

Where we are headed

❑ Single Cycle Problems:

- what if we had a more complicated instruction like floating point?
- wasteful of area

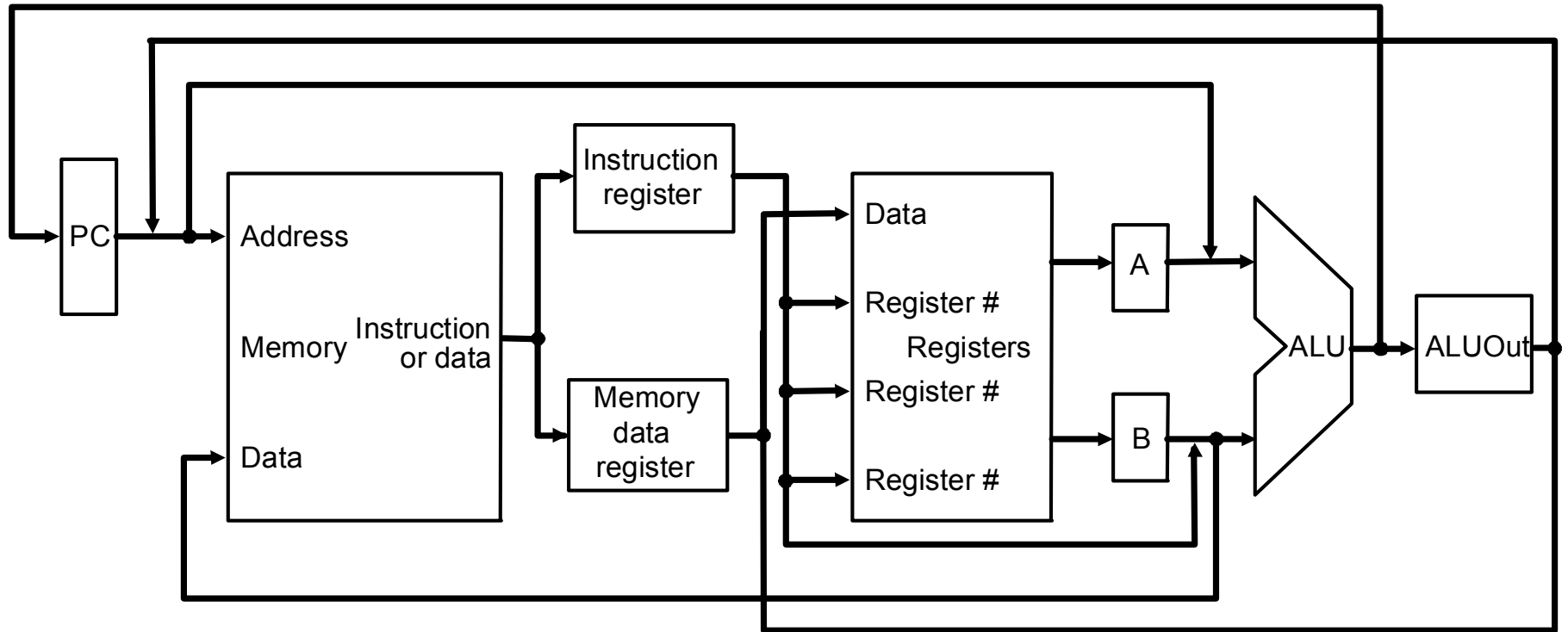
❑ One Solution:

- use a “smaller” cycle time
- have different instructions take different numbers of cycles
- a “multicycle” datapath:

❑ We will be reusing functional units

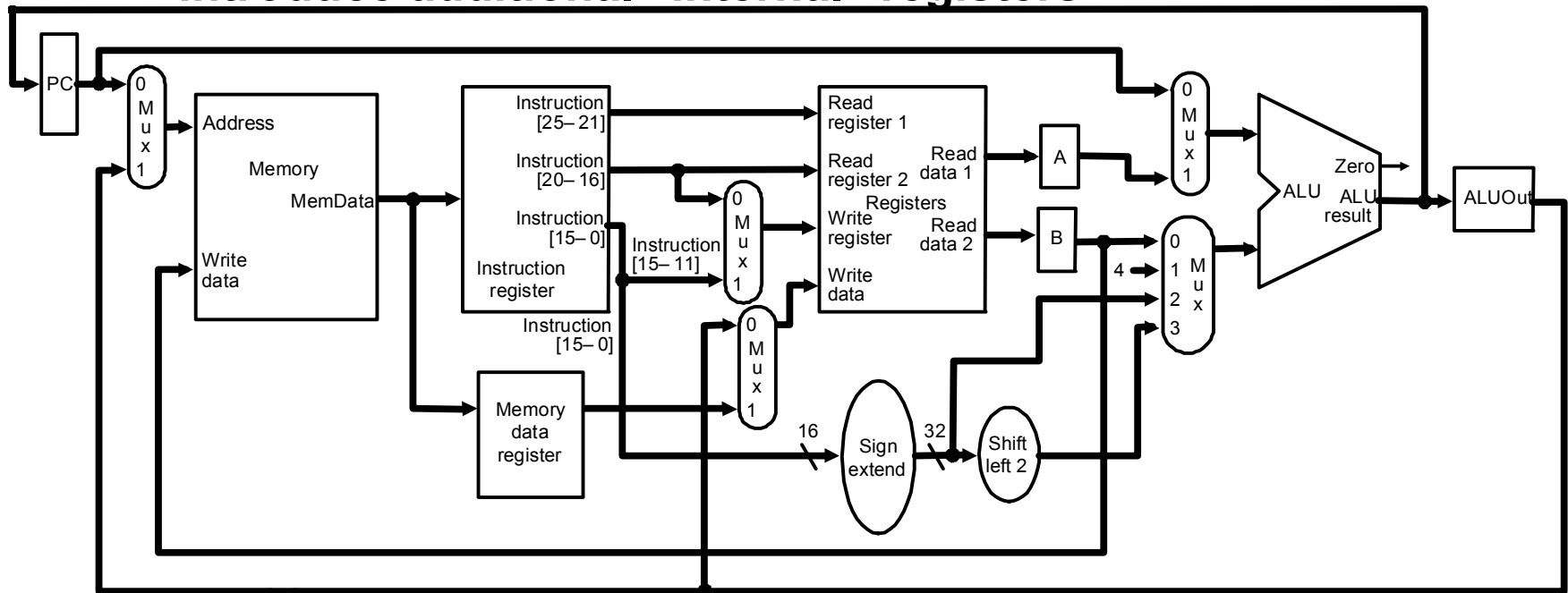
- ALU used to compute address and to increment PC
- Memory used for instruction and data

Multicycle Approach

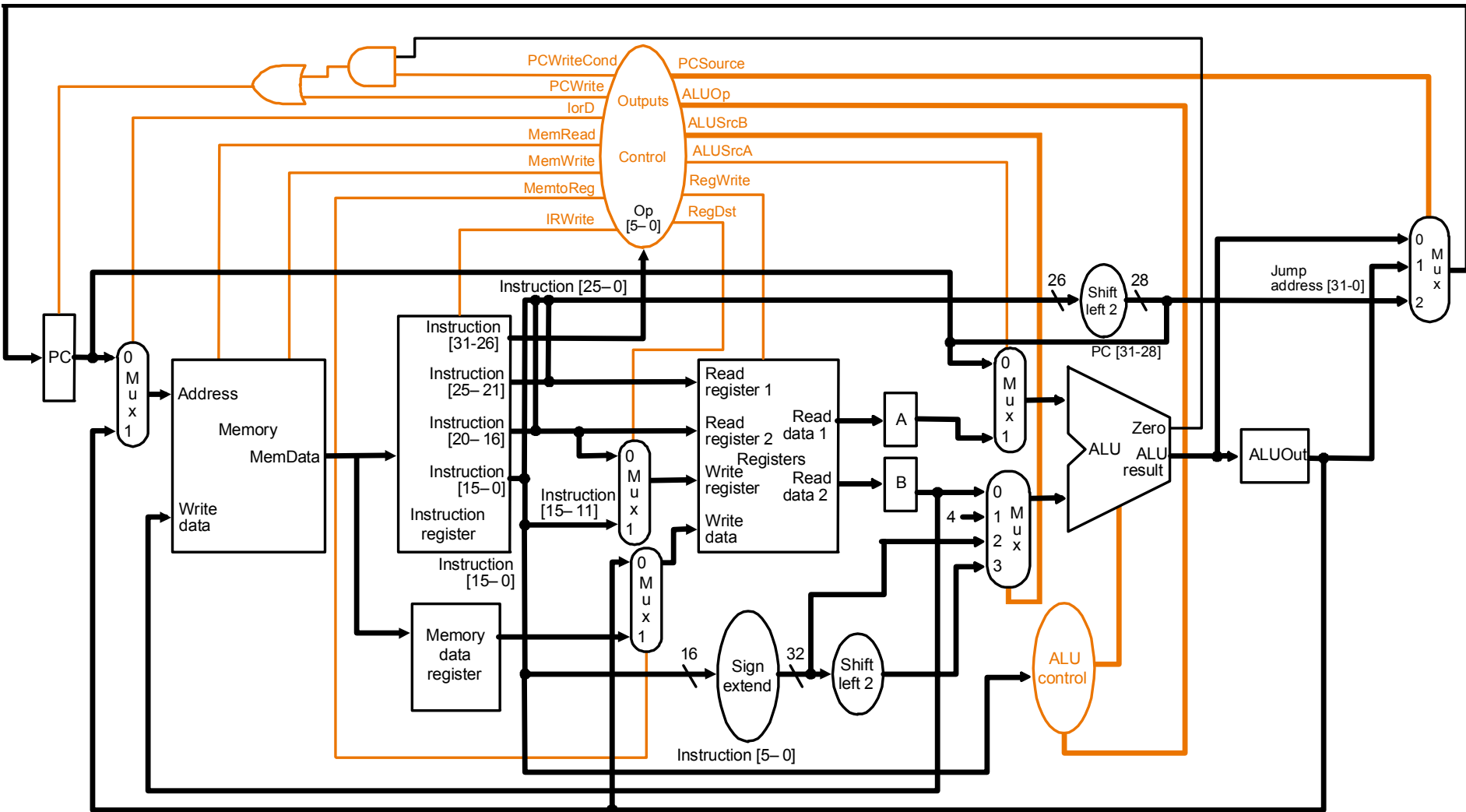


Multicycle Approach

- ❑ Break up the instructions into steps, each step takes a cycle
 - balance the amount of work to be done
 - restrict each cycle to use only one major functional unit
- ❑ At the end of a cycle
 - store values for use in later cycles (easiest thing to do)
 - introduce additional “internal” registers



Datapath of Multicycle Implementation



Five Execution Steps

- ❑ Instruction Fetch (F)
- ❑ Instruction Decode and Register Fetch (D)
- ❑ Execution, Memory Address Computation, or Branch Completion (EX)
- ❑ Memory Access or R-type instruction completion (M)
- ❑ Write-back step (W)

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

Step 1: Instruction Fetch

- ❑ Use PC to get instruction and put it in the Instruction Register.
- ❑ Increment the PC by 4 and put the result back in the PC.
- ❑ Can be described succinctly using RTL "Register-Transfer Language"

```
IR = Memory[PC];  
PC = PC + 4;
```

Can we figure out the values of the control signals?

MemRead=0; assert IRWrite; lrd=0; (IR = Memory[PC])

ALUSrcA=0; ALUSrcB=01; ALUOp=00; (PC = PC + 4)

PCSource=00; assert PCWrite; (store the incremented instruction address back into the PC)

Step 2: Instruction Decode and Register Fetch

- ❑ Read registers *rs* and *rt* in case we need them
- ❑ Compute the branch address in case the instruction is a branch
- ❑ RTL:

$A = \text{Reg}[\text{IR}[25-21]];$

$B = \text{Reg}[\text{IR}[20-16]];$

$\text{ALUOut} = \text{PC} + (\text{sign-extend}(\text{IR}[15-0]) \ll 2);$

$\text{ALUSrcA} = 0; \text{ALUSrcB} = 11; \text{ALUOp} = 00; \text{(branch target)}$

Step 3 (instruction dependent)

- ❑ ALU is performing one of three functions, based on instruction type

- ❑ Memory Reference:

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

- ❑ R-type:

`ALUOut = A op B;`

- ❑ Branch:

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

What are the control signals?

Step 4 (R-type or memory-access)

- ❑ Loads and stores access memory

```
MDR = Memory[ALUOut];  
    or  
Memory[ALUOut] = B;
```

- ❑ R-type instructions finish

```
Reg[IR[15-11]] = ALUOut;
```

The write actually takes place at the end of the cycle on the edge

Write-back step

□ `Reg[IR[20-16]] = MDR;`

What about all the other instructions?

Summary:

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