1 and imm are always used as source operands. rd is used as a destination.

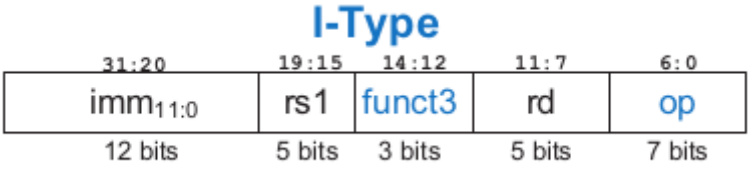


Figure below shows an example for LW instruction represented in assembly code and machine code.

Address	Instruction	Type	Fields				Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

I-type instructions have a 12-bit immediate field, but the immediates are used in 32-bit operations. For example, lw adds a 12-bit offset to a 32-bit base register. What should go in the upper half of the 32 bits?

For positive immediates, the upper half should be all 0's, but for negative immediates, the upper half should be all 1's. This is called sign extension. An N-bit two's complement number is sign-extended to an M-bit number (M >N) by copying the sign bit (most significant bit) of the N-bit number into all of the upper bits of the M-bit number. Sign-extending a two's complement number does not change its value.

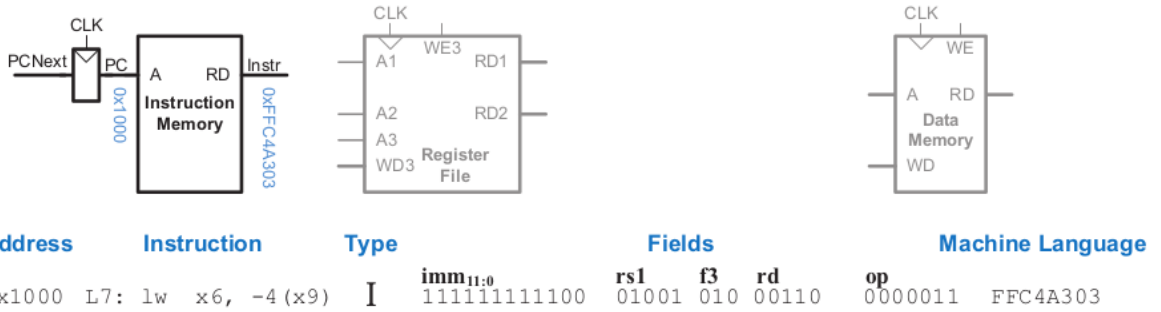
Example:

A = 0000 1010 0101
B = 1010 1100 1101
Zero Extension of A = 0000 0000 0000 0000 0000 0000 1010 0101
Zero Extension of B = 0000 0000 0000 0000 0000 1010 1100 1101

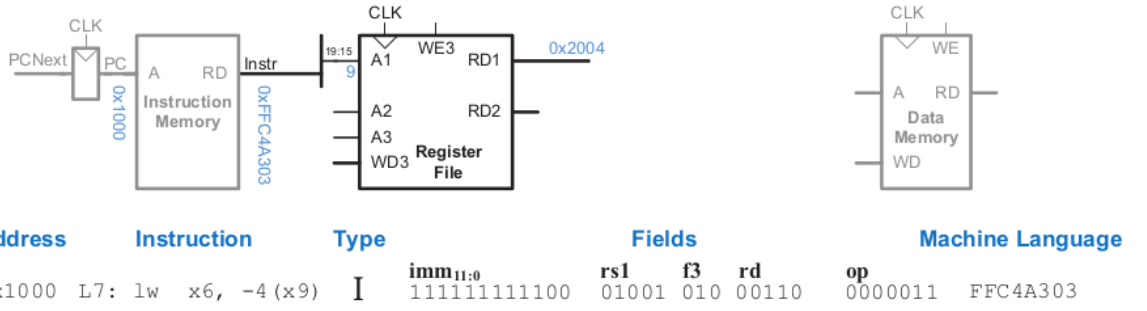
A = 0101 0000 1101
B = 1011 1100 1111
Sign Extension of A = 0000 0000 0000 0000 0000 0101 0000 1101
Sign Extension of B = 1111 1111 1111 1111 1111 1011 1100 1111

DataPath for Load Instruction

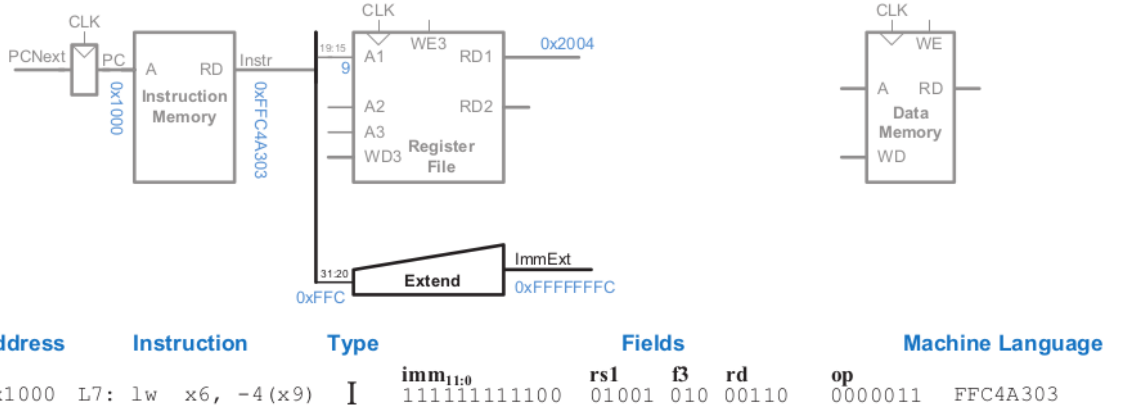
This section gradually develops the single-cycle datapath, adding one piece at a time to the state elements. The program counter (PC) register contains the address of the instruction to execute. The first step is to read the instruction from instruction memory. Figure below shows that the PC is simply connected to the address input of the instruction memory. The instruction memory reads out, or fetches, the 32-bit instruction, labeled Instr.



For a load instruction, the next step is to read the source register containing the base address. This register is specified in the rs1 field of the instruction, Instr[19:15]. These bits of the instruction are connected to the address input of one of the register file read ports, A1, as shown in Figure below. The register file reads the register value onto RD1.

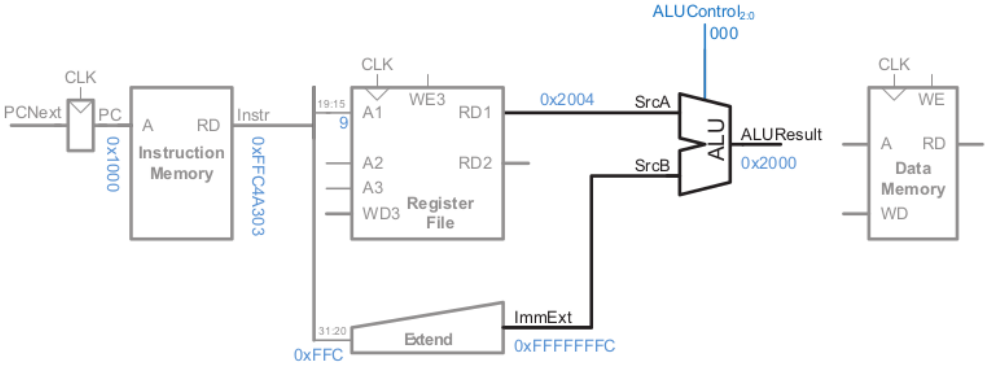


The load instruction also requires an offset. The offset is stored in the immediate field of the instruction, Instr[31:20]. Because the 12-bit immediate might be either positive or negative, it must be sign-extended to 32 bits, as shown in Figure below.



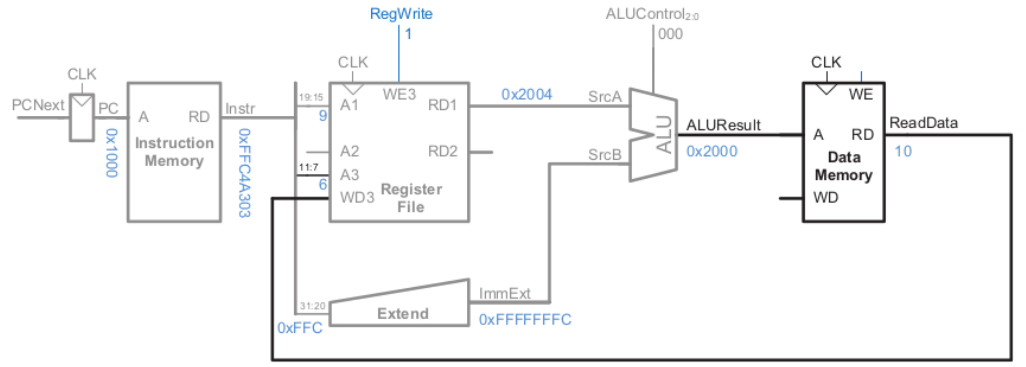
The 32-bit sign-extended value is called ImmExt. The sign extension simply copies the sign bit (most significant bit) of a short input into all of the upper bits of the longer output. Specifically, ImmExt[15:0] = Instr[31:20] and ImmExt[31:16] = Instr[31].

The processor must add the base address to the offset to find the address to read from memory. Figure below introduces an ALU to perform this addition. The ALU receives two operands, SrcA and SrcB. SrcA comes from the register file, and SrcB comes from the sign-extended immediate. The ALU can perform many operations. The 3-bit ALUControl signal specifies the operation. The ALU generates a 32-bit ALUResult and a Zero flag, which indicates whether ALUResult == 0. For a load instruction, the ALUControl signal should be set to 000 to add the base address and offset.



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

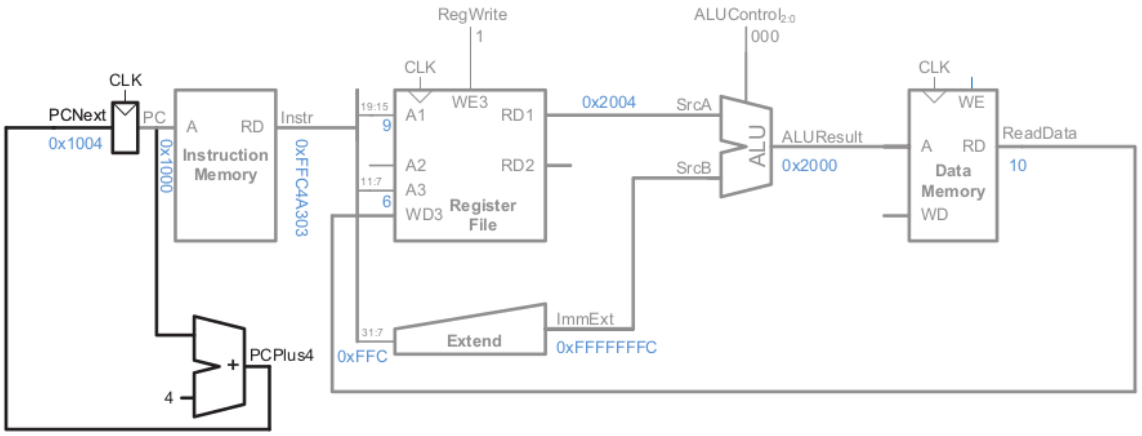
ALUResult is sent to the data memory as the address for the load instruction. The data is read from the data memory onto the ReadData bus, then written back to the destination register in the register file at the end of the cycle, as shown in Figure below. Port 3 of the register file is the write port.



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

The destination register for the load instruction is specified in the rd field, Instr[11:7], which is connected to the port 3 address input, A3, of the register file. The ReadData bus is connected to the port 3 write data input, WD3, of the register file. A control signal called RegWrite is connected to the port 3 write enable input, WE3, and is asserted during a load instruction so that the data value is written into the register file.

The write takes place on the rising edge of the clock at the end of the cycle. While the instruction is being executed, the processor must compute the address of the next instruction, PCNext. Because instructions are 32 bits = 4 bytes, the next instruction is at PC + 4. Figure below shows that datapath uses another adder to increment the PC by 4. The new address is written into the program counter on the next rising edge of the clock. This completes the datapath for the load instruction.



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

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 Verilog code for the sign extend and adder block. Make sure to name the input and output ports correctly with the correct number of bits. Attach the code.**
2. **Write a complete data path for Load 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 datapath for Load Instruction in Verilog.