EEE304 – Digital Design with HDL (II)

Lecture 9

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- A Multicycle Implementation of the Processor

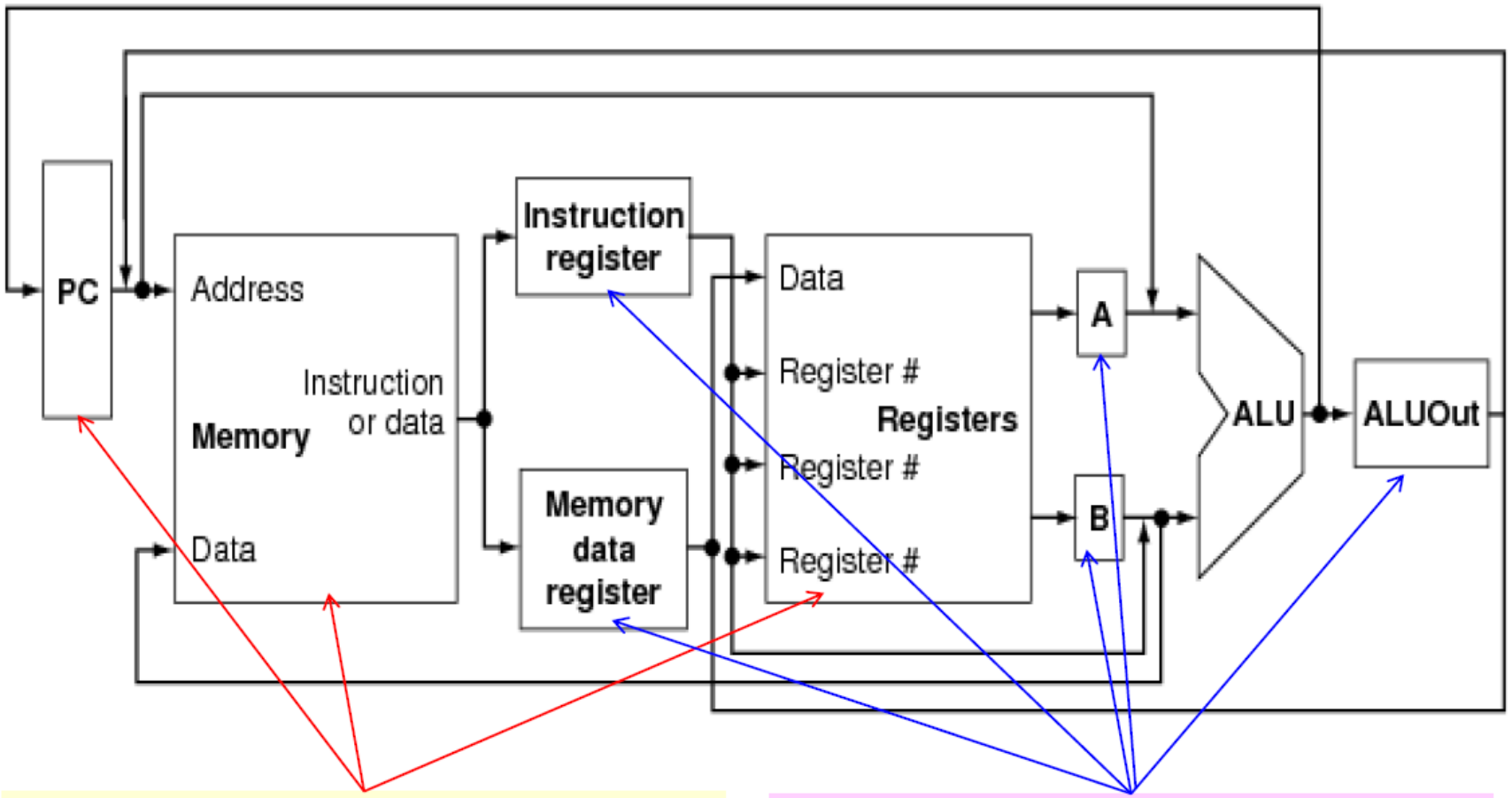
Multicycle Implementation

- Instructions are executed in a series of steps, and each step will take 1 clock cycle.
- Different instructions take different number of clock cycles.
- Multicycle implementation allows a functional unit to be used more than once per instruction, which helps to reduce the amount of hardware required.

Multicycle Implementation

- There is a single ALU rather than an ALU and two adders.
- A single memory is used for both instructions and data.
- One or more registers are added after every major functional unit to hold the output of that unit until the value is used in a subsequent clock cycle.

The high-level view of the multicycle datapath



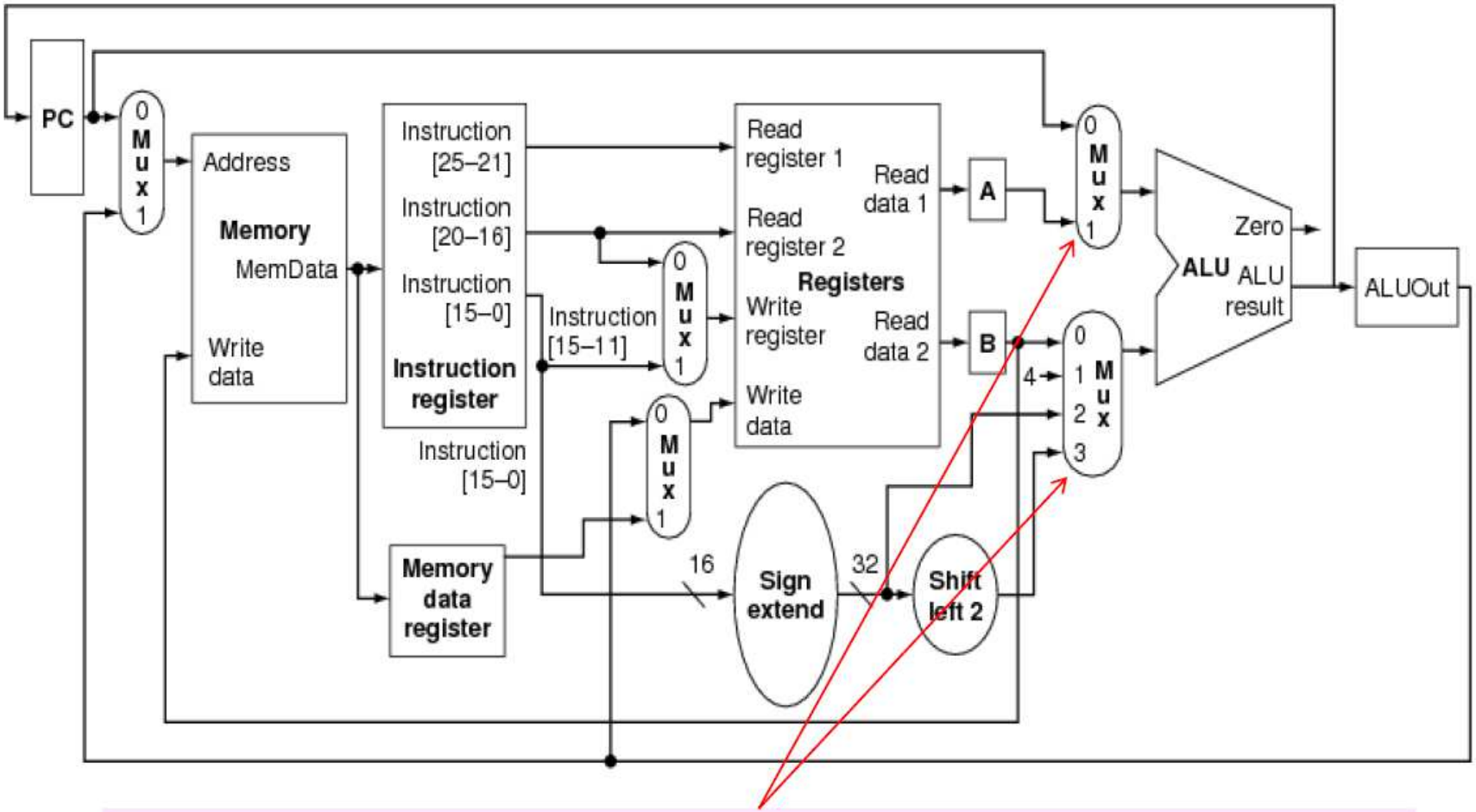
Data used by subsequent instructions is stored into one of the programmer-visible state elements.

Data used by the same instruction in a later cycle is stored into one of these additional registers.

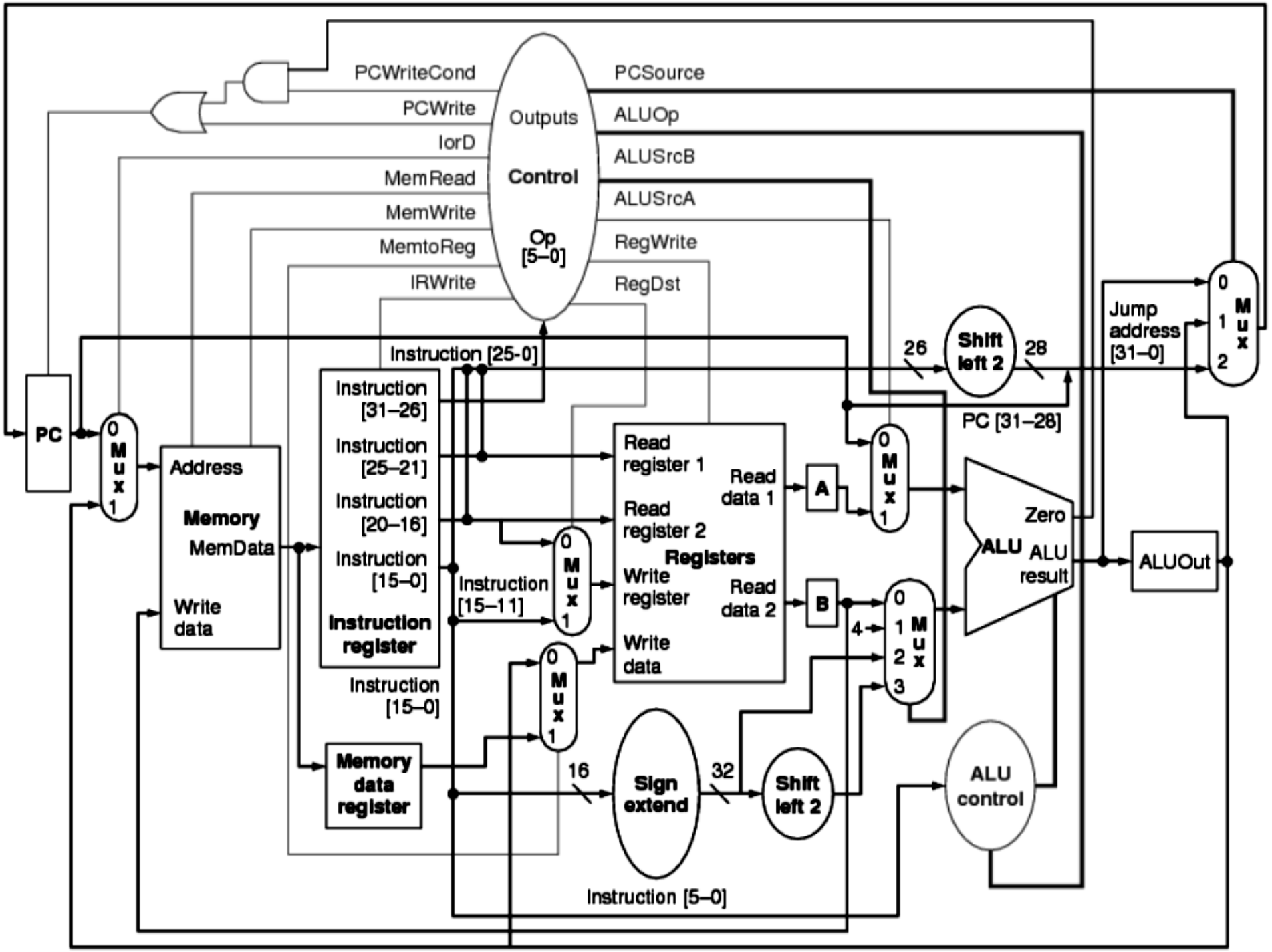
Temporary registers

- The Instruction register (**IR**) saves the output of the memory for an instruction read.
- The Memory data register (**MDR**) saves the output of the memory for a data read.
- Why are two separate registers used for the memory block?
- The **A** and **B** registers are used to hold the register operand values read from the register file.
- The **ALUOut** register holds the output of ALU.
- All the registers except the IR hold data only between a pair of adjacent clock cycles for each instruction.

Multicycle data path for MIPS handles the basic instructions.



Because several functional units are shared for different purposes we need both to add new multiplexers and to expand existing multiplexers.


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25-0]
26
Shift
left 2
28
Jump
address
[31–0]
M
u
x
0
1
2
PC [31–28]
PC
Address
Memory
MemData
Write
data
Instruction
[31–26]
Instruction
[25–21]
Instruction
[20–16]
Instruction
[15–0]
Instruction
register
Instruction
[15–11]
Instruction
[15–0]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
Zero
ALU
ALU
result
ALUOut
Memory
data
register
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5–0]

Actions of the 1-bit control signals

Signal Name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register number destination number for the Write register comes from the rd field.
RegWrite	None	The general-purpose register selected by the Write register input is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None	Content of memory at the location specified by the Address input is put on the Memory data output.
MemWrite	None	Memory contents at the location specified by the Address input is replaced by the value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by the PCSource.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

Actions of the 2-bit control signals

Signal Name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC+4) is sent to the PC for writing.
	01	The contents of the ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0]) shifted left 2 bits and concatenated with PC+4[31:28] is sent to the PC for writing.

Breaking the Instruction Execution into Clock Cycles

- The goal is to maximise the performance.
- We attempt to keep the amount of work per cycle roughly equal.
- At the end of every clock cycle any data values that will be needed on a subsequent cycle must be stored into a register (PC, RF, Memory, IR, A, B, MDR, AlUOut).
- The registers are written by the active clock edge. We can continue to read the register until the next clock cycle.
- All the operations listed in one step occur in parallel with 1 clock cycle, while successive steps operate in series in different clock cycles.

1. Instruction Fetch

IR <=Memory[PC]
PC <=PC + 4

- Send the PC to the memory and perform a read
- Write the instruction into the Instruction register (IR)
- Increment the PC by 4
- <= symbol means that all right-hand sides are evaluated and then all assignments are made at the active edge of the clock.
- Control signals: MemRead=1, IRWrite=1, IorD=0, ALUSrcA=0, ALUSrcB=01, ALUOp=00, PCSource=00, PCWrite=1

2. Instruction Decode and Register Fetch

A <=Reg[IR[25:21]]
B <=Reg[IR[20:16]]
ALUOut<=PC + (sign-extend(IR[15:0]) <<2)

- Since the instruction is not known yet, we take actions which are not harmful.
- rs and rt registers are read from the register file and are stored into the temporary registers A and B.
- The branch target address is computed which will be ignored if the instruction turns out not to be a branch.
- Control Signals: ALUSrcA=0, ALUSrcB= 11, ALUOp=00

3. Execution: Memory Address Computation, or Arithmetic-Logical operation

***Memory reference*:**

ALUOut<=A + sign-extend (IR[15:0]);

Operation: The ALU is adding the operands to form the memory address

Control Signals: ALUSrcA=1, ALUSrcB= 10, ALUOp=00

***Arithmetic-Logical Instruction (R-type)*:**

ALUOut<=A op B;

*Operation:*The ALU is performing the operation specified by the function code on the two values read from the register file in the previous cycle.

Control Signals: ALUSrcA=1, ALUSrcB= 00, ALUOp=10

3. Execution: Branch or Jump completion

***Branch*:**

if (A == B) PC <=ALUOut;

Operation: The ALU is used to do the equal comparison between the two registers read in the previous step.

Control Signals: ALUSrcA=1, ALUSrcB= 00, ALUOp=01, PCWriteCond=1, PCSource= 01;

***Jump*:**

PC <={ PC[31:28] , (IR[25:0] , 2'b00) };

*Operation:*The PC is replaced by the jump address.

Control Signals: PCSource= 10, PCWrite=1;

4. Memory Access or R-type Instruction Completion

***Memory reference*:**

*load:*MDR <=Memory [ALUOut] ; or
*store:*Memory [ALUOut] <=B ;

Operation: The address used is the one computed during the previous step and stored in ALUOut.

Control Signals : load: MemRead=1, IorD=1;
store: MemWrite=1, IorD=1;

***Arithmetic-Logical Instruction (R-type)*:**

Reg[IR [15:11]] <=ALUOut;

Operation: Place the content of the ALUOut into the result register (rd).

Control Signals: RegDst=1, RegWrite=1, MemtoReg=0;

5. Memory Read Completion step

load: Reg[IR[20:16]] <=MDR;

Operation: Write the load data, which was stored into MDR in the previous cycle, into the register file.

Control Signals:

MemtoReg=1 (write the result from memory)

RegWrite=1 (cause a write to register file)

RegDst=0 (choose the rt field as the register number)

Summary of the steps taken to execute any instruction class

Step name	Action for R-type Instructions	Action for memory reference instructions	Action for branches	Action for jumps
Instruction fetch	IR <= Memory[PC] PC <= PC + 4			
Instruction decode /register fetch	A <= Reg [IR[25:21]] B <= Reg [IR[20:16]] ALUOut <= PC + (sign-extend(IR[15:0]) <<2)			
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + Sign-extend (IR [15:0])	if (A==B) PC <= ALUOut	PC <= {PC[31:28], (IR[25:0] , 2'b00) }
Memory access or R-type completion	Reg [IR[15:11]] <= ALUOut	Load: MDR<=Memory[ALUOut] Save: Memory[ALUOut] <= B		
Memory read completion		Load: Reg [IR[20:16]] <= MDR		

CPI in a Multicycle Proceesor

- The SPECINT2000 instruction mix has: 25% loads, 10% stores, 11% branches, 2% jumps and 52% ALU instructions.
- We assume each state in the multicycle CPU requires 1 clock cycle and the number of clock cycles per instruction are: Loads= 5, Stores = 4, Branches= 3 ALU instructions = 4, Jumps= 3

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}} = \frac{\sum \text{Instruction count}_i \times \text{CPI}_i}{\text{Instruction count}}$$

$$\text{CPI} = \sum \frac{\text{Instruction count}_i}{\text{Instruction count}} \times \text{CPI}_i$$

$$\text{CPI} = 0.25 \times 5 + 0.10 \times 4 + 0.52 \times 4 + 0.11 \times 3 + 0.02 \times 3 = 4.12$$