MOD_4 THE PROCESSOR

Processor Microarchitecture taking MIPS microprocessor as example.

- Microarchitecture is the specific arrangement of registers , ALUs, finite state machines (FSMs), memories, and other logic building blocks needed to implement an architecture

Architectural State and Instruction Set

- Computer architecture is defined by its instruction set and architectural state.
- The architectural state for the MIPS processor consists of the program counter and the 32 registers. Any MIPS microarchitecture must contain all of this state.
- Based on the current architectural state, the Processor executes a particular instruction with a particular set of data to produce a new architectural state.

.

- To keep the microarchitectures easy to understand, we consider only a subset of the MIPS instruction set.
- Specifically, we handle the following instructions:
- ► R-type arithmetic/logic instructions: add
- ► Memory instructions: lw
- Add $s0,$S1,$S2   OP,rs,rt,rd,shmt,funct
-   6 5 5 5   5   6
- Lw $S3, -24($S4)   OP,rs,rt,immd
-   6   5 5   16

Design Process

- We will divide our microarchitectures into two interacting parts: the datapath and the control.
- The datapath operates on words of data. It contains structures such as memories, registers, ALUs, and multiplexers.
- MIPS is a 32-bit architecture, so we will use a 32-bit datapath.
- The control unit receives the current instruction from the datapath and tells the datapath how to execute that instruction.
- Specifically, the control unit produces multiplexer select, register enable, and memory write signals to control the operation of the datapath.

- 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 7.1 shows a block diagram with the four state elements: the program counter, register file, and instruction and data memories.

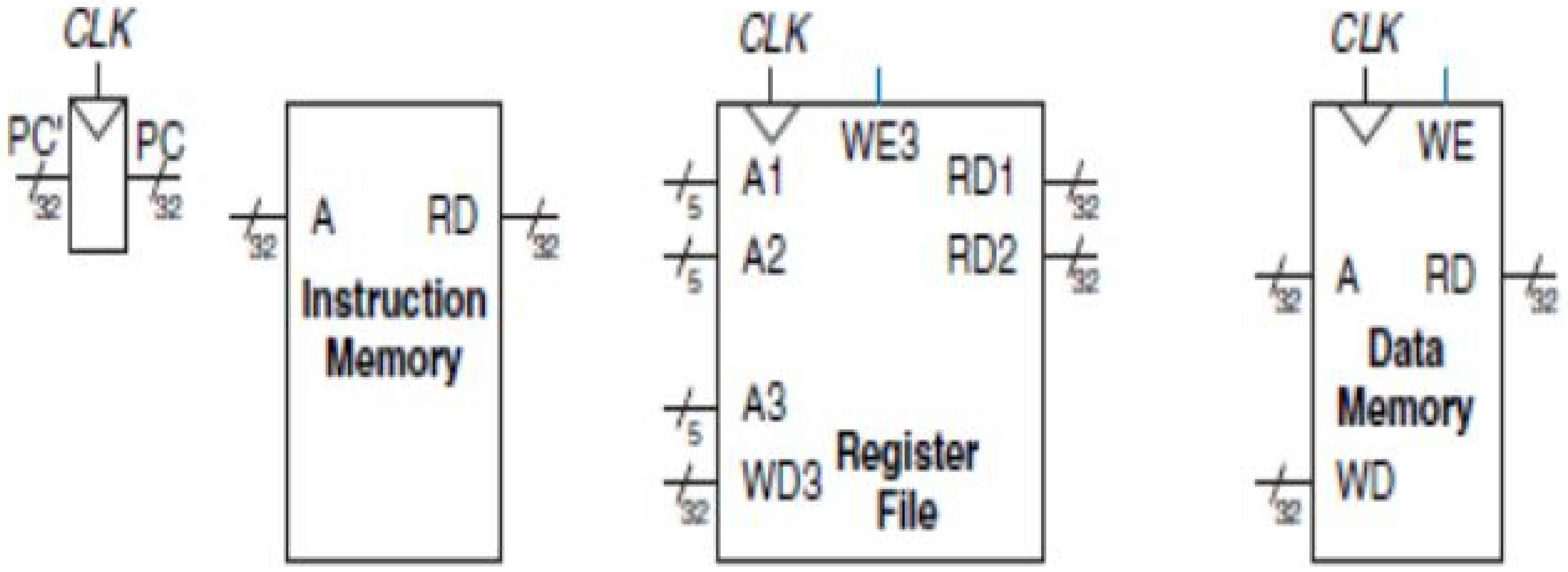

Figure 7.1 State elements of MIPS processor

- The **program counter** is an ordinary 32-bit register. Its output, PC points to the current instruction. Its input PC′ 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 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.
- The **Data memory** has a single read/write port. If the write enable WE is 1 it writes data WD into address A on the rising edge of the clock. If the write enable is 0 it reads address A onto RD.

MIPS Microarchitectures

- Three microarchitectures for the MIPS processor architecture: single-cycle, multicycle, and pipelined.
- The single-cycle microarchitecture executes an entire instruction in one cycle. It is easy to explain and has a simple control unit.
- The cycle time is limited by the slowest instruction.
- The multicycle microarchitecture executes instructions in a series of shorter cycles.

SINGLE-CYCLE PROCESSOR (Building a data path)

- Design a MIPS microarchitecture that executes instructions in a single cycle.
- We begin constructing the datapath by connecting the state elements with combinational logic that can execute the various instructions.
- Control signals determine which specific instruction is carried out by the datapath at any given time.
- The controller contains combinational logic that generates the appropriate control signals based on the current instruction.

Single-Cycle Datapath

- The program counter (PC) register contains the address of the instruction to execute.
- The first step is to read this instruction from instruction memory. Figure 7.2 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.

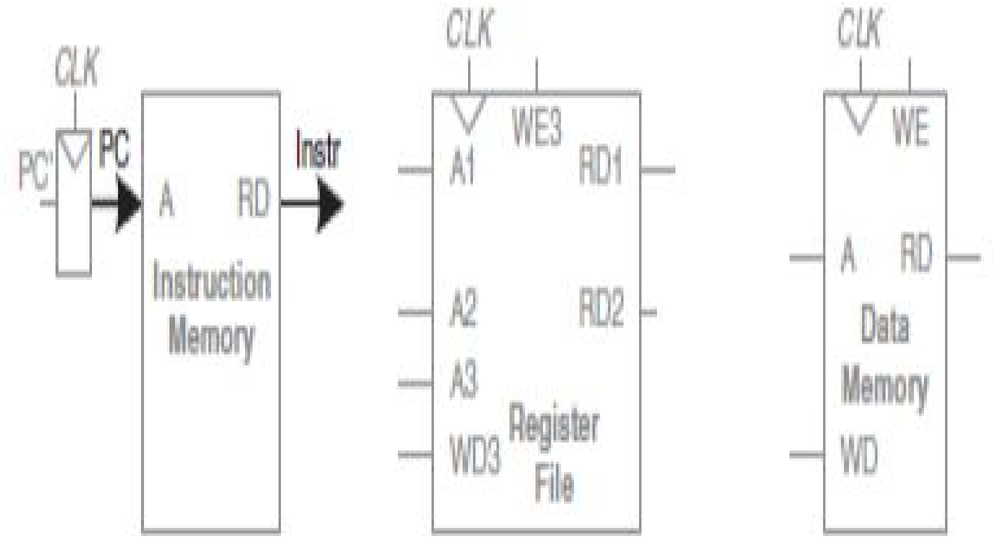

Figure 7.2 Fetch instruction from memory

•

- The processor's actions depend on the specific instruction that was fetched.
- First we will work out the datapath connections for the lw instruction. Then we will consider how to generalize the datapath to handle the other instructions.
- For a lw instruction, the next step is to read the source register containing the base address. This register is specified in the rs field of the instruction, Instr25:21. These bits of the instruction are connected to the address input of one of the register file read ports, A1, as shown in Figure 7.3.
- The register file reads the register value onto RD1.
- The lw instruction also requires an offset. The offset is stored in the immediate field of the instruction, Instr15:0. Because the 16-bit immediate might be either positive or negative, it must be sign-extended to 32 bits, as shown in Figure 7.4. The 32-bit sign-extended value is called SignImm.
- Recall from Section 1.4.6 that 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, SignImm15:0 = Instr15:0 and SignImm31:16 = Instr15.

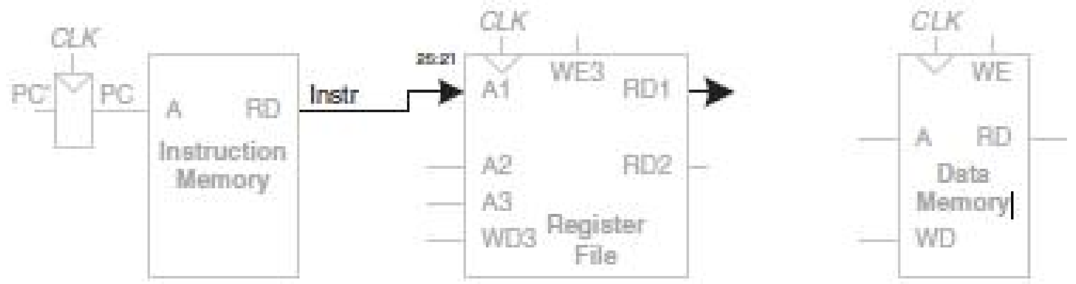

Figure 7.3 Read source operand from register file

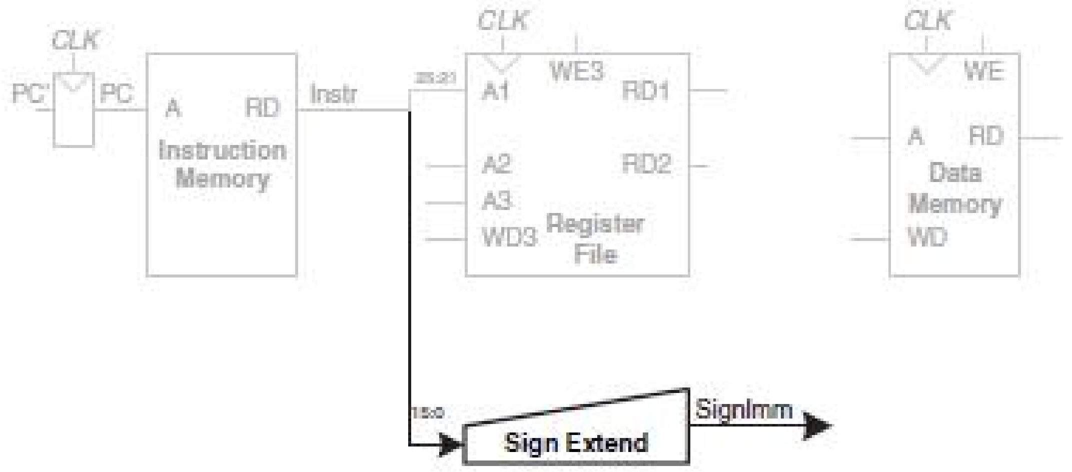

Figure 7.4 Sign-extend the immediate

•

- The processor must add the base address to the offset to find the address to read from memory. Figure 7.5 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 3-bit ALUControl signal specifies the operation. The ALU generates a 32-bit ALUResult and a Zero flag, that indicates whether ALUResult == 0.
- For a lw instruction, the ALUControl signal should be set to 010 to add the base address and offset. ALUResult is sent to the data memory as the address for the load instruction, as shown in Figure 7.5.
- 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 7.6. Port 3 of the register file is the write port.

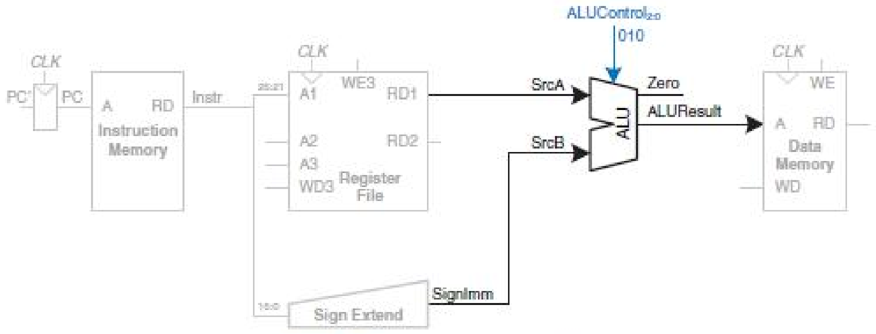

Figure 7.5 Compute memory address

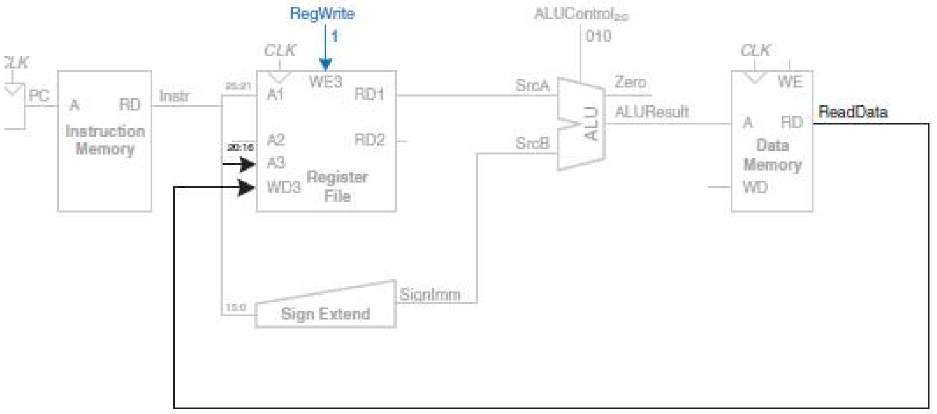

Figure 7.6 Write data back to register file

- The destination register for the lw instruction is specified in the rt field, Instr20:16, 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 lw 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, PC′. Because instructions are 32 bits = 4 bytes, the next instruction is at PC + 4. Figure 7.7 uses another adder to increment the PC by 4.
- The new address is written into the program

Single-Cycle Control

- The control unit computes the control signals based on the opcode and funct fields of the instruction, Instr31:26 and Instr5:0. Figure 7.11 shows the entire single-cycle MIPS processor with the control unit attached to the datapath.
- Most of the control information comes from the opcode, but R-type instructions also use the funct field to determine the ALU operation.
- Thus, we will simplify our design by factoring the control unit into two blocks of combinational logic, as shown in Figure 7.12. The main decoder computes most of the outputs from the opcode.
- It also determines a 2-bit ALUOp signal. The ALU decoder uses this ALUOp signal in conjunction with the funct field to compute ALUControl.
- The meaning of the ALUOp signal is given in Table 7.1.

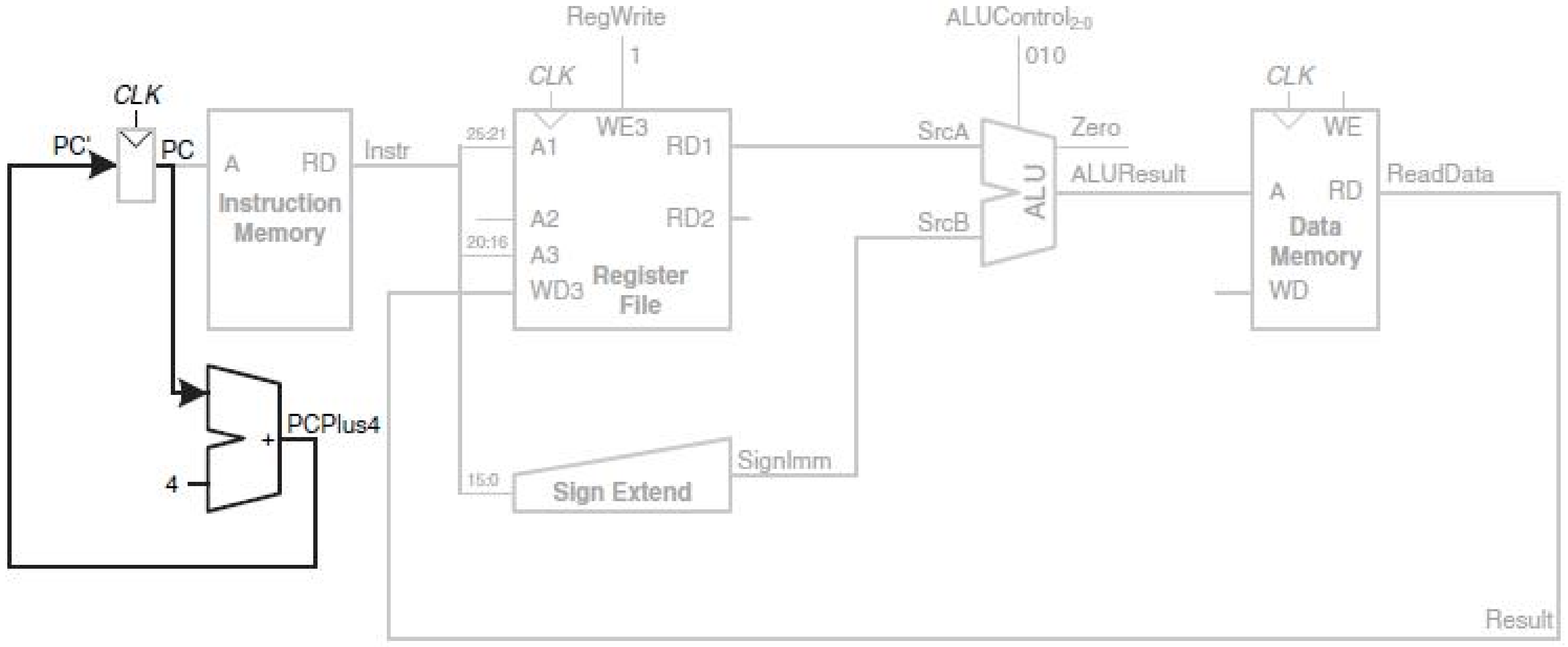

Figure 7.7 Determine address of next instruction for PC

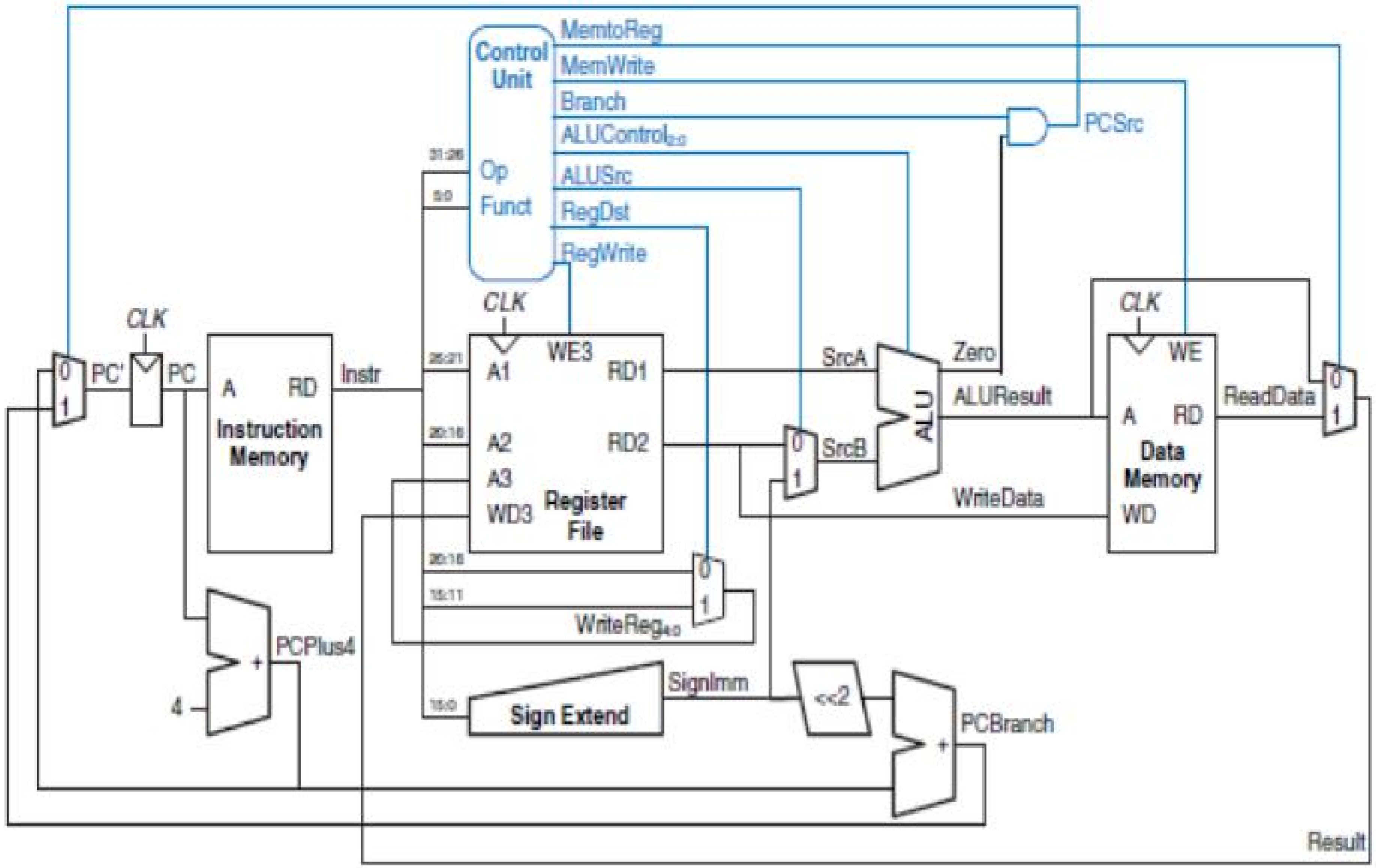

Figure 7.11 Complete single-cycle MIPS processor

Table 7.1 ***ALUOp*** **encoding**

ALUOp	Meaning
00	add
01	subtract
10	look at `funct` field
11	n/a

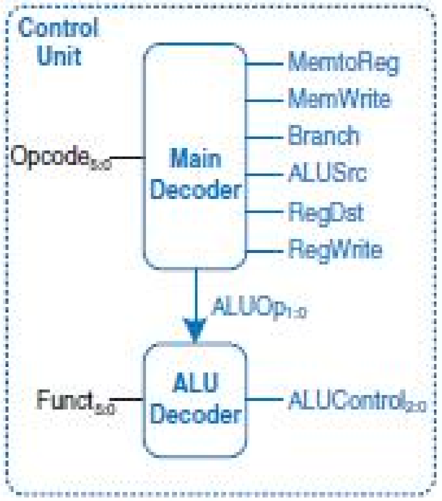

Figure 7.12 **Control unit internal structure**

- Table 7.2 is a truth table for the ALU decoder. Recall that the meanings of the three ALUControl signals were given in Table 5.1.
- Because ALUOp is never 11, the truth table can use don’t care’s X1 and 1X instead of 01 and 10 to simplify the logic. When ALUOp is 00 or 01, the ALU should add or subtract, respectively. When ALUOp is 10, the decoder examines the funct field to determine the ALUControl.
- Note that, for the R-type instructions we implement, the first two bits of the funct field are always 10, so we may ignore them to simplify the decoder.
- The control signals for each instruction were described as we built the datapath. Table 7.3 is a truth table for the main decoder that summarizes the control signals as a function of the opcode.

Table 7.2 ALU decoder truth table

ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

Table 7.3 Main decoder truth table

Instruction	Opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	ALUOp
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01