Chapter - 5

The Processor: Datapath and Control

Processor Implementation Style

- Single Cycle
  - Perform each instruction in one cycle
  - Clock cycle must be long enough for slowest instruction, therefore
  - Disadvantage: only as fast as slowest instruction.
- Multi Cycle
  - Break, fetch/execute cycle into multiple steps
  - Performs 1 step in each clock cycle
  - Advantages: each instruction uses only as many cycle as it needs
- Pipelined
  - Executes each instrcution in multiple steps
  - Performs 1 step/per instruction in each clock cycle
  - Process multiple instructions in parallel.

ALU

- The **Arithmetic Logic Unit (ALU)** is the heart of any CPU. An ALU performs three kinds of operations, i.e.
- Arithmetic operations such as Addition/Subtraction,
- Logical operations such as AND, OR, etc. and
- Data movement operations such as Load and Store

- An instruction execution in a CPU is achieved by the movement of data/datum associated with the instruction. This movement of data is facilitated by the Datapath.

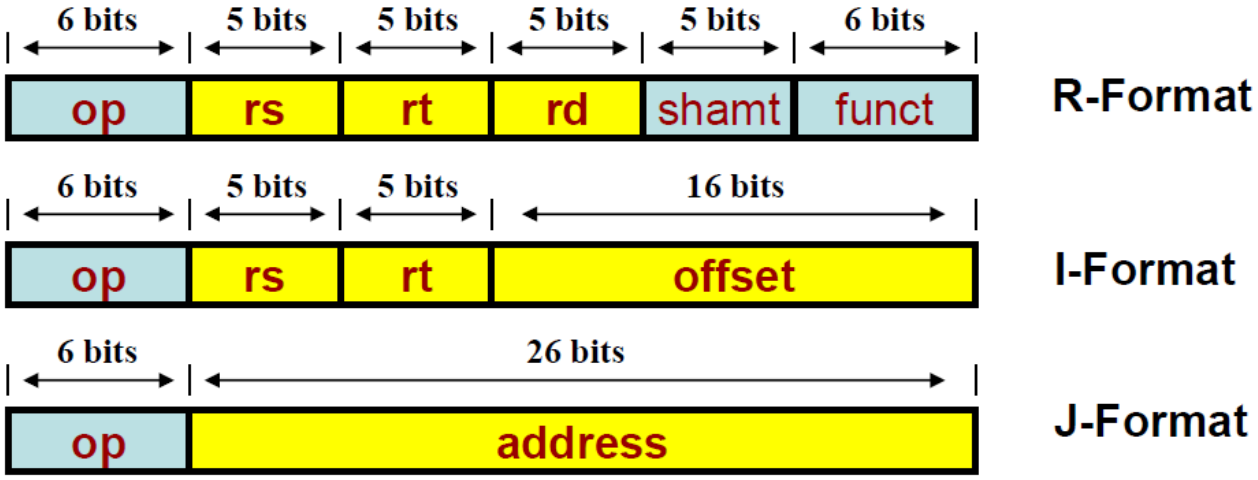
5.1 Introduction

A subset of the core MIPS instruction set:

- The memory-reference instruction – lw, sw
- The arithmetic-logical instruction – add, sub, and, or, slt
- The branch instructions - branch equal (beq), jump (j)

Implementing MIPS

- We're ready to look at an implementation of the MIPS instruction set
- Simplified to contain only
  - arithmetic-logic instructions: `add, sub, and, or, slt`
  - memory-reference instructions: `lw, sw`
  - control-flow instructions: `beq, j`



An overview of the implementation

For every instruction, the first two steps are identical:

1. Sent the PC to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read.

An overview of the implementation (Cont.)

Even across different instruction classes there are some similarities: Example

- All instruction classes use the ALU after reading the registers.
  - Memory-reference – address calculation
  - Arithmetic-logic – operation execution
  - Branch – comparison.

CPU Data Path

What is a datapath? Why a datapath?

- ***"A class has 26 boys and 30 girl students. Find the total number of students?"***
  - Identify how many boys, how many girls
  - Understand that the sum to be solved using addition
  - In your notebook, on a new page, create a workspace
  - In the workspace, write 26 in a line and 30 in another line
  - Now do the addition, if required use rough space for work out
  - Rewrite this as result = 56
- Oh! you are a geek. Your computer also needs to do all these or more steps to solve this problem.

- In fact, in the process of solving the problem, the CPU has to get this instruction from memory
  - know the address of the memory location wherein the data about boys and girls are kept
  - decode the instruction to be ADD
  - Get the boys girls data from memory to CPU workspace (Registers)
  - Navigate this data to ALU so that addition can be carried out
  - Write the result from ALU to the Result Space.

- So, if we are using our brain and notepad space, the CPU uses Registers, ALU, MEMORY, etc. The functional components that make up the requirements of all the instruction execution. The DATAPTH is a collection of registers, ALUs, multiplexers, status registers and their interconnection are called DATAPATH.

- **A DATAPATH is the collection of state elements, computation elements, and interconnections that together provide a conduit for the flow and transformation of data in the processor during execution**

An overview of the implementation (Cont.)

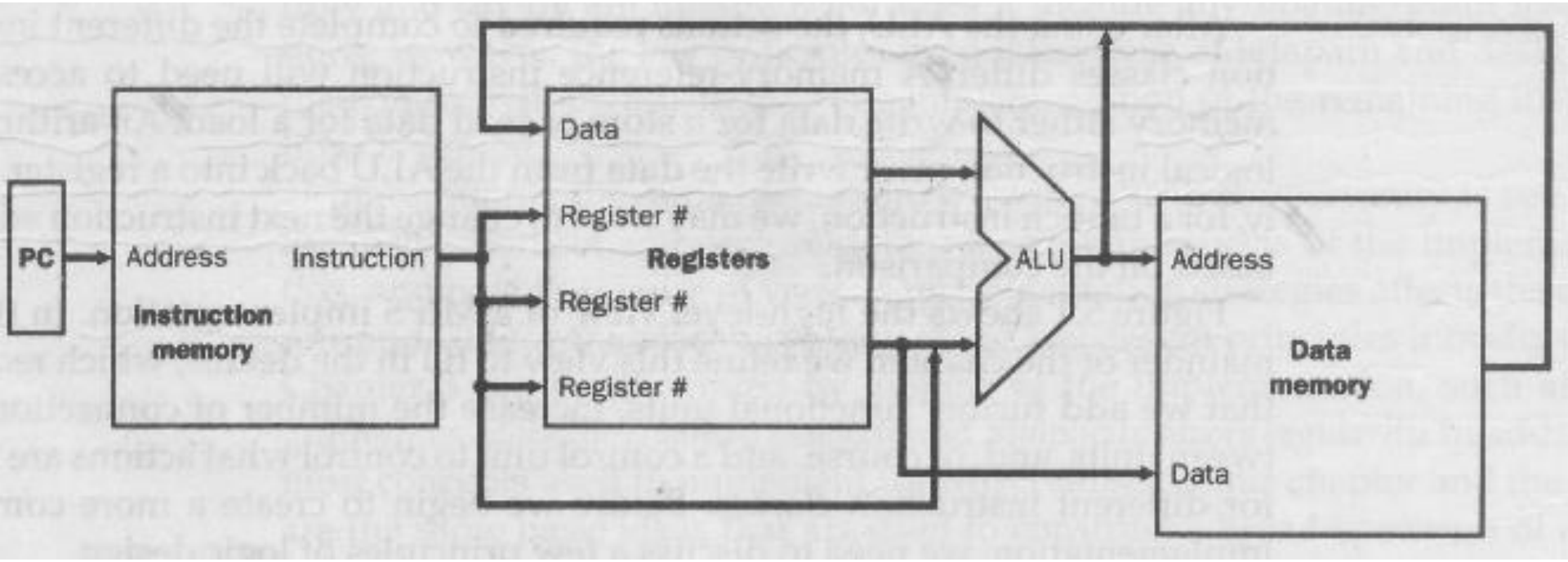


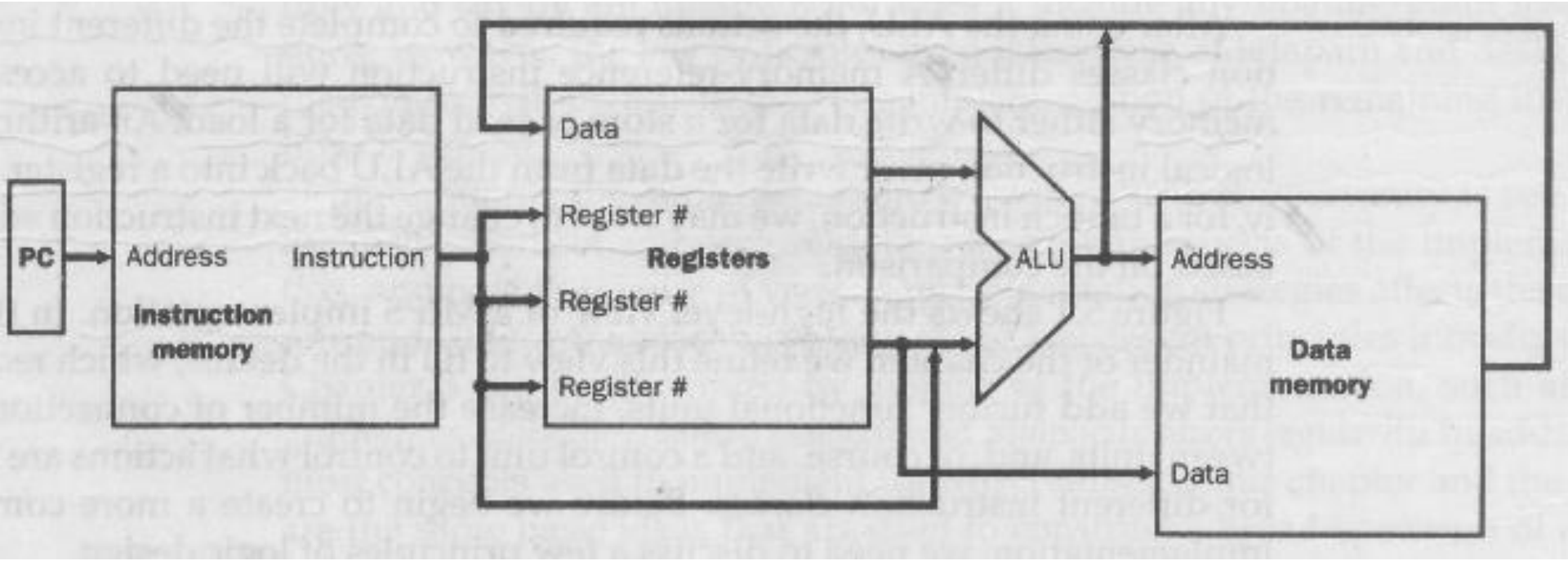
Fig. – 5.1 An abstract view of the implementation.

Essentially a DATAPATH consists of the following elements.

- **ALU** – one or more to carry out the computation. ALU is not only used in data operations but also in address calculation too.
- **Instruction Register and decoder** – to decode what instruction to be executed and how to execute the instruction.
- **Program Counter** – Always points to the next instruction to be executed and manages the flow of instructions.
- **Memory** –
  - Instruction memory is mostly read-only in the fetch phase.
  - Data memory is required to access the operand and result writing.
  - The memory is accessed over a bus from the CPU.
  - To access memory, the address of the memory location is required in addition to Read/Write of data.
  - The Memory Address Register (MAR) holds the address of the memory location to be accessed.
  - The Memory Data Register (MDR) holds the data. It holds the data read from memory (Data-in) in the case of memory read; holds the data to be written into the memory location in the case of Memory write operations. Thus MDR is a bidirectional register.

- **Registers** – Registers are in physical proximity and internal to the CPU. These are ultra-fast than Memory. Most of the times the operands are brought from memory and kept in registers. Rather these registers are used as a workspace and rough space for workout.
- **Register files** – These are multiport register set enabling faster and parallel access to the register set.
- **Internal Registers** – Instruction Register, Memory Address Register, Memory Data Register. These are not accessible to the programmer.
- **Multiplexers** – Anything is reachable with these. These allow what is to be allowed out based on the selection input.
- **Internal bus** – which connects all these elements.
- **Control unit** – the master which manages the datapath elements.

The MIPS Subset Implementation



Uses a single long clock cycle for every instruction.

Begins execution on one clock edge and completes execution on the next clock edge.

Building a Datapath

- **Fetching Instructions**

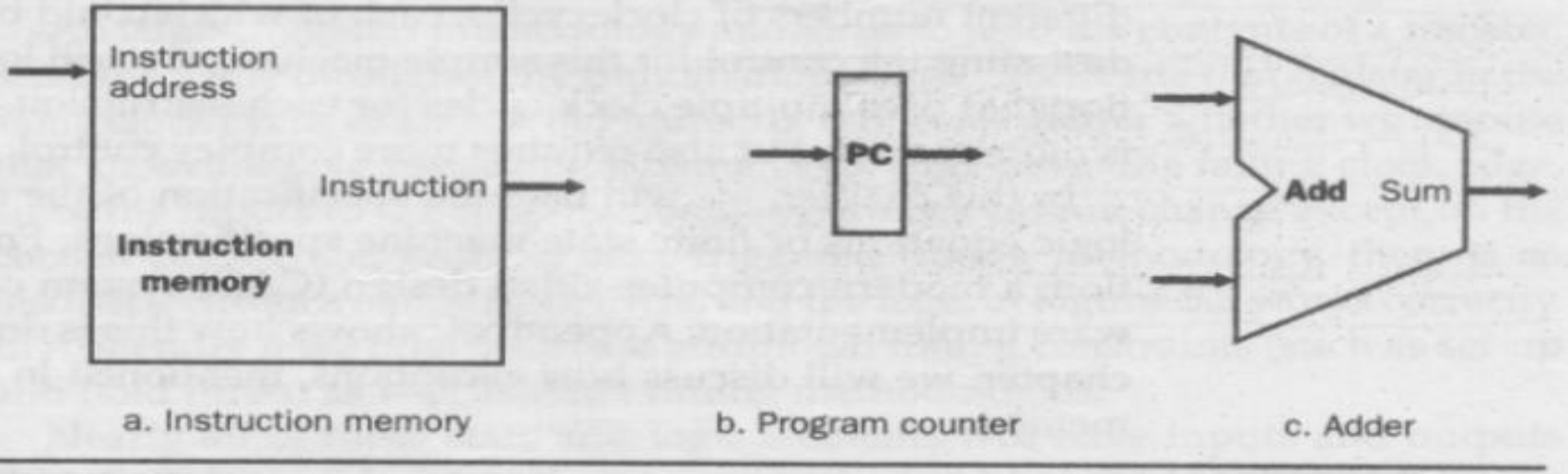
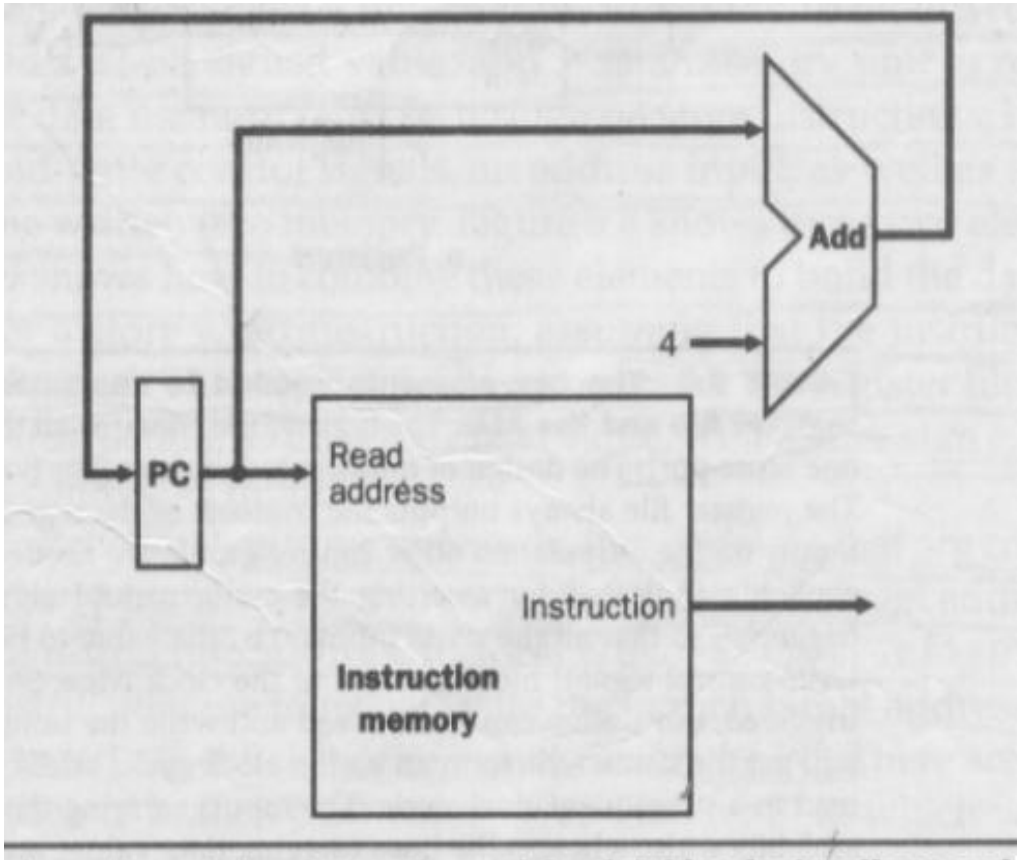


FIGURE 5.4 Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address.

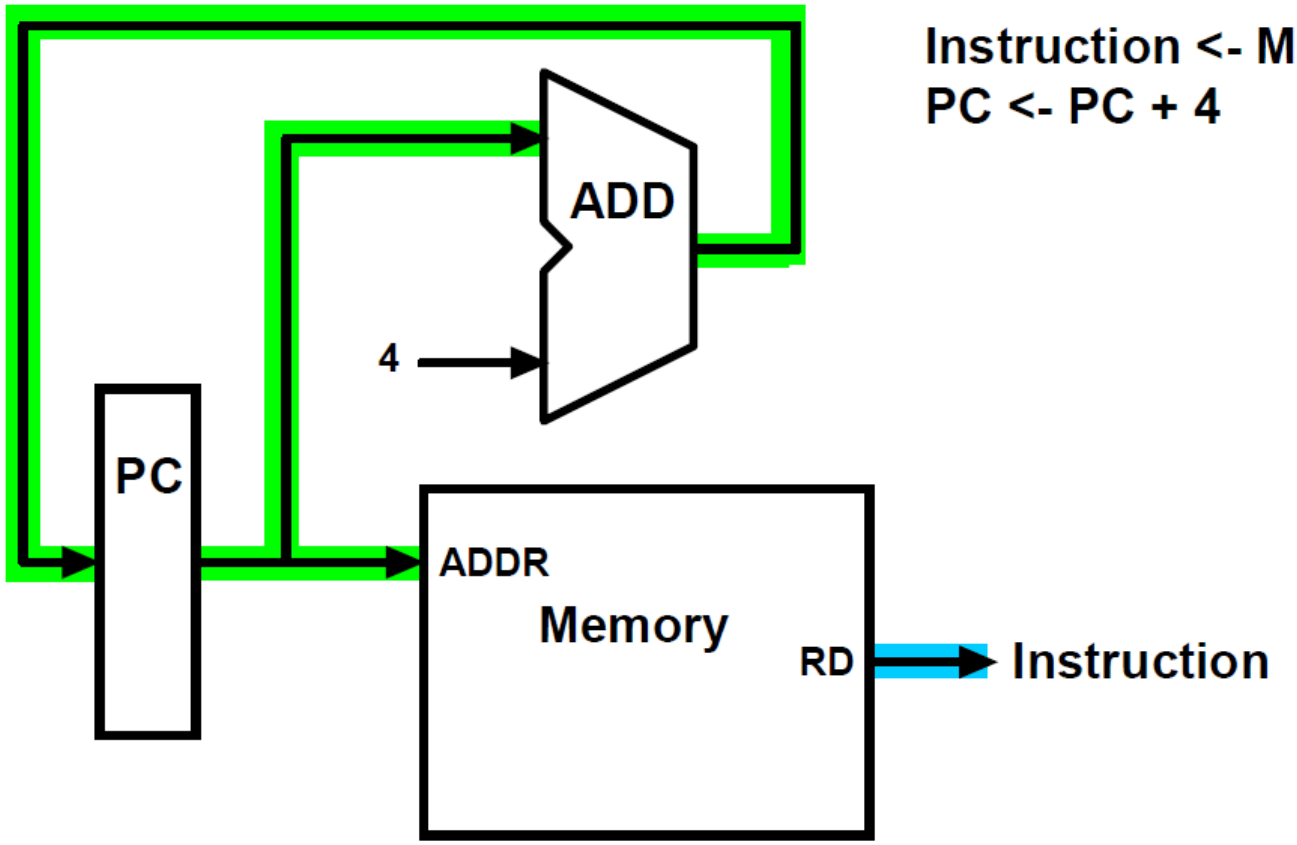
- Instruction memory (IM) and PC are state elements.
- IM is used to hold and supply instruction given an address
- PC keeps the address of the instructions.
- An adder to increment the PC to the address of the next instruction

Building a Datapath (Cont.)

- Fetching Instructions
  - Use the PC to read instruction address
  - Fetch the instruction from memory and increment PC
  - Use field of the instruction to select registers to read
  - Execute depending on the instruction
  - Repeat…



A portion of the datapath used for fetching instructions and incrementing the program counter.


Instruction <- MEM[PC]
PC <- PC + 4
ADD
4
PC
ADDR
Memory
RD
Instruction

Building a Datapath (Cont.)

- Arithmetic-logical Instructions
- add $t1, $t2, $t3

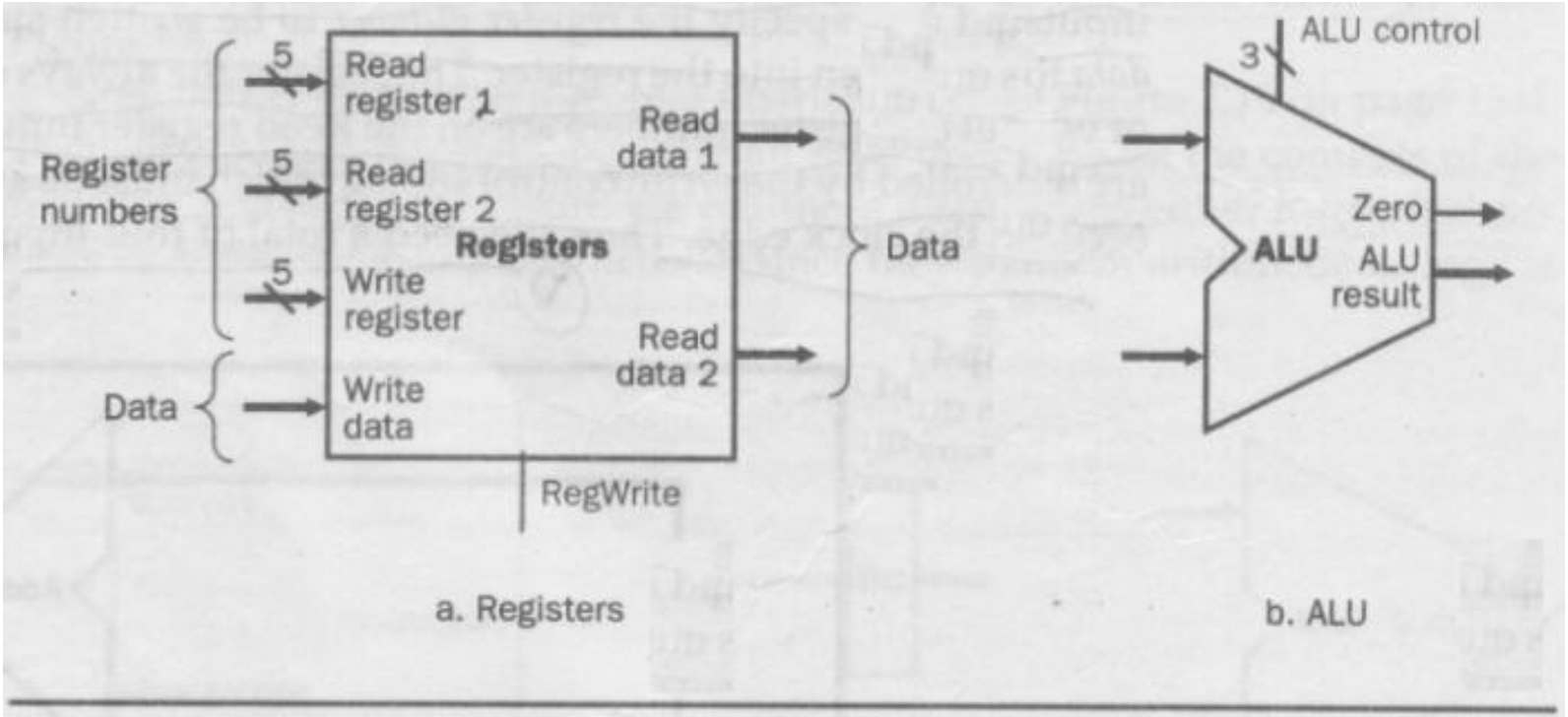


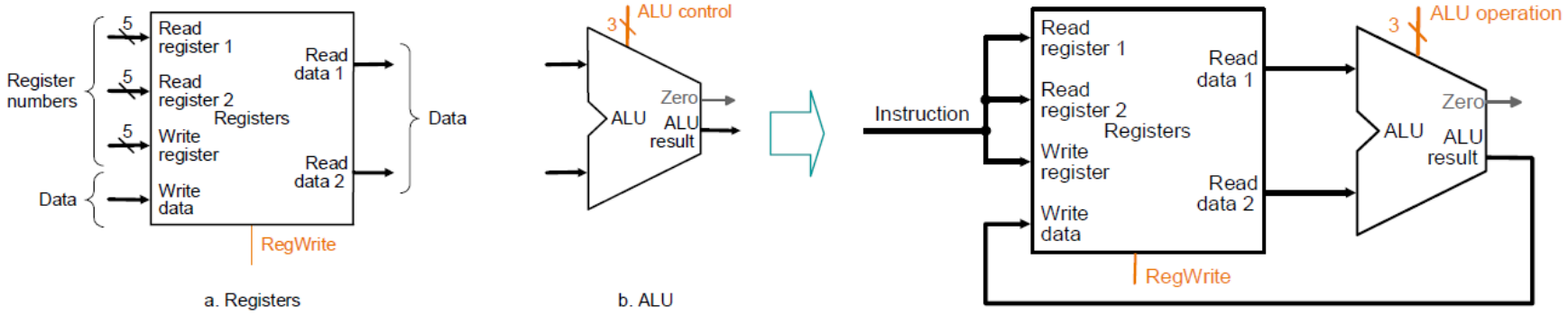
FIGURE 5.6 The two elements needed to implement R-format ALU operations are the register file and the ALU.

Building a Datapath (Cont.)

- Arithmetic logical operations:

- The register file contains all the registers and has two read ports and one write ports.
- Register file always outputs the contents of the registers corresponding to the read registers.
- Write is controlled by the write control signal
- We need total four inputs (three for register number and one for data)
- ALU is controlled with the ALU operation signal which is 32 bit wide.

Datapath: R-Type Instruction

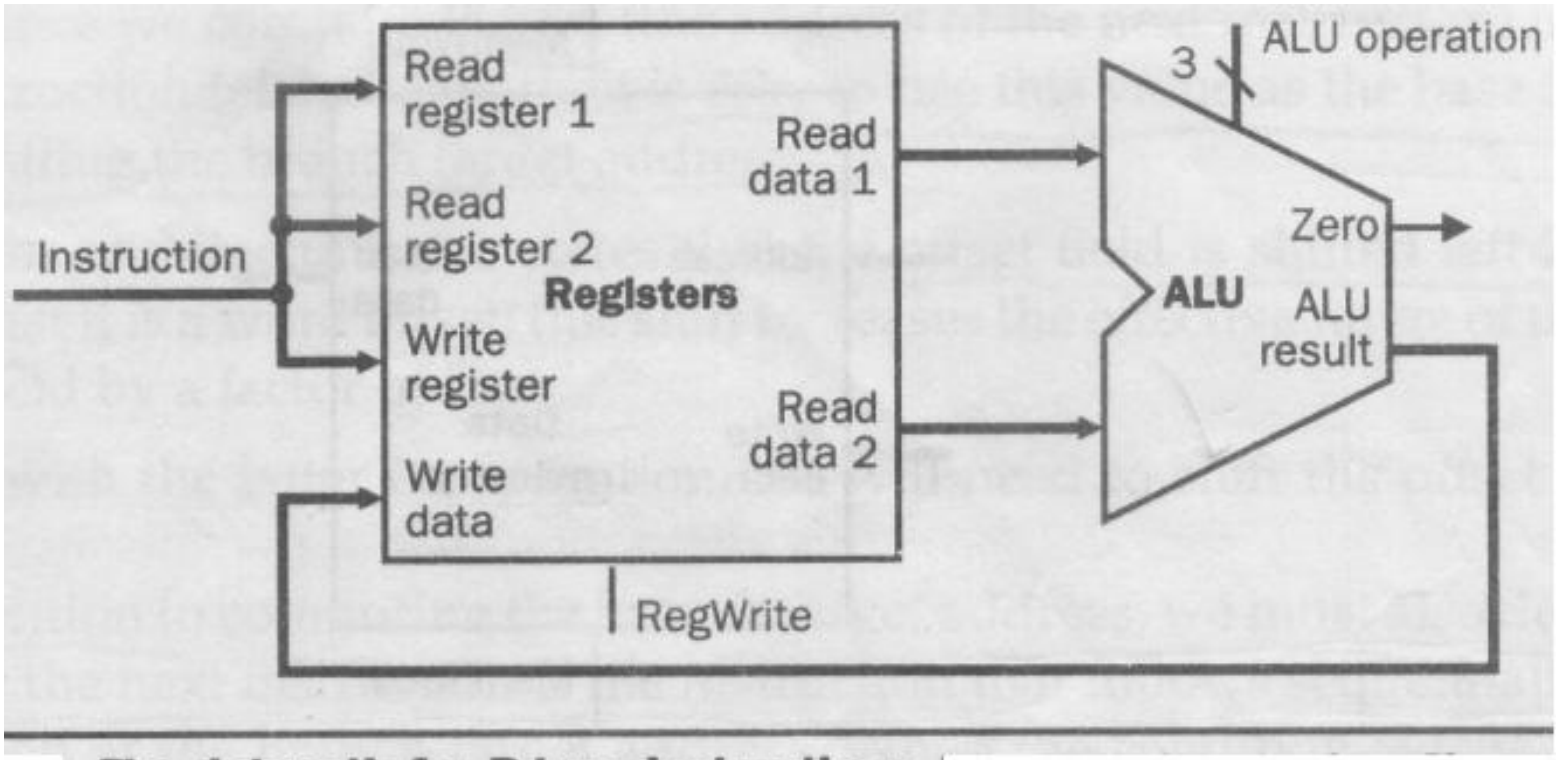


Two elements used to implement R-type instructions

Datapath

Building a Datapath (Cont.)

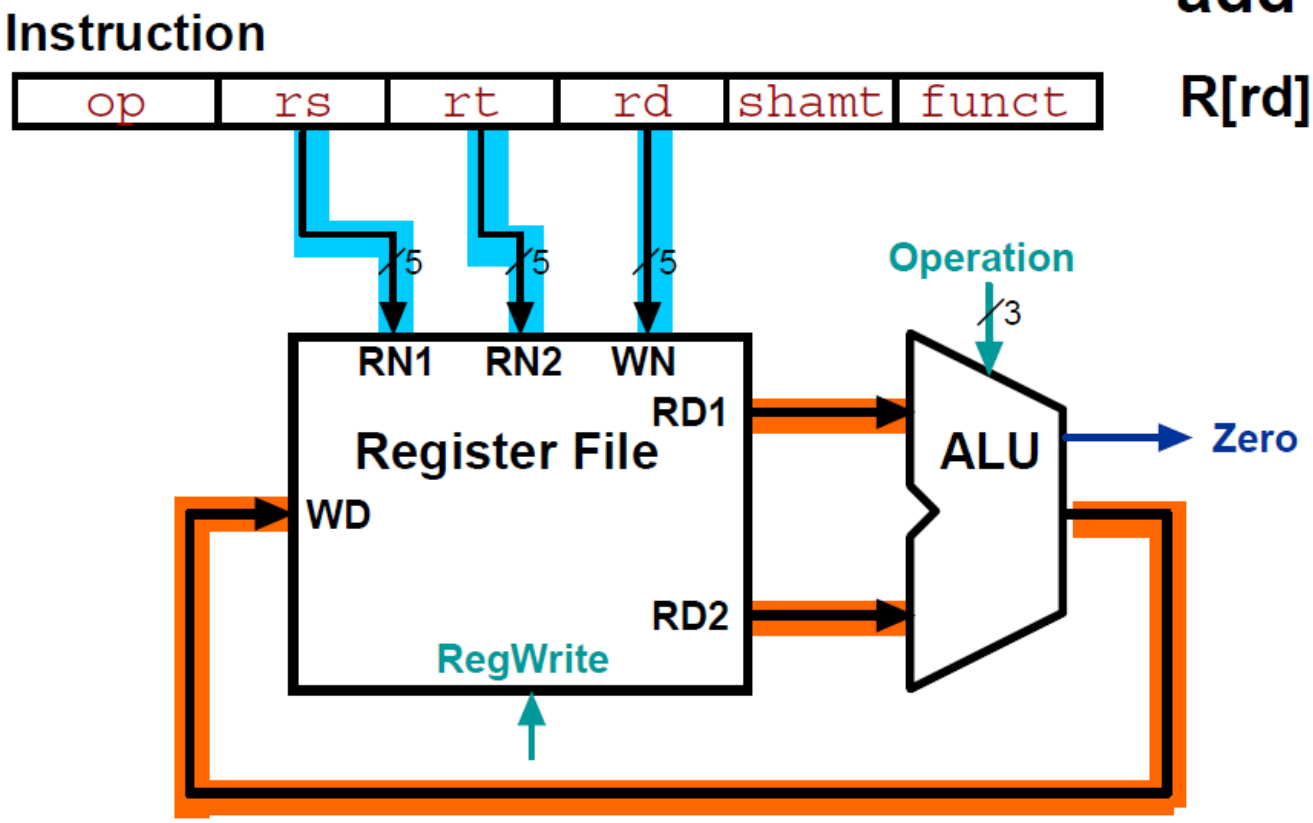
- Arithmetic-logical Instructions
- add $t1, $t2, $t3



The datapath for R-type instructions.

add rd, rs, rt

R[rd] <- R[rs] + R[rt];



Building a Datapath (Cont.)

- Load word / Store word
- lw $s1, 100 ($s2)

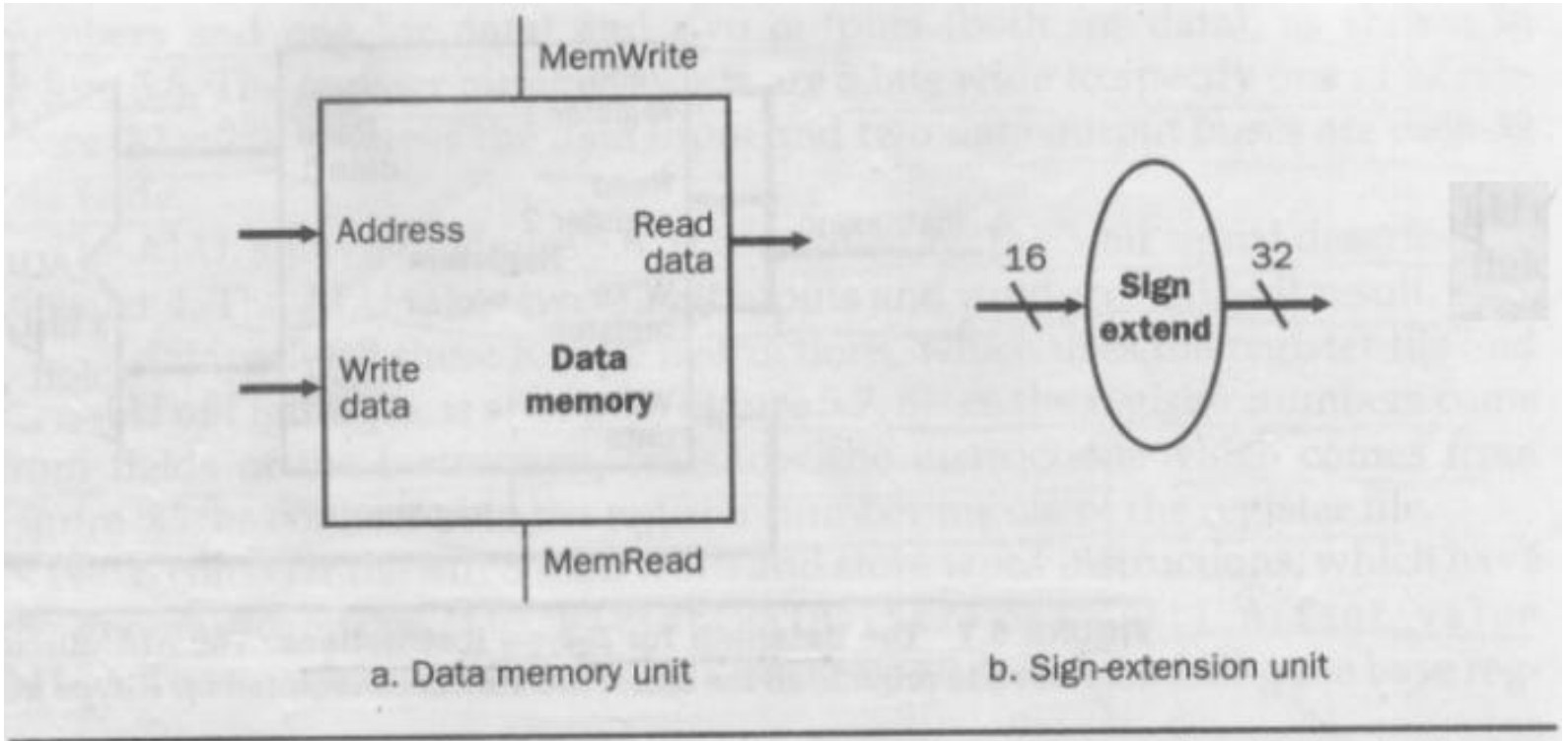


FIGURE 5.8 The two units needed to implement loads and stores, in addition to the register file and ALU of Figure 5.6, are the data memory unit and the sign extension unit.

Building a Datapath (Cont.)

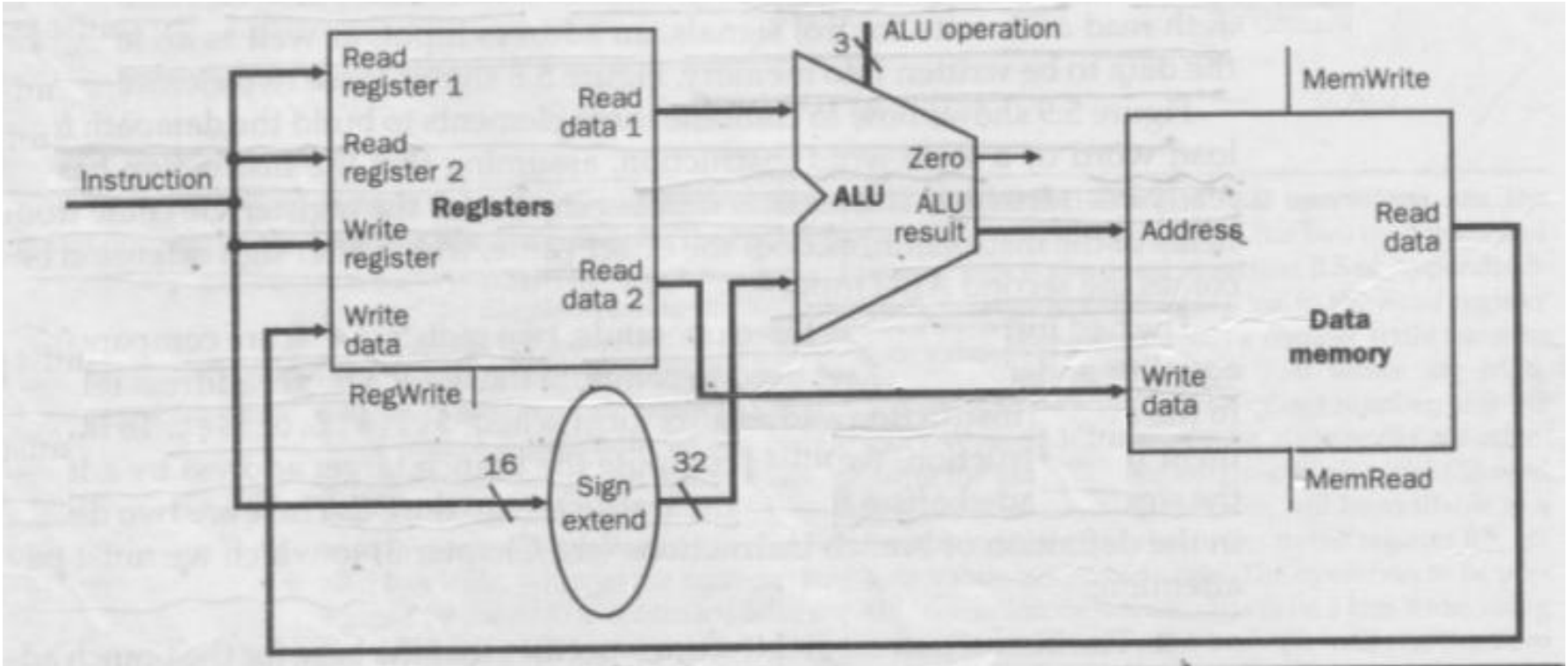
- Load/store word:
- The memory unit in a state element with inputs for the address and write data, having a single output for the read result

- Separate read and write control signal

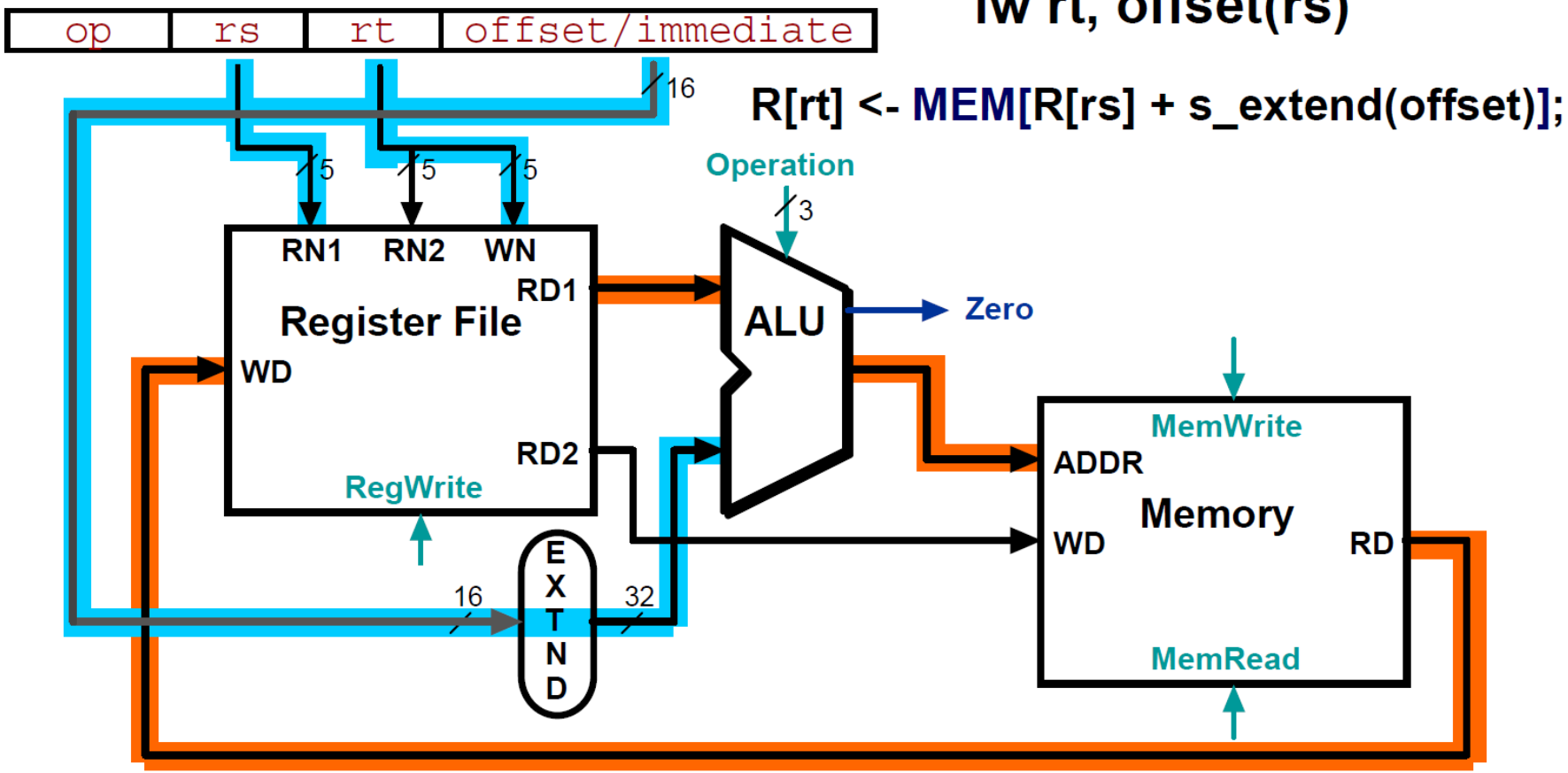
- The sign-extension unit has a 16-bit input that is sign extended into a 32-bit result appearing on the output

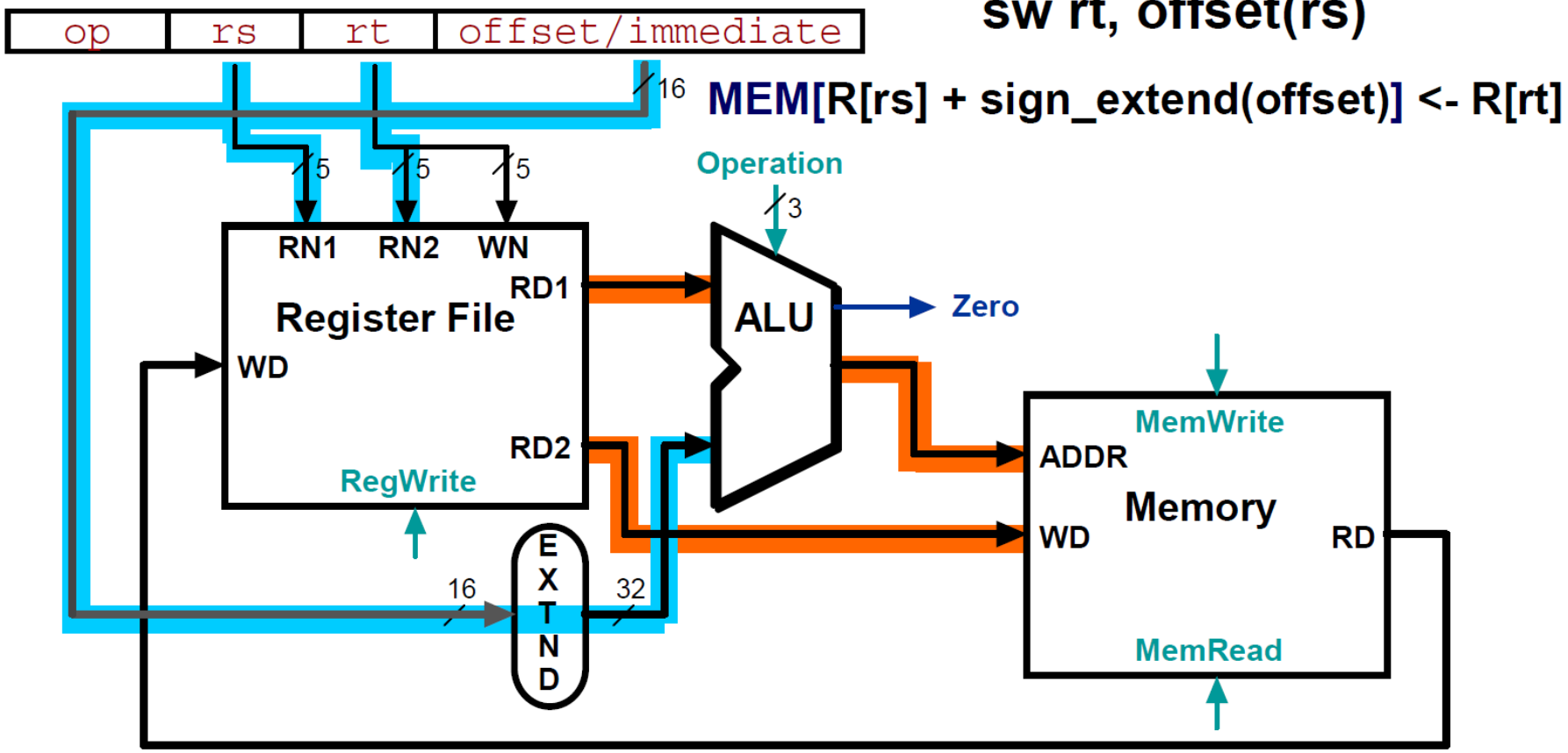
Building a Datapath (Cont.)

- Load word / Store word
- lw $s1, 100 ($s2)



The datapath for a load or store does a register access, followed by a memory address calculation, then a read or write from memory, and a write into the register file if the instruction is a load.


op
rs
rt
offset/immediate
lw rt, offset(rs)
R[rt] <- MEM[R[rs] + s_extend(offset)];
16
5
5
5
Operation
3
RN1
RN2
WN
RD1
Register File
WD
RD2
RegWrite
ALU
Zero
MemWrite
ADDR
Memory
WD
RD
MemRead
16
EXTND
32


op
rs
rt
offset/immediate
sw rt, offset(rs)
MEM[R[rs] + sign_extend(offset)] <- R[rt]
16
5
5
5
RN1
RN2
WN
RD1
Register File
WD
RD2
RegWrite
Operation
3
ALU
Zero
MemWrite
ADDR
Memory
WD
RD
MemRead
16
EXTND
32

Building a Datapath (Cont.)

- Branch Instructions
- beq $t1, $t2, offset

6 bits	5 bits	5 bits	16 bits
op	rs	rt	address / immediate

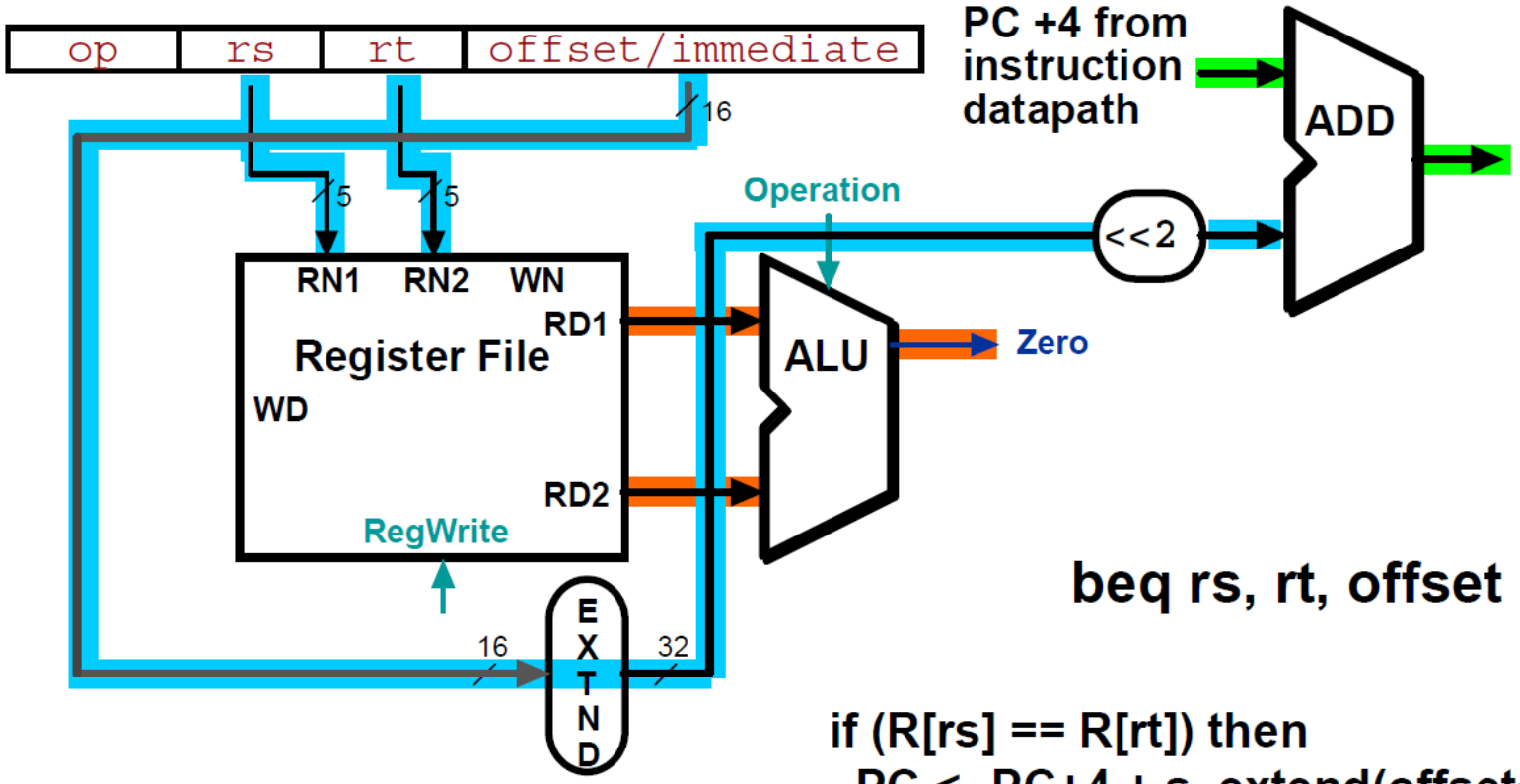
- Details of branch instructions:
  - It has three operands , two registers that are compared for equality and a 16 bit offset used to compute the branch target address.
  - Branch target address = sign extended offset field + PC
  - To compute the branch target address the branch datapath includes- 1. a sign extension and 2. adder
  - The offset field is shifted left 2 bits.

Building a Datapath (Cont.)

- Jump
- j 2500 # go to 10000

6 bits	5 bits	5 bits	16 bits
op	rs	rt	address / immediate

- Replace the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits.


op
rs
rt
offset/immediate
16
5
5
PC +4 from
instruction
datapath
ADD
Operation
<<2
RN1
RN2
WN
RD1
Register File
WD
RD2
RegWrite
ALU
Zero
16
EXTND
32
beq rs, rt, offset
if (R[rs] == R[rt]) then
PC <- PC+4 + s_extend(offset<<2)

Building a Datapath (Cont.)

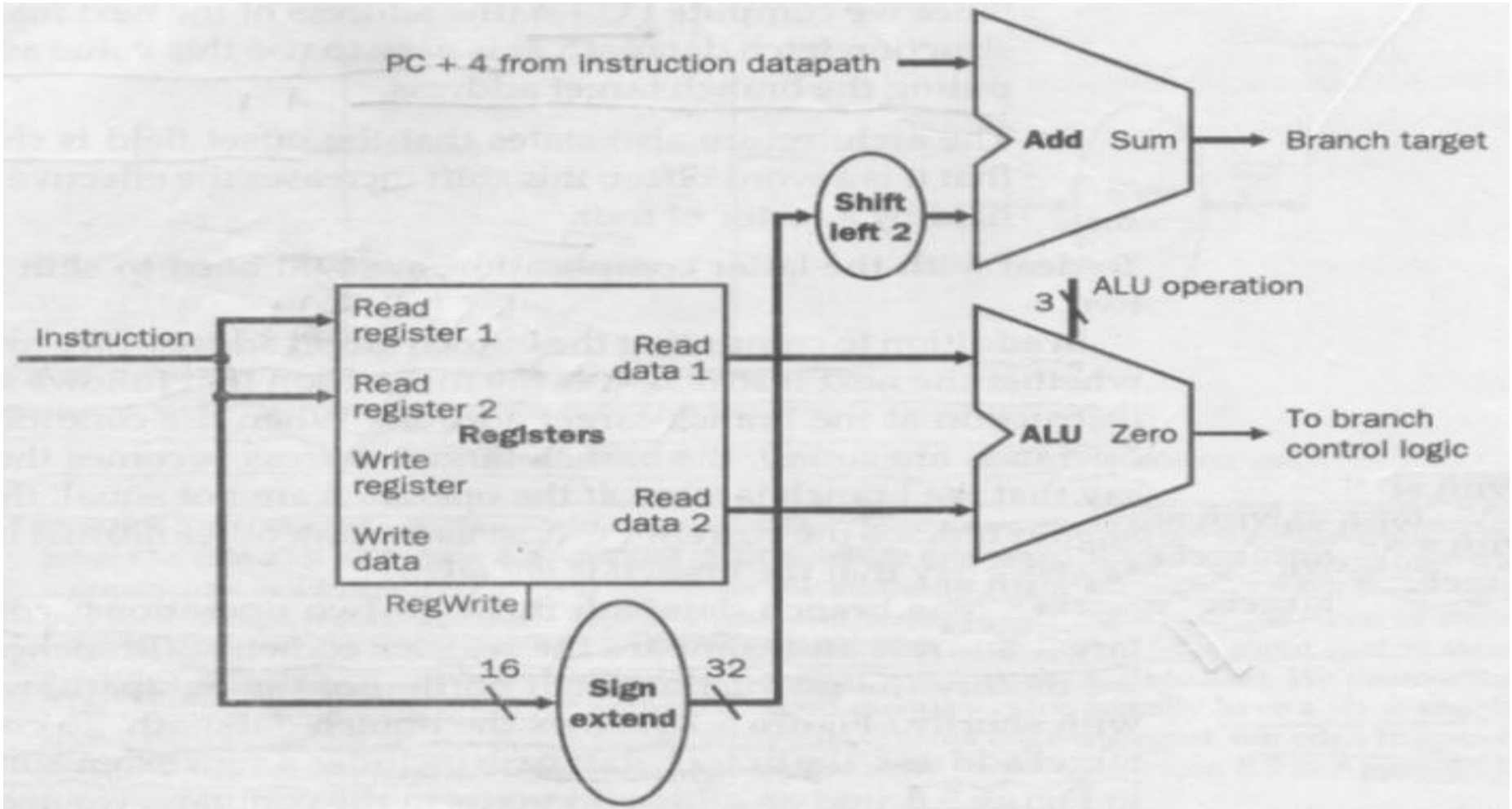


FIGURE 5.10 The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.

A Simple Implementation Scheme

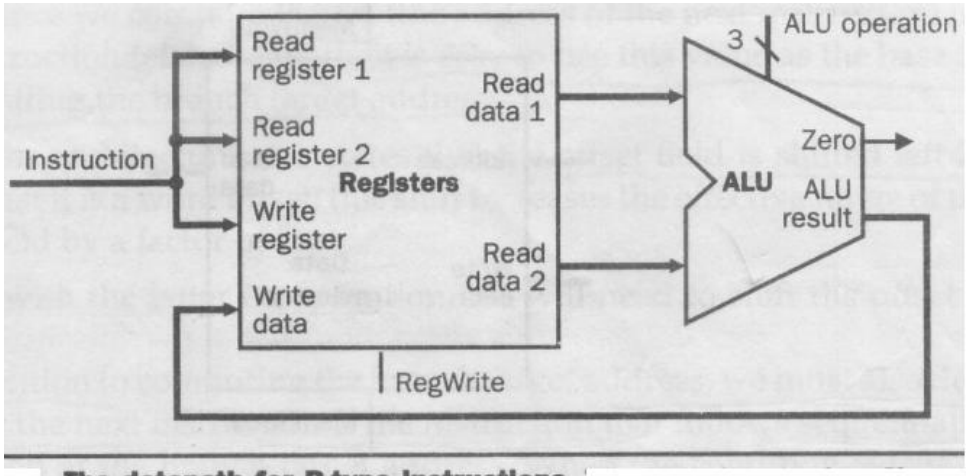
- A datapath is a collection of functional units, such as arithmetic logic units or multipliers, that perform at a processing operations, registers, and buses. Along with the control unit it composes the central processing unit (CPU).

A Simple Implementation Scheme

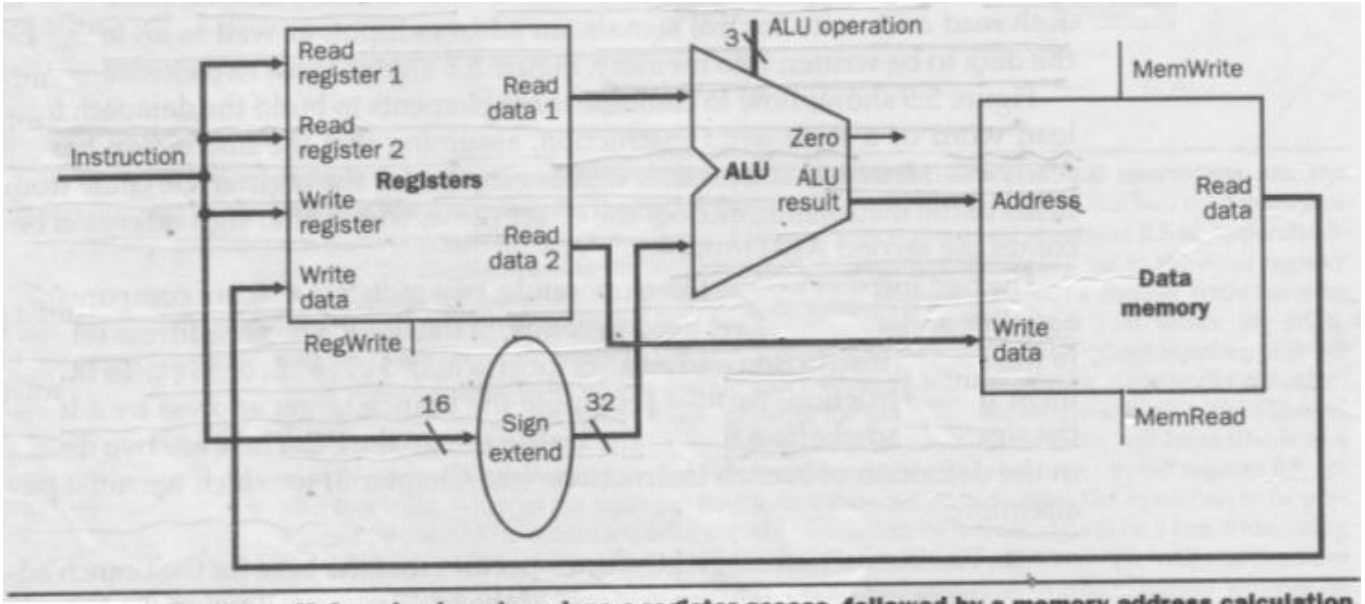
- Creating a Single Datapath
  - Single cycle implementation
  - No datapath resource can be used more than once per instruction.
  - So any element needed more than once must be duplicated (instruction memory and data memory are separate)
  - To share a datapath element we need multiplexer / data selector.

A Simple Implementation Scheme

- Combining Datapath:



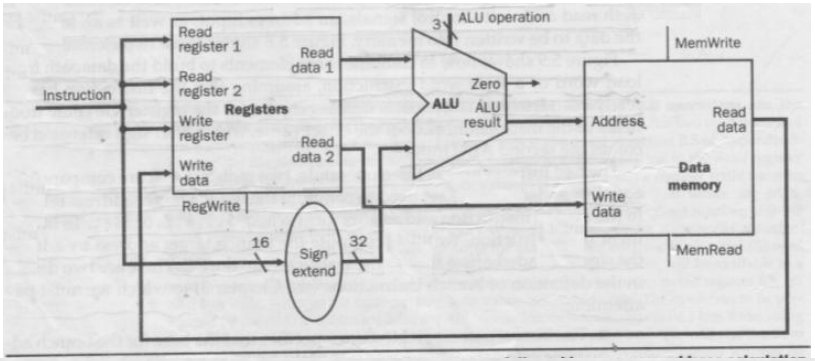
The datapath for R-type instructions.

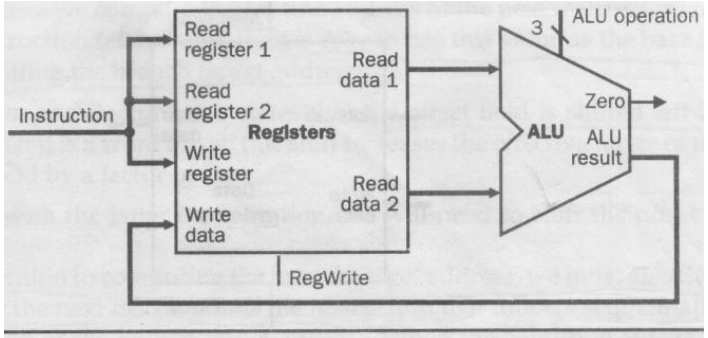


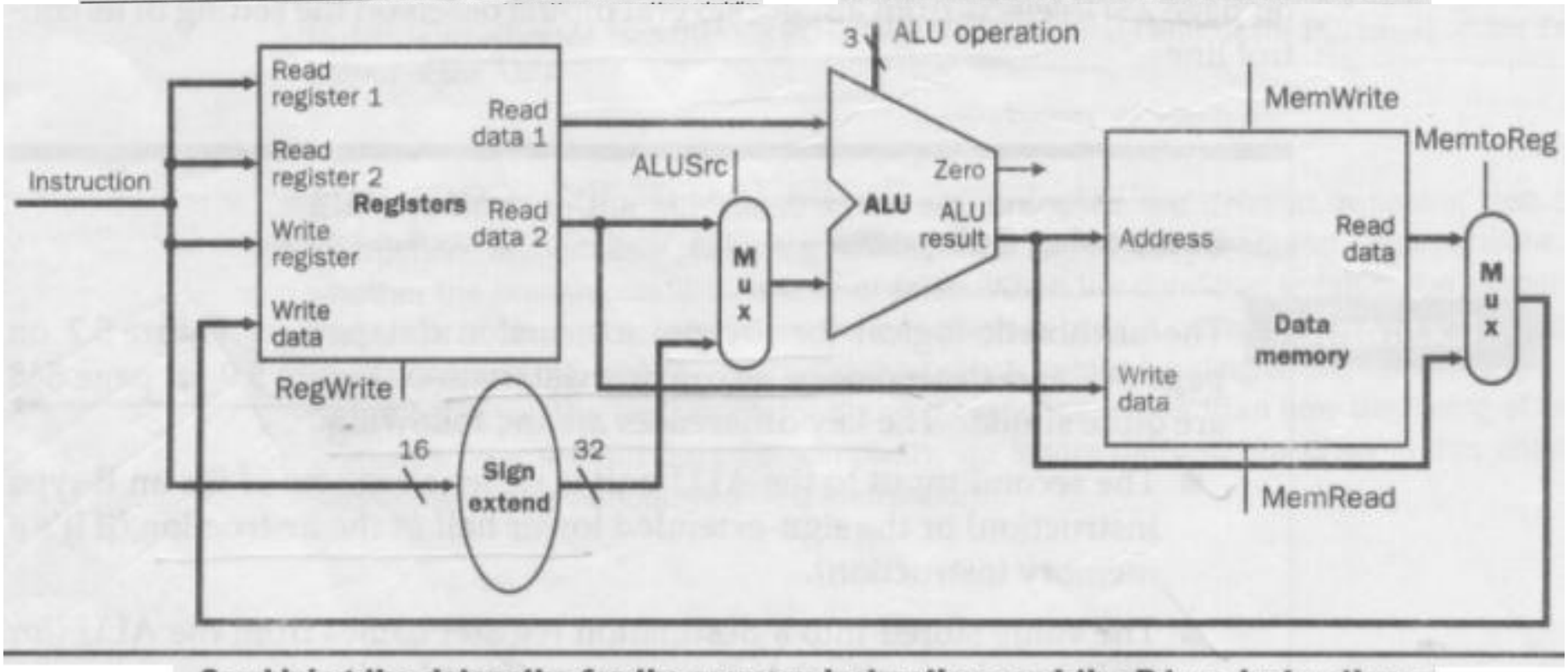
The datapath for a load or store does a register access, followed by a memory address calculation, then a read or write from memory, and a write into the register file if the instruction is a load.

A Simple Implementation Scheme

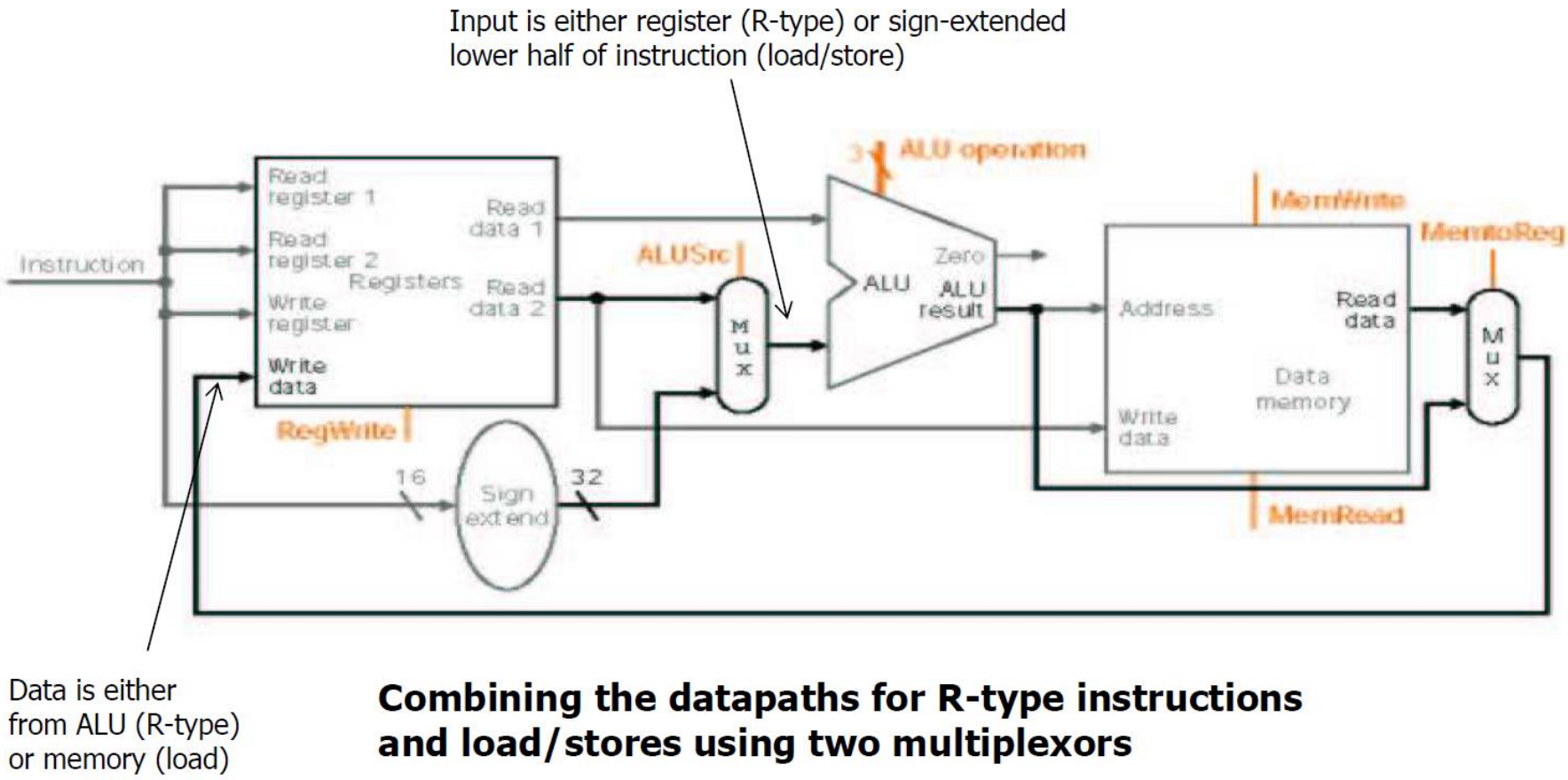
Datapath for memory instructions and R-type instructions





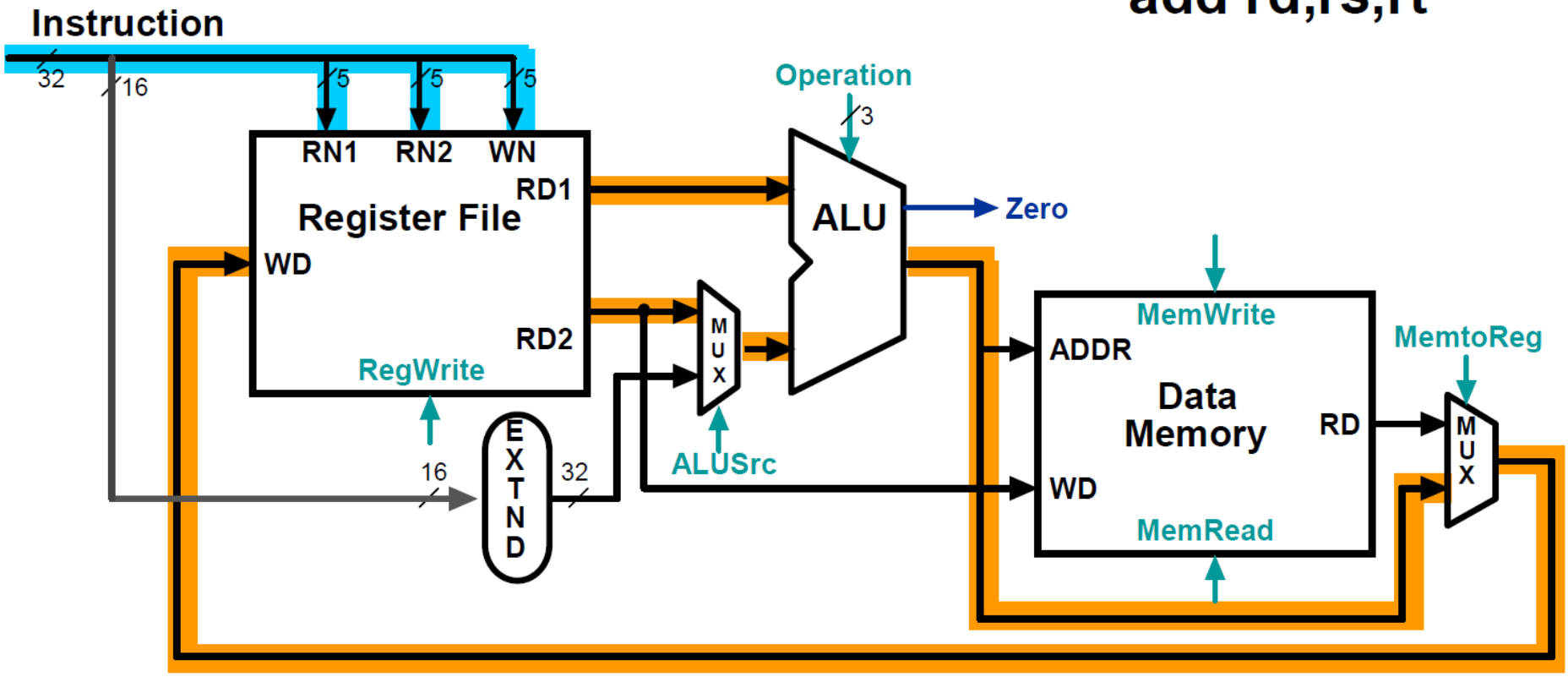


Combining the datapaths for the memory instructions and the R-type instructions.

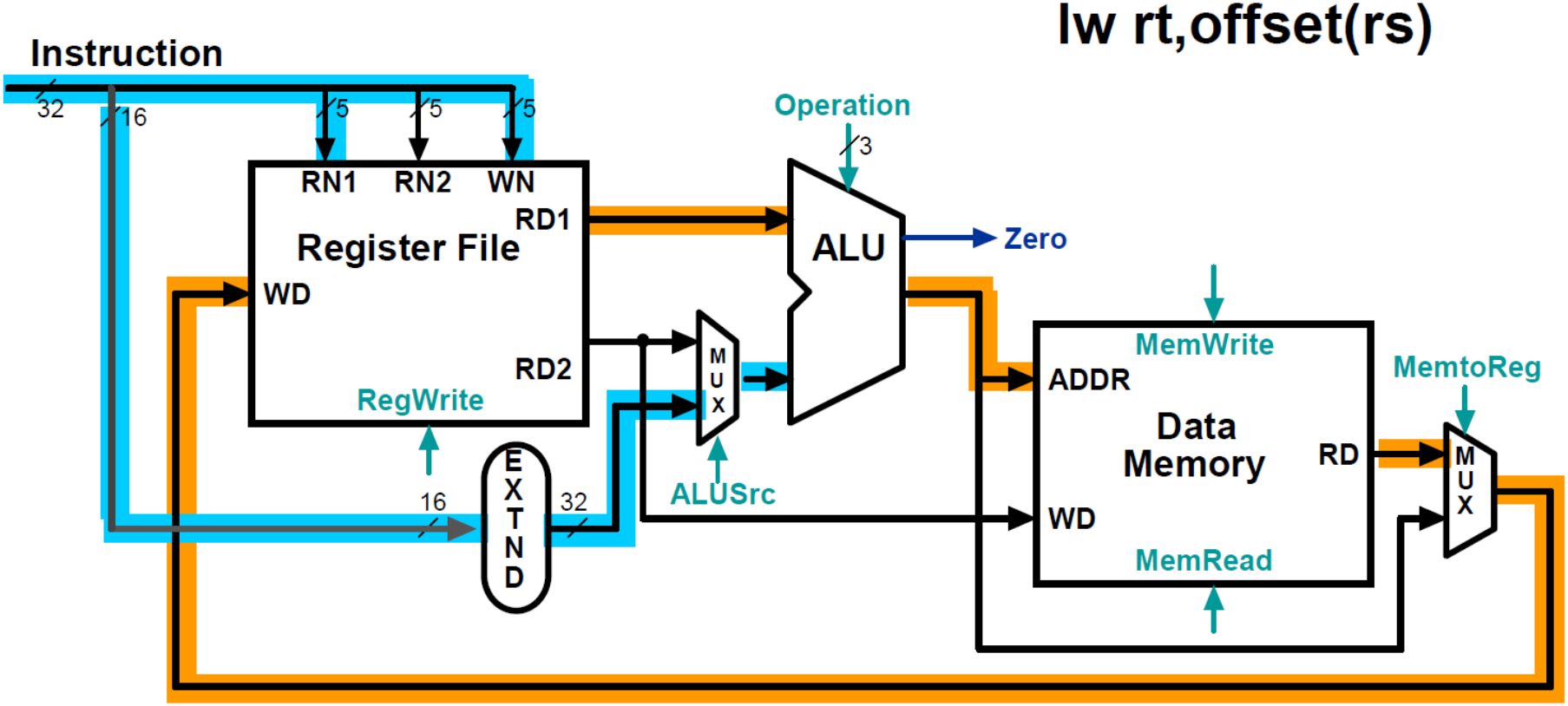


Combining the datapaths for R-type instructions and load/stores using two multiplexors

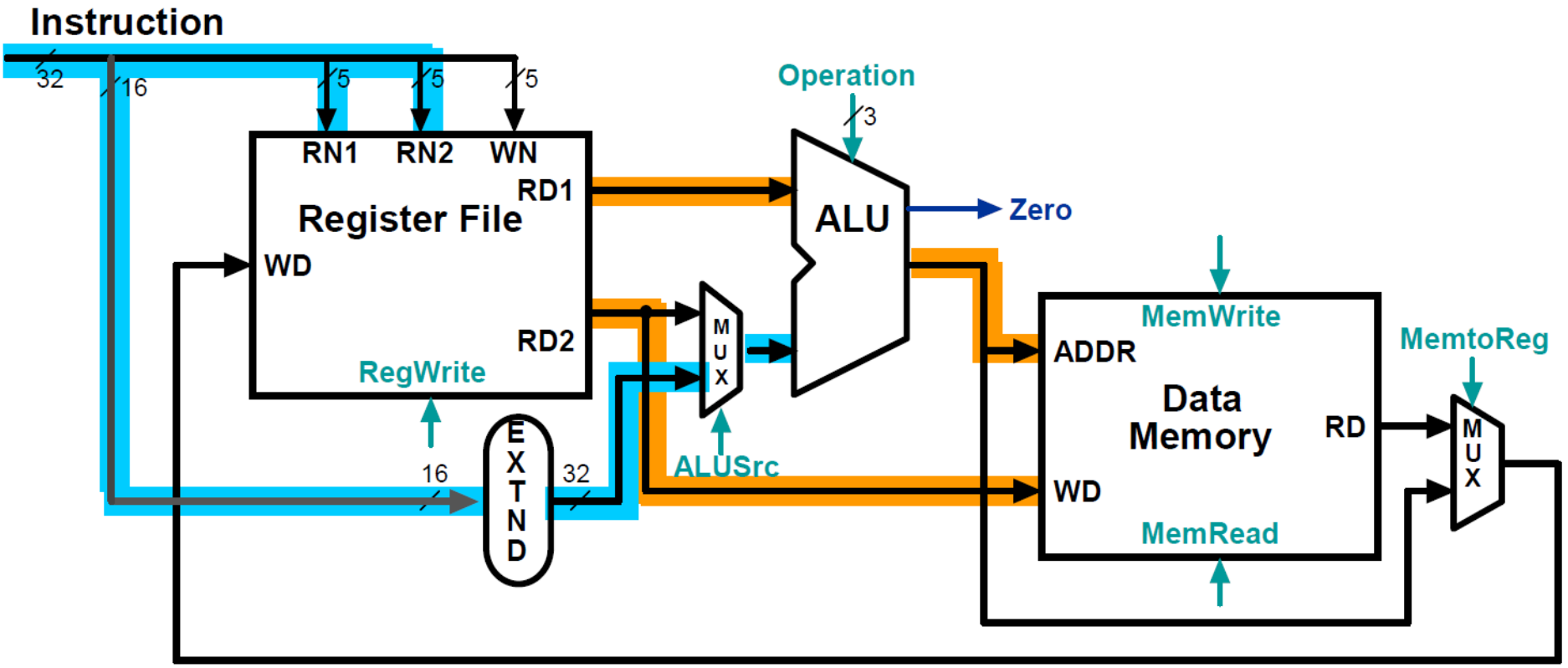
add rd,rs,rt


Instruction
32
16
5
5
5
RN1
RN2
WN
RD1
Register File
WD
RD2
RegWrite
16
EXTND
32
M
U
X
ALUSrc
Operation
3
ALU
Zero
MemWrite
ADDR
Data
Memory
RD
WD
MemRead
MemtoReg
M
U
X

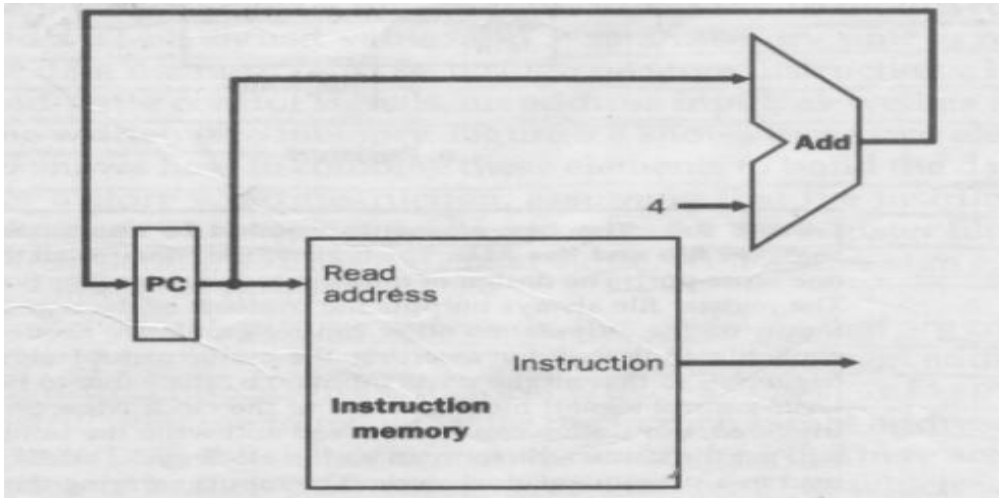
lw rt,offset(rs)



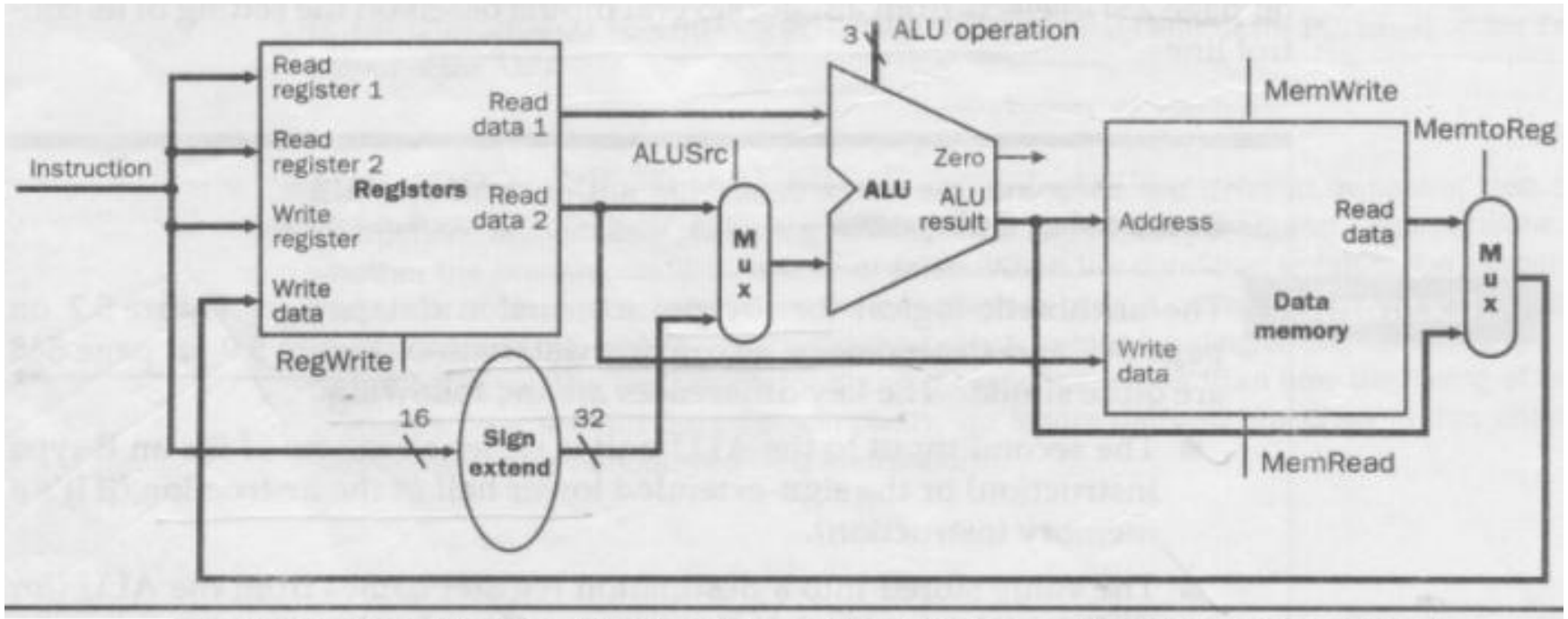
sw rt,offset(rs)



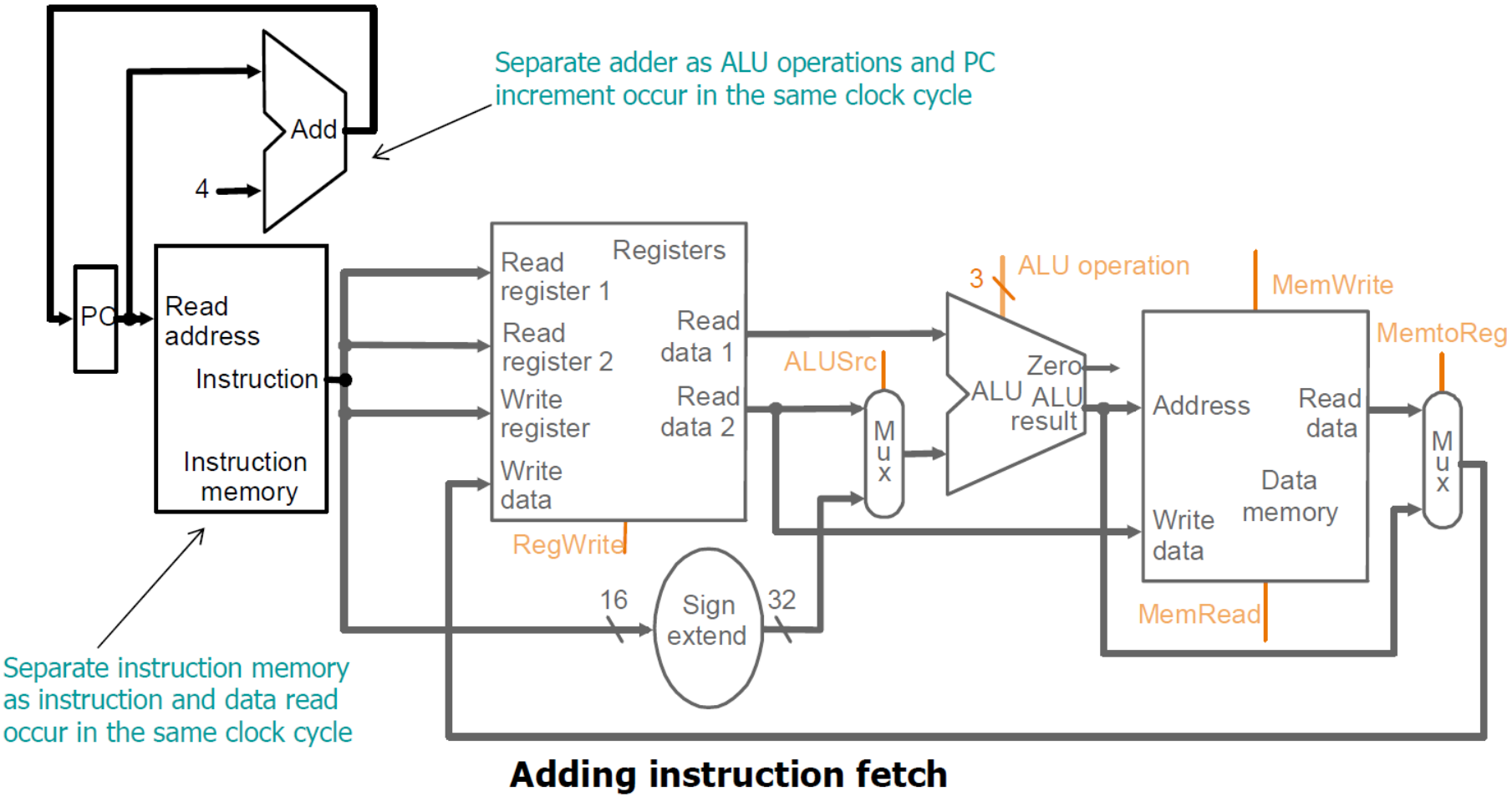
A Simple Implementation Scheme



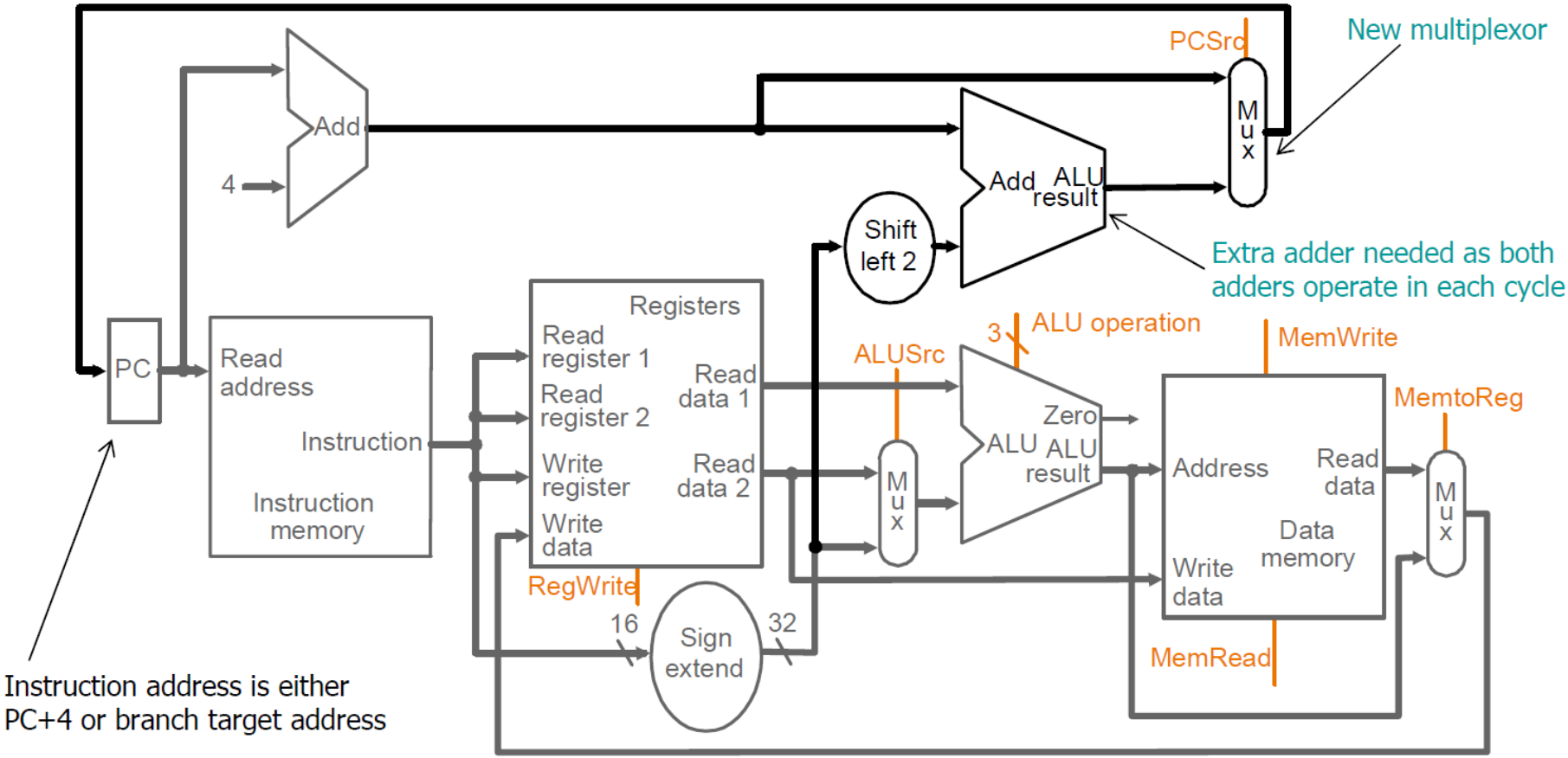
A portion of the datapath used for fetching instructions and incrementing the program counter.



Combining the datapaths for the memory instructions and the R-type instructions.



Adding instruction fetch



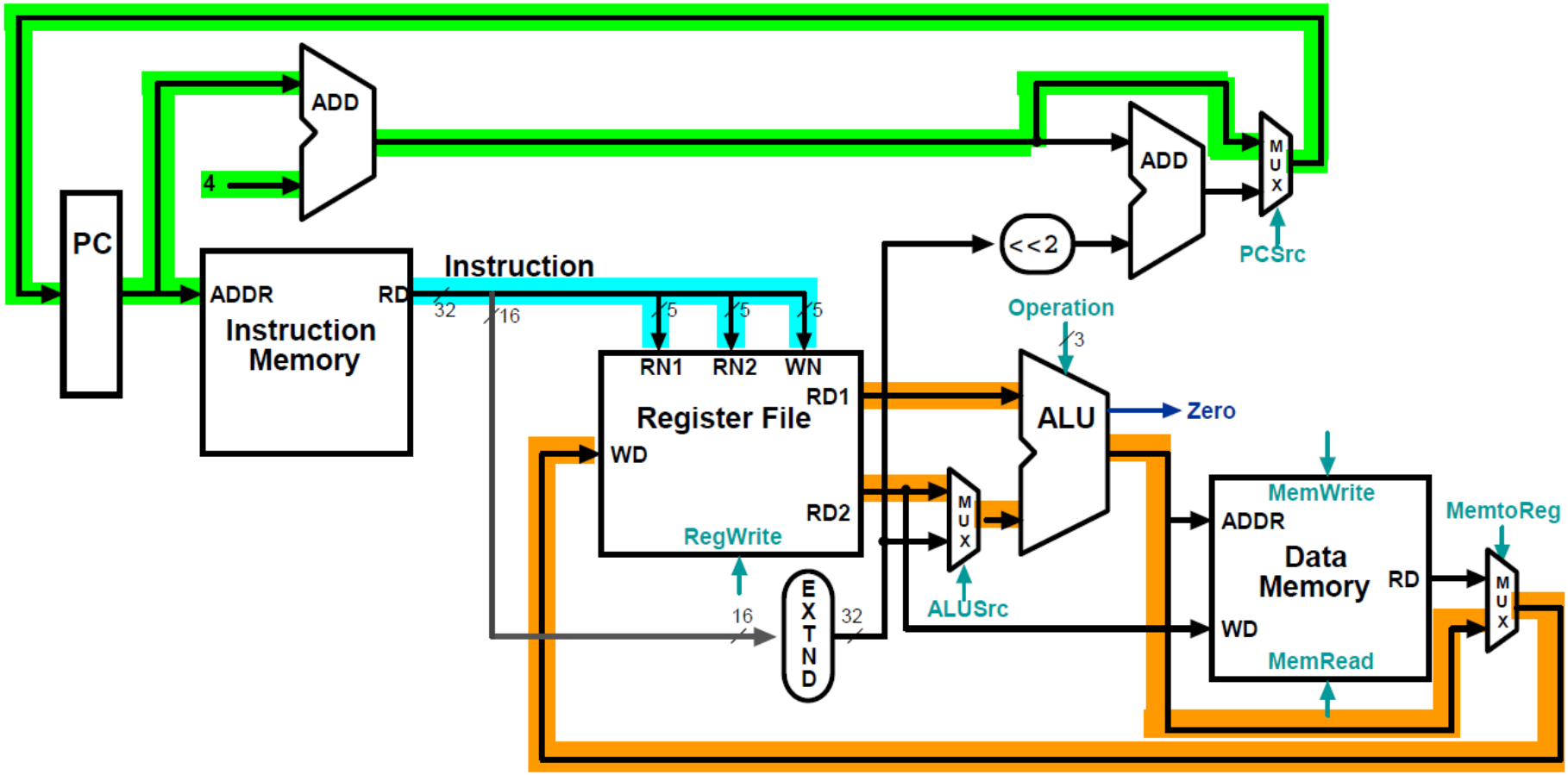
Adding branch capability and another multiplexor

Important note: in a single-cycle implementation data cannot be stored during an instruction – it only moves through combinational logic

Question: is the MemRead signal really needed?! Think of RegWrite…!

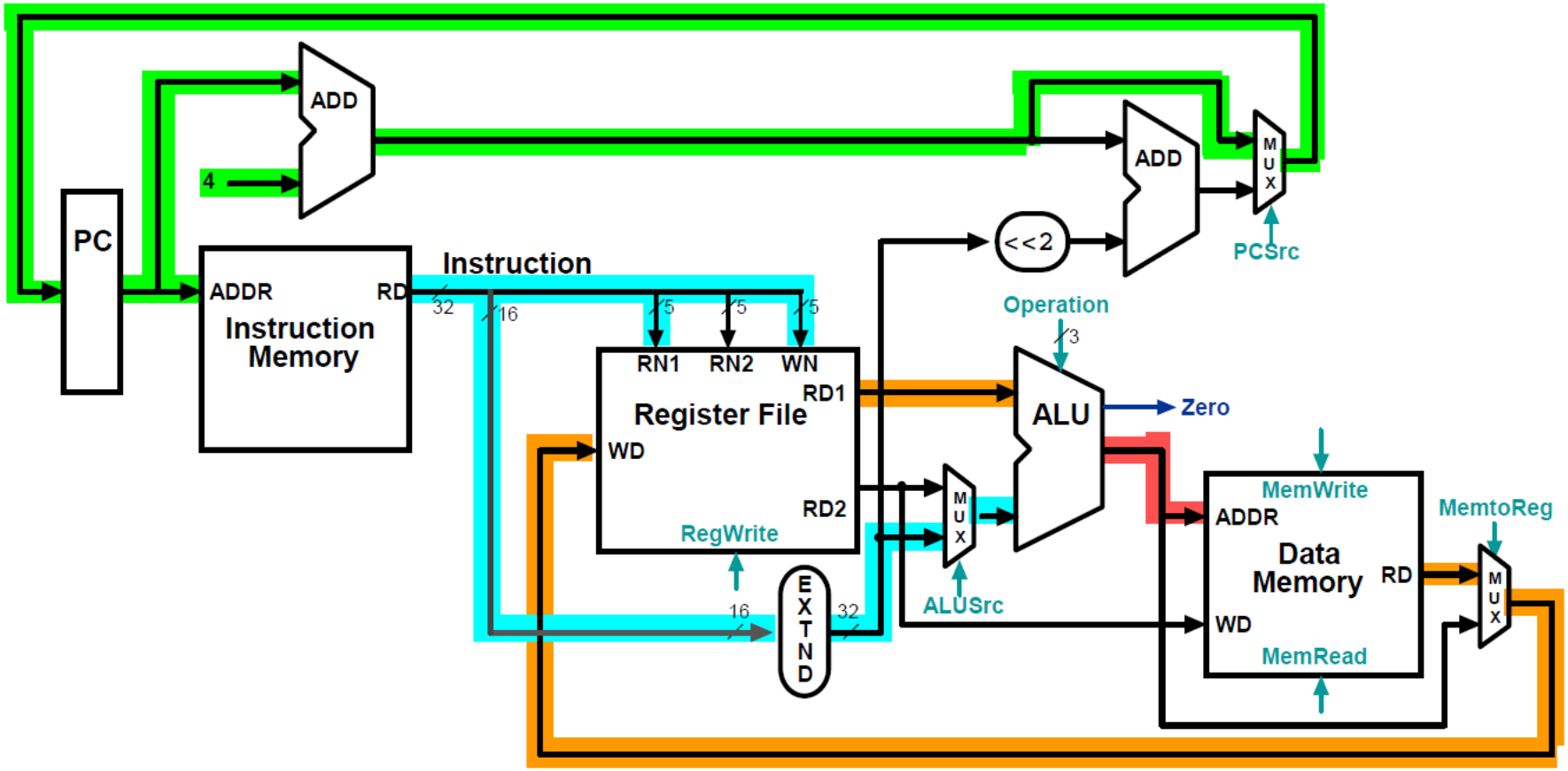
Datapath Executing `add`

add rd, rs, rt



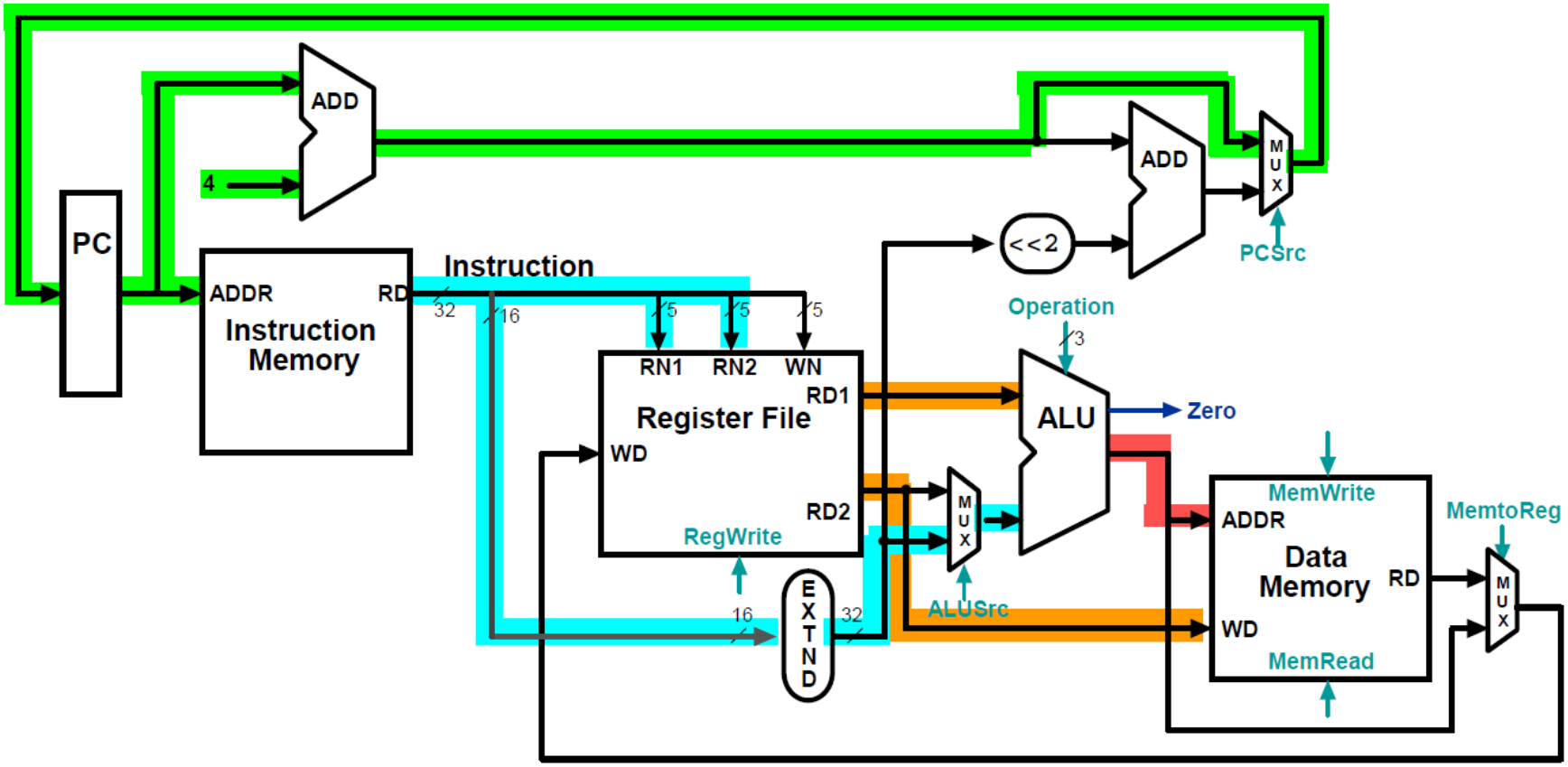
Datapath Executing `lw`

lw rt,offset(rs)



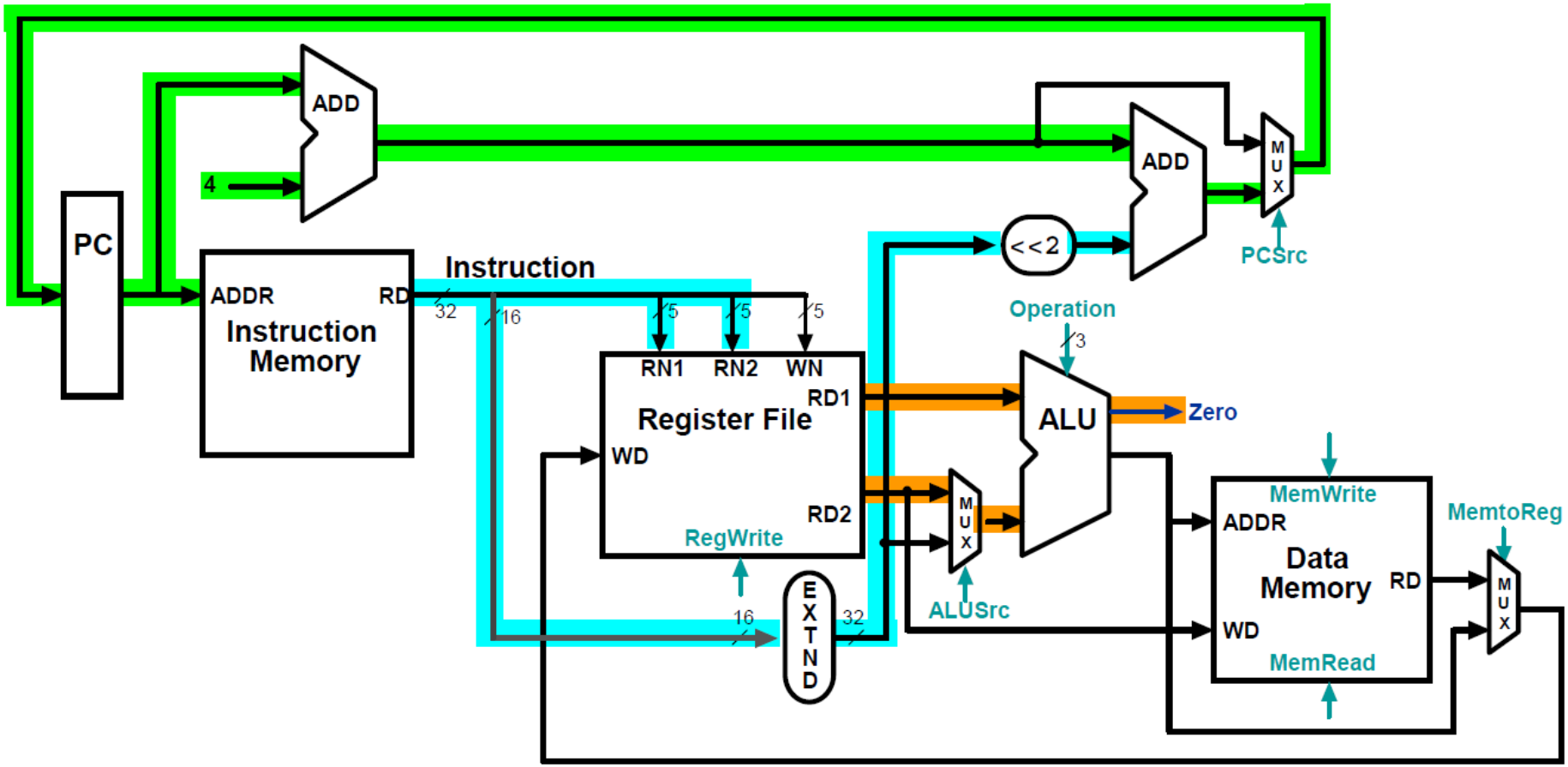
Datapath Executing `sw`

sw rt,offset(rs)



Datapath Executing `beq`

beq r1,r2,offset



A Simple Implementation Scheme (Cont.)

- **Designing the Main Control Unit**
- R-type

31-26	25-21	20-16	15-11	10-6	5-0
op	rs	rt	rd	shamt	funct

- lw, sw

31-26	25-21	20-16	15-0
35 or 43	rs	rt	address

- Branch

31-26	25-21	20-16	15-0
4	rs	rt	address

A Simple Implementation Scheme (Cont.)

- Designing the Main Control Unit
- Observations:
  - op field always contained in bits 31-26.
  - Two registers to be read are always specified at positions 25-21 and 20-16. (R-type, beq, sw)
  - Base register for lw and sw is always in 25-21
  - 16-bit offset (beq, lw, sw) are always in 15-0
  - Destination register is in 20-16 (lw) or 15-11 (R-type).

- Designing the Main Control Unit
- The different positions for the two destination registers implies a selector (i.e., a mux) to locate the appropriate field for each type of instruction.

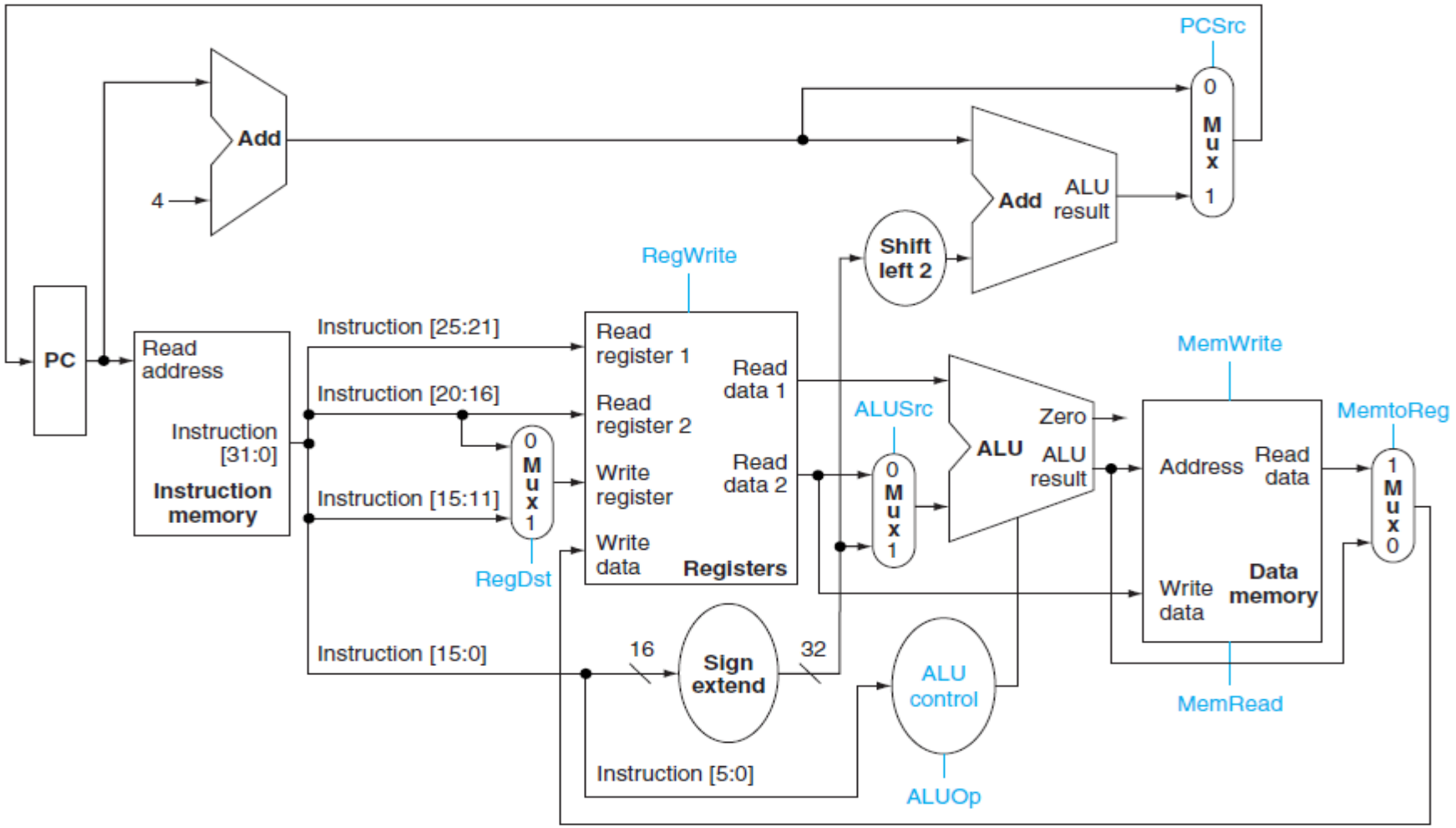


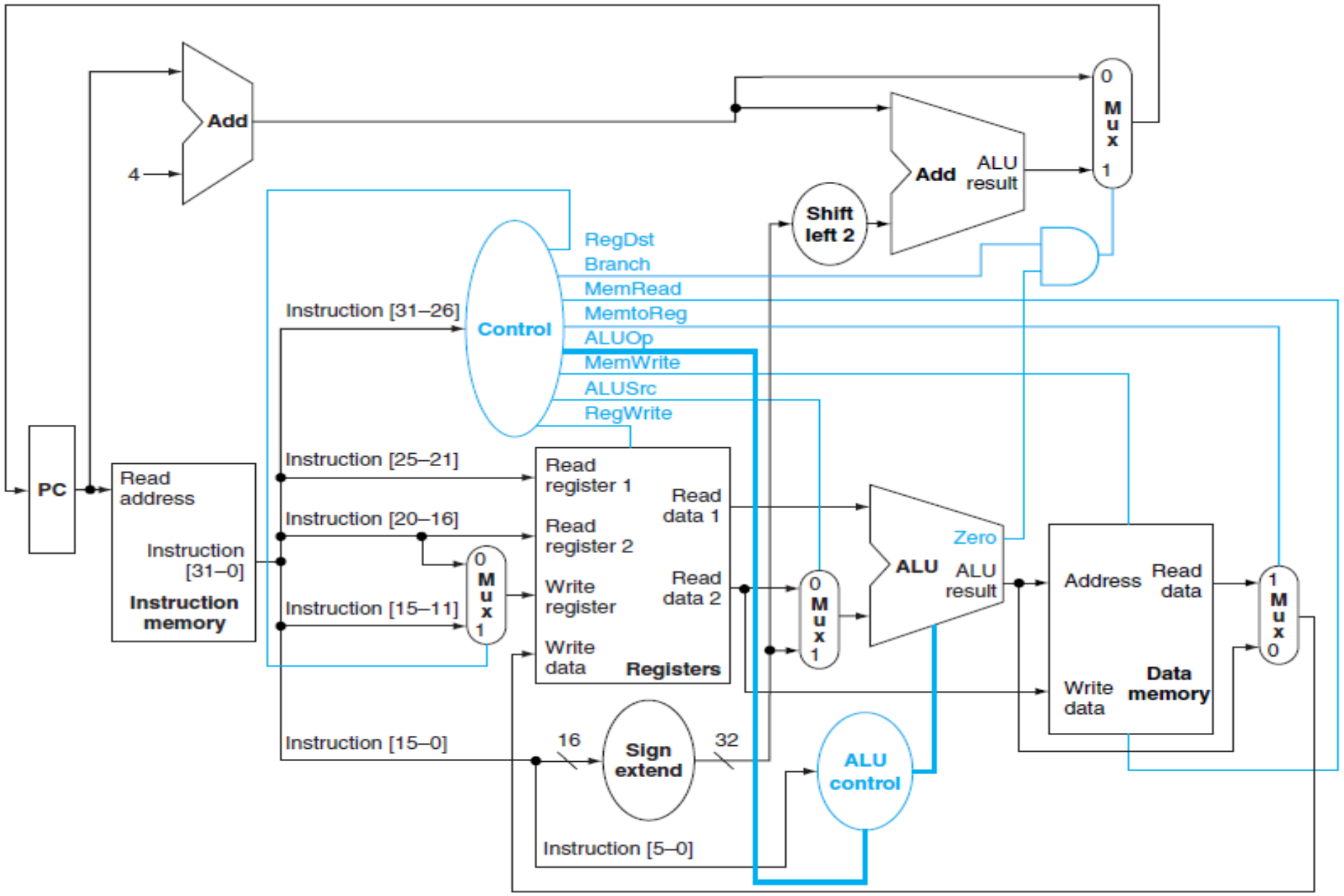
FIGURE 5.15 The datapath of Figure 5.12 with all necessary multiplexors and all control lines identified. The control lines are

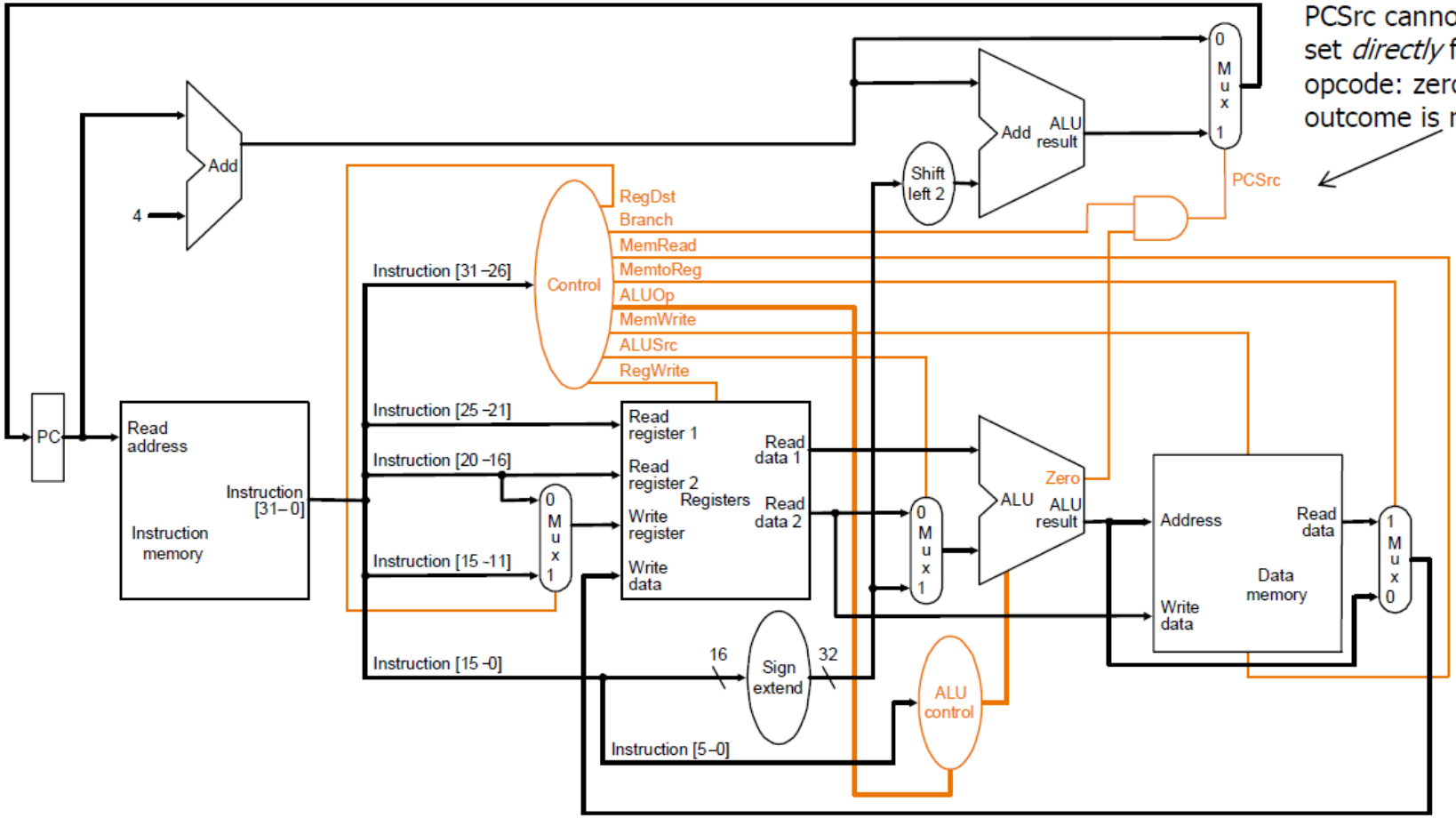
A Simple Implementation Scheme (Cont.)

- The effect of each of the seven control lines

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

- The simple datapath with the control unit



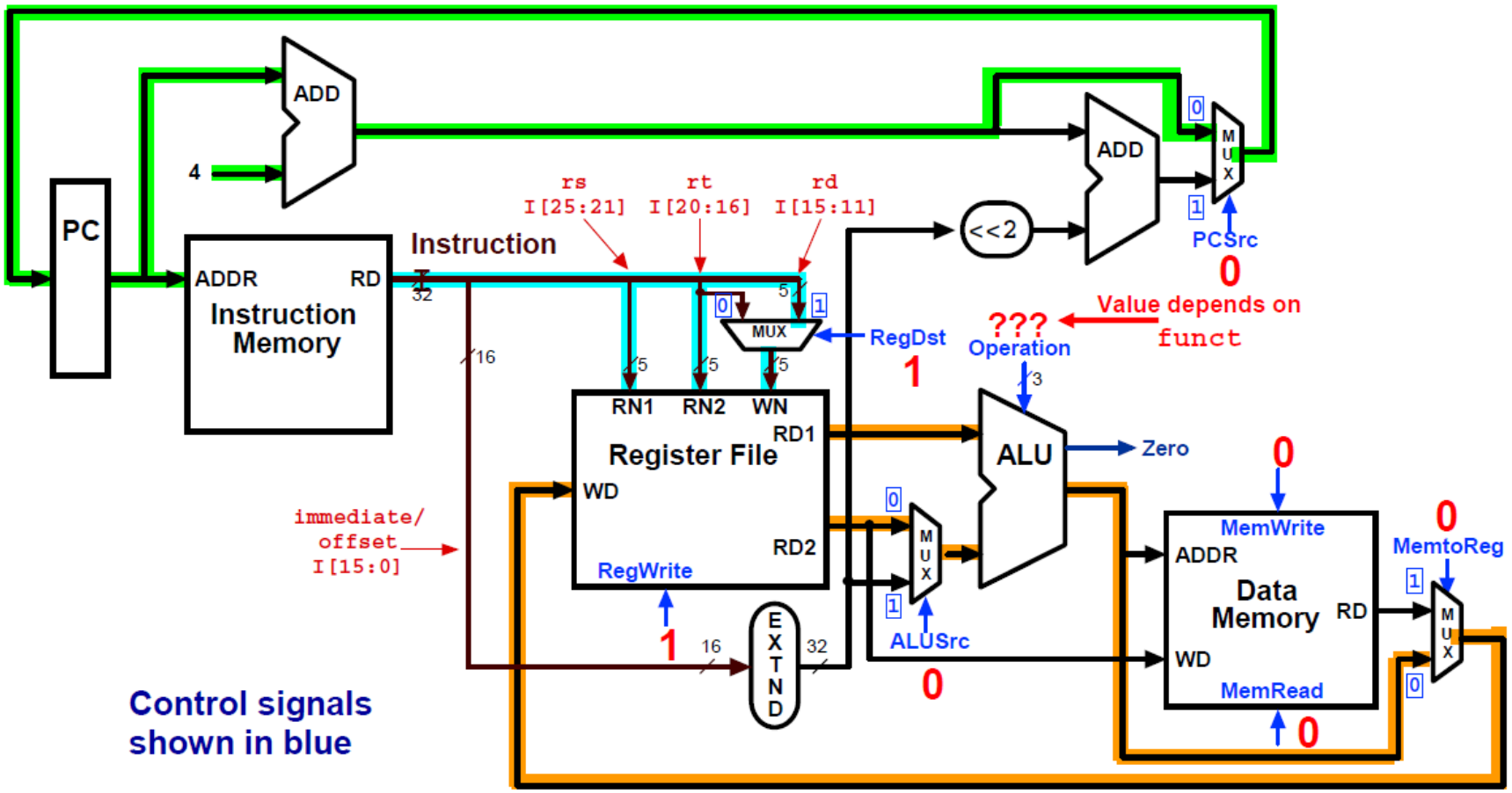


Determining control signals for the MIPS datapath based on instruction opcode

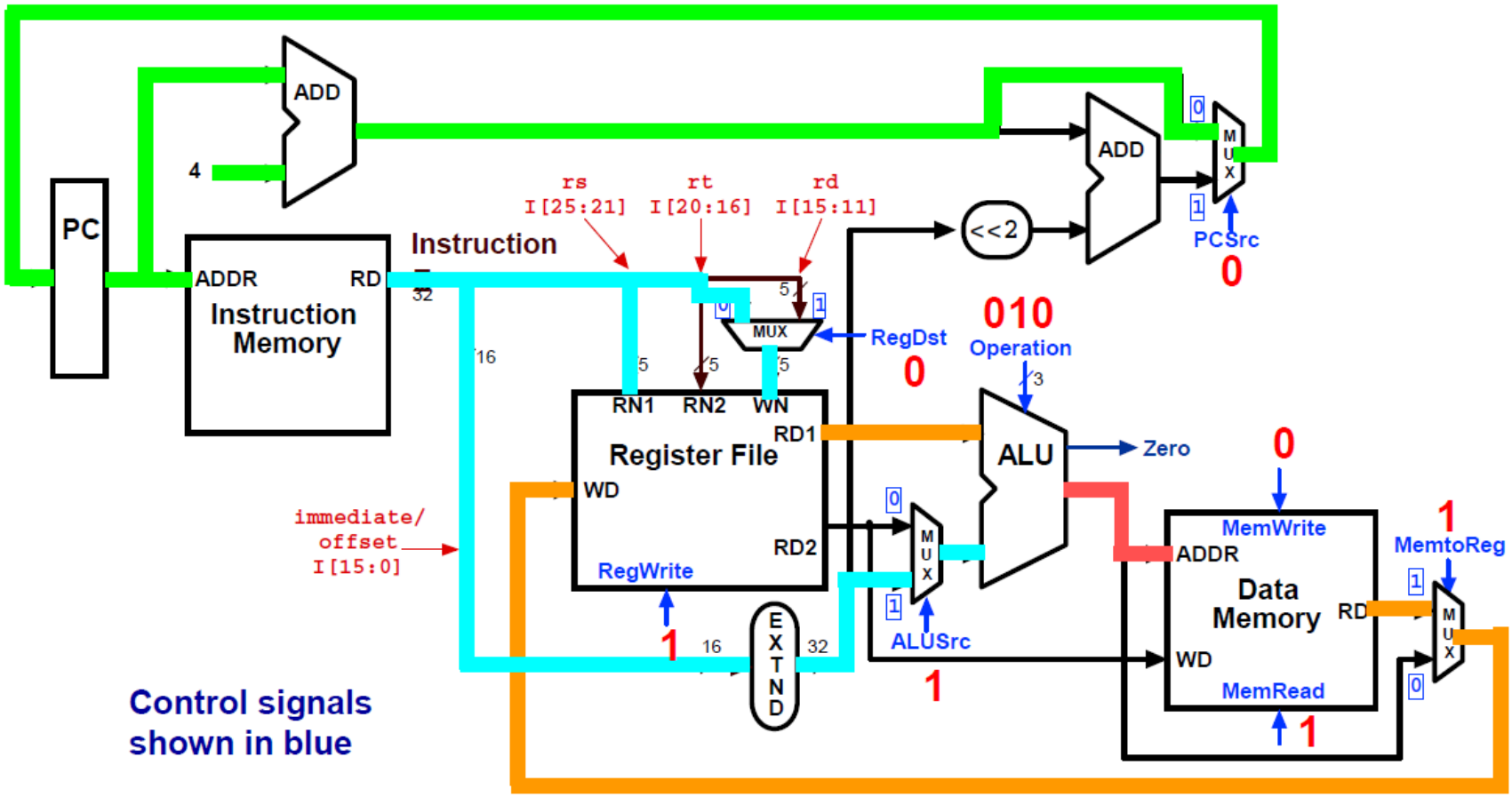
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Control Signals:

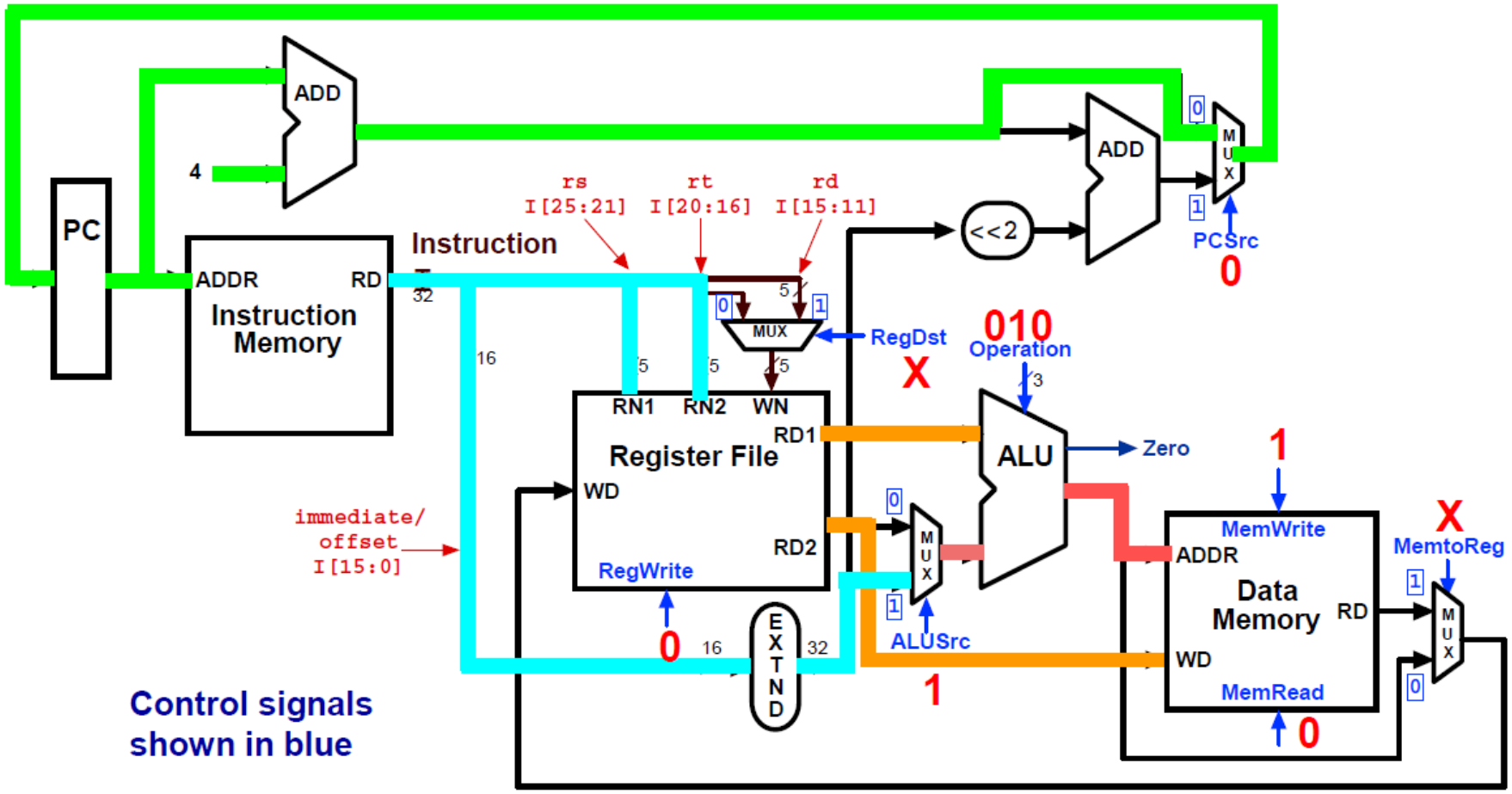
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0



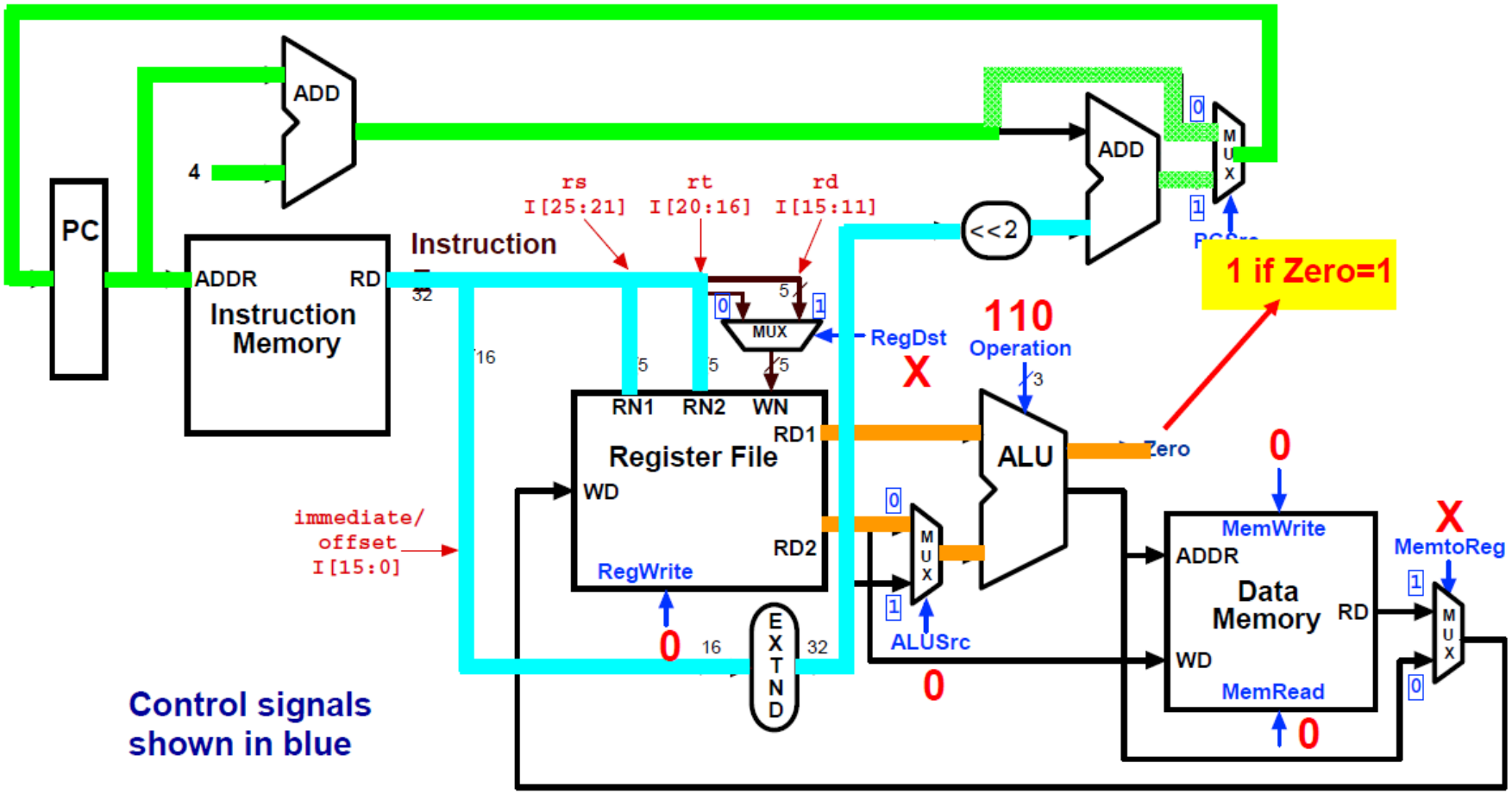
Control Signals: `lw` Instruction



Control Signals: `sw` Instruction



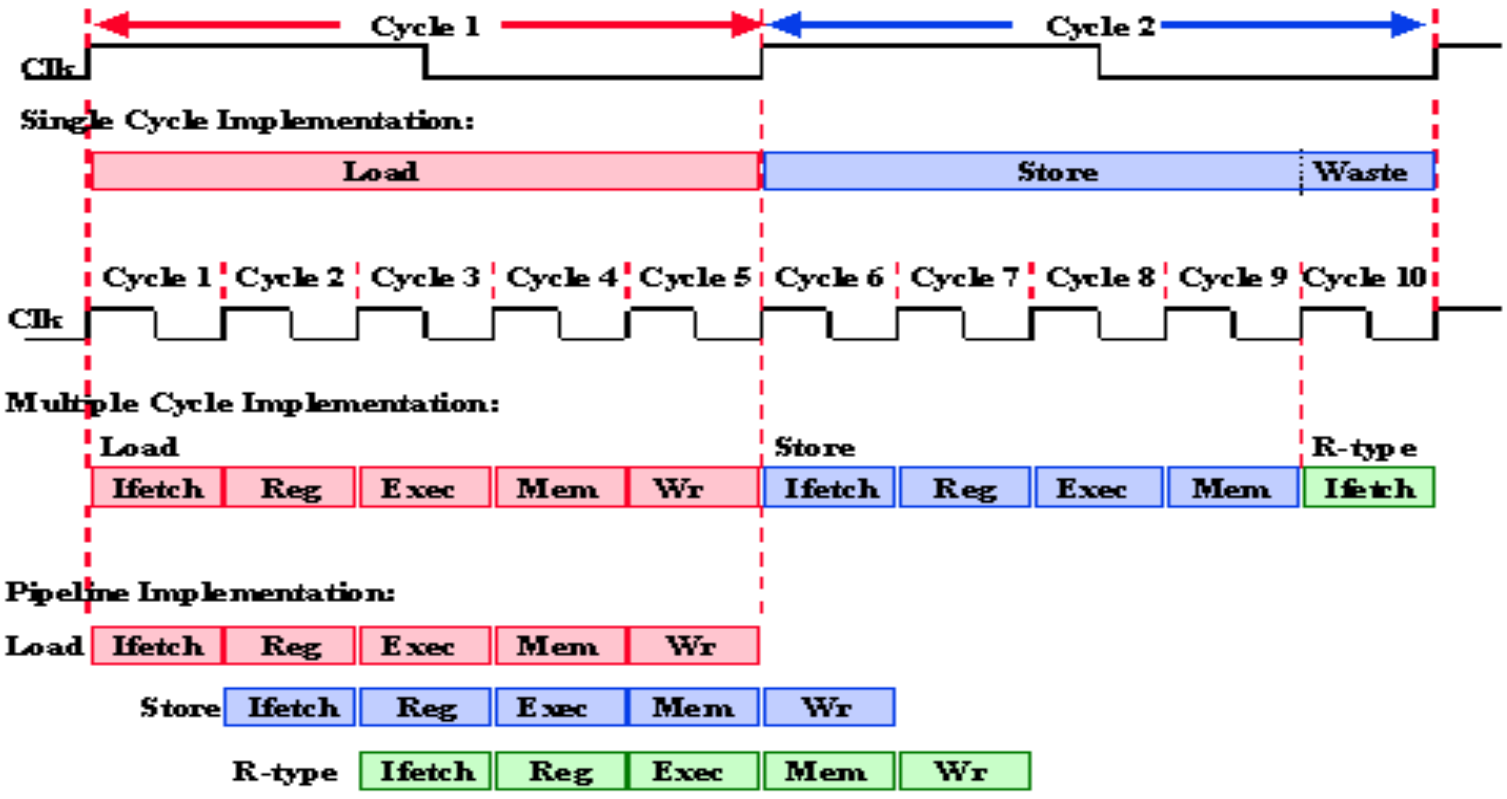
Control Signals: `beq` Instruction



A Multicycle Implementation

- Each step in the execution will take 1 clock cycle.
- Multicycle implementation allows a functional unit to be used more than once per instruction, as long as it is used on different clock cycles.
- This sharing can help reduce the amount of hardware required.

Single Cycle, Multiple Cycle, vs. Pipeline



A Multicycle Implementation

- **Advantages:**

- The ability to allow instructions to take different numbers of clock cycles.

- The ability to share functional units within the execution of a single instruction.

A Multicycle Implementation

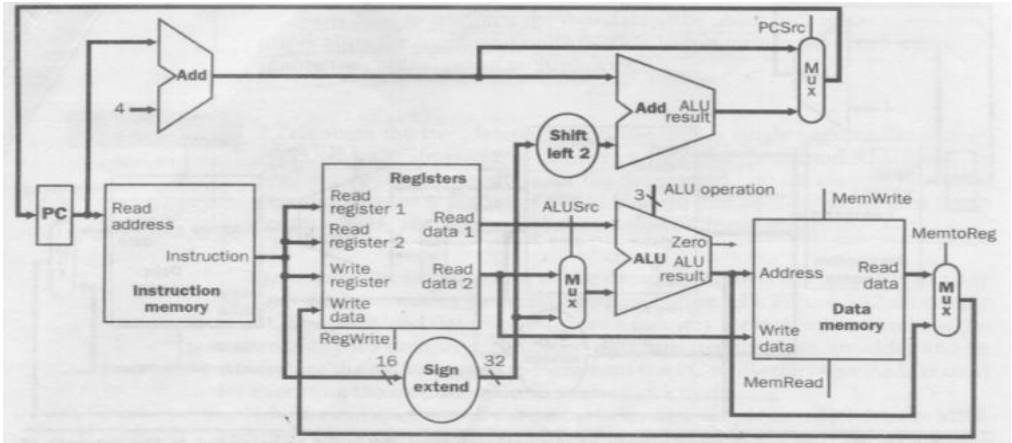


FIGURE 5.13 The simple datapath for the MIPS architecture combines the elements required by different instruction classes.

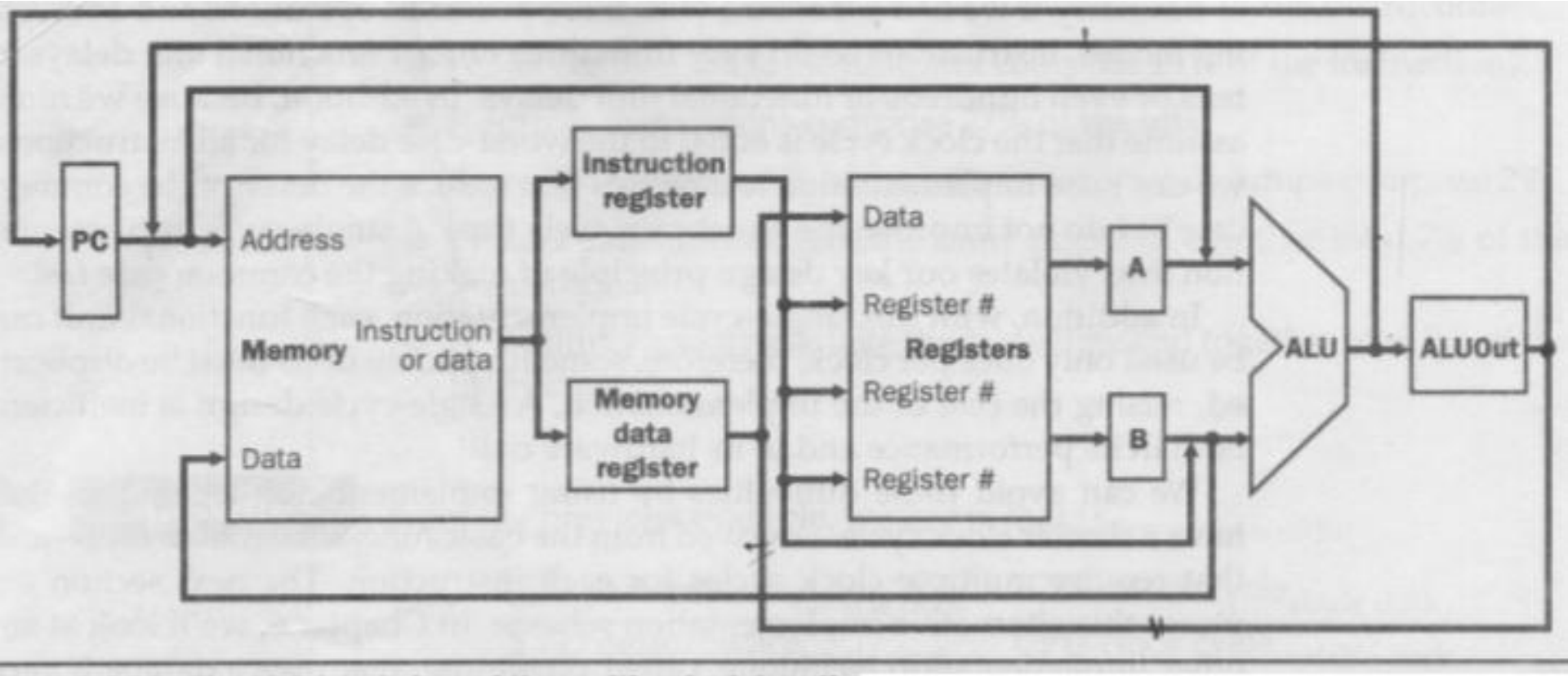


FIGURE 5.30 The high-level view of the multicycle datapath.

A Multicycle datapath for basic instructions

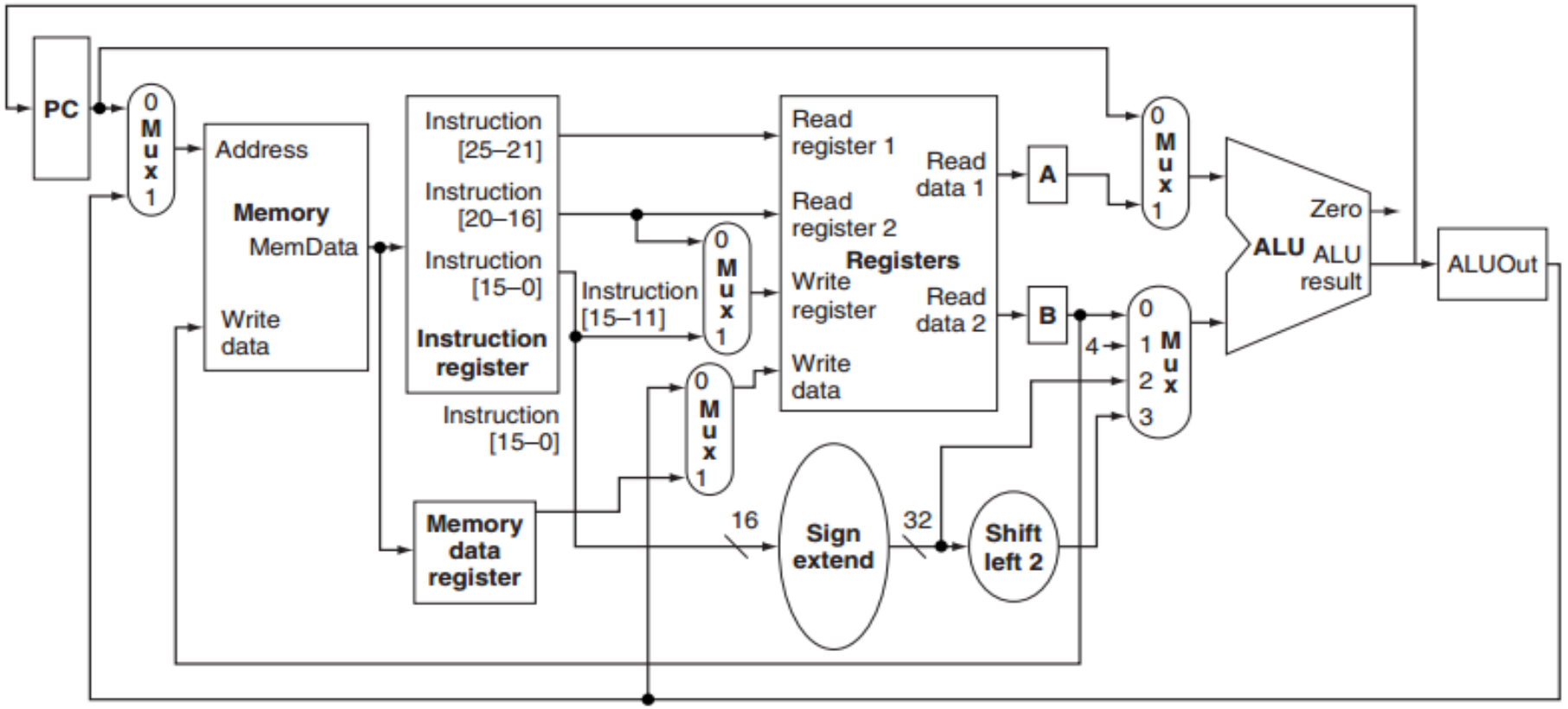


FIGURE 5.26 Multicycle datapath for MIPS handles the basic instructions. Although this datapath supports normal incrementing of the PC, a few more connections and a multiplexor will be needed for branches and jumps; we will add these shortly. The additions versus the single-clock datapath include several registers (IR, MDR, A, B, ALUOut), a multiplexor for the memory address, a multiplexor for the top ALU input, and expanding the multiplexor on the bottom ALU input into a four-way selector. These small additions allow us to remove two adders and a memory unit.

A Multicycle Implementation

- **New Registers:**

The following temporary registers are important to the multicycle datapath implementation discussed in this section:

- *Instruction Register* (IR) saves the data output of memory for a subsequent instruction read
- *Memory Data Register* (MDR) saves memory output for a data read operation;
- *A and B Registers* (A,B) store ALU operand values read from the register file and
- *ALU Output Register* (ALUout) contains the result produced by the ALU.

Differences between a single-cycle and multi-cycle datapath

- In the multicycle datapath, one memory unit stores both instructions and data, whereas the single-cycle datapath requires separate instruction and data memories.

- The multicycle datapath uses on ALU, versus an ALU and two adders in the single-cycle datapath.

- In the single-cycle implementation, the instruction executes in one cycle (by design) and the outputs of all functional units must stabilize within one cycle. In contrast, the multicycle implementation uses one or more registers to temporarily store (buffer) the ALU or functional unit outputs. This *buffering* action stores a value in a temporary register until it is needed or used in a subsequent clock cycle.