The Processor: Datapath

Dr. Vincent C. Emeakaroha
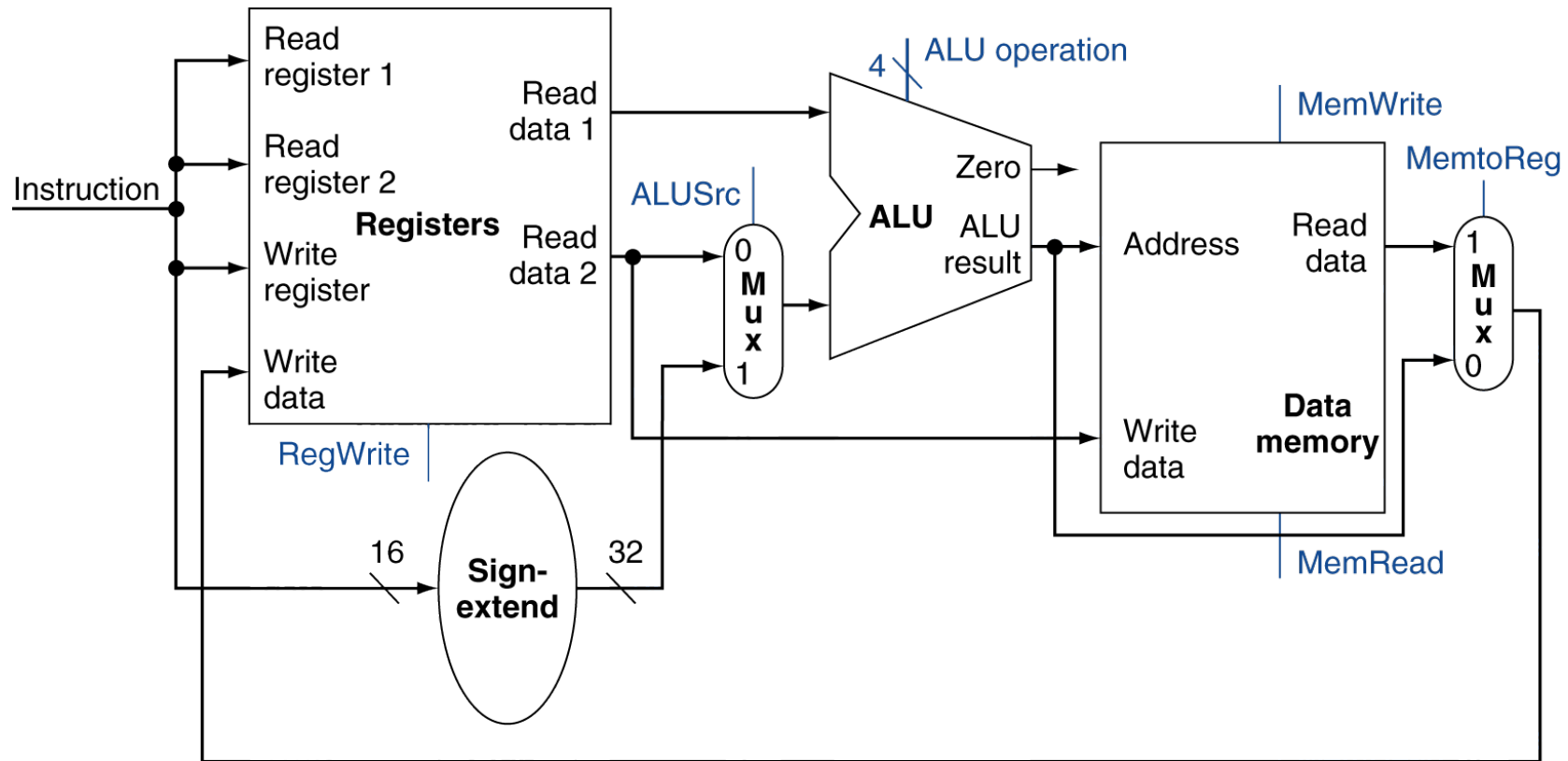
15-02-2017
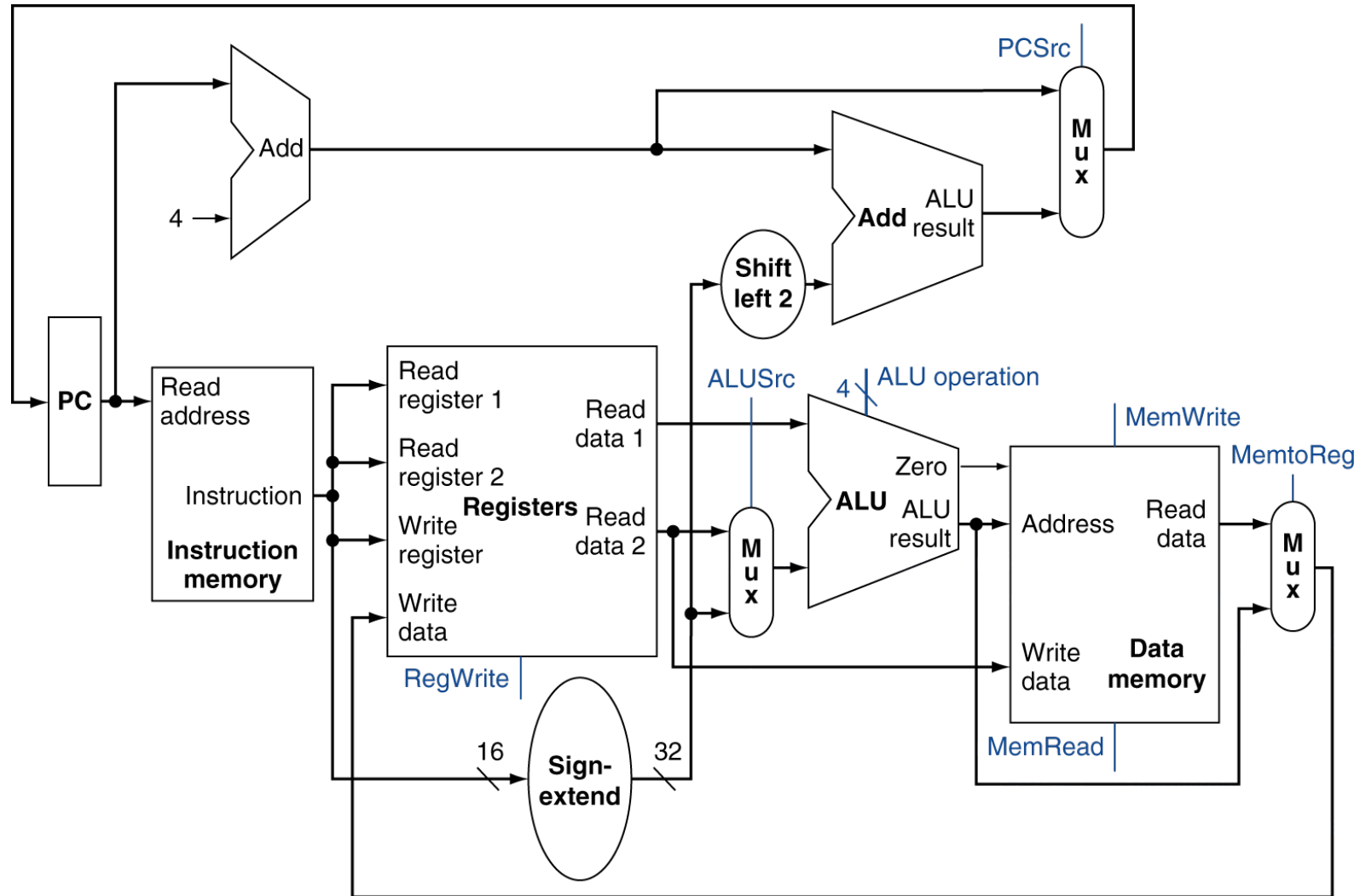
vc.emeakaroha@cs.ucc.ie

Composing the Elements

- First-cut data path does an instruction in one clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions

R-Type/Load/Store Datapath

Full Datapath

ALU Control

- ALU used for
  - Load/Store: F = add
  - Branch: F = subtract
  - R-type: F depends on funct field

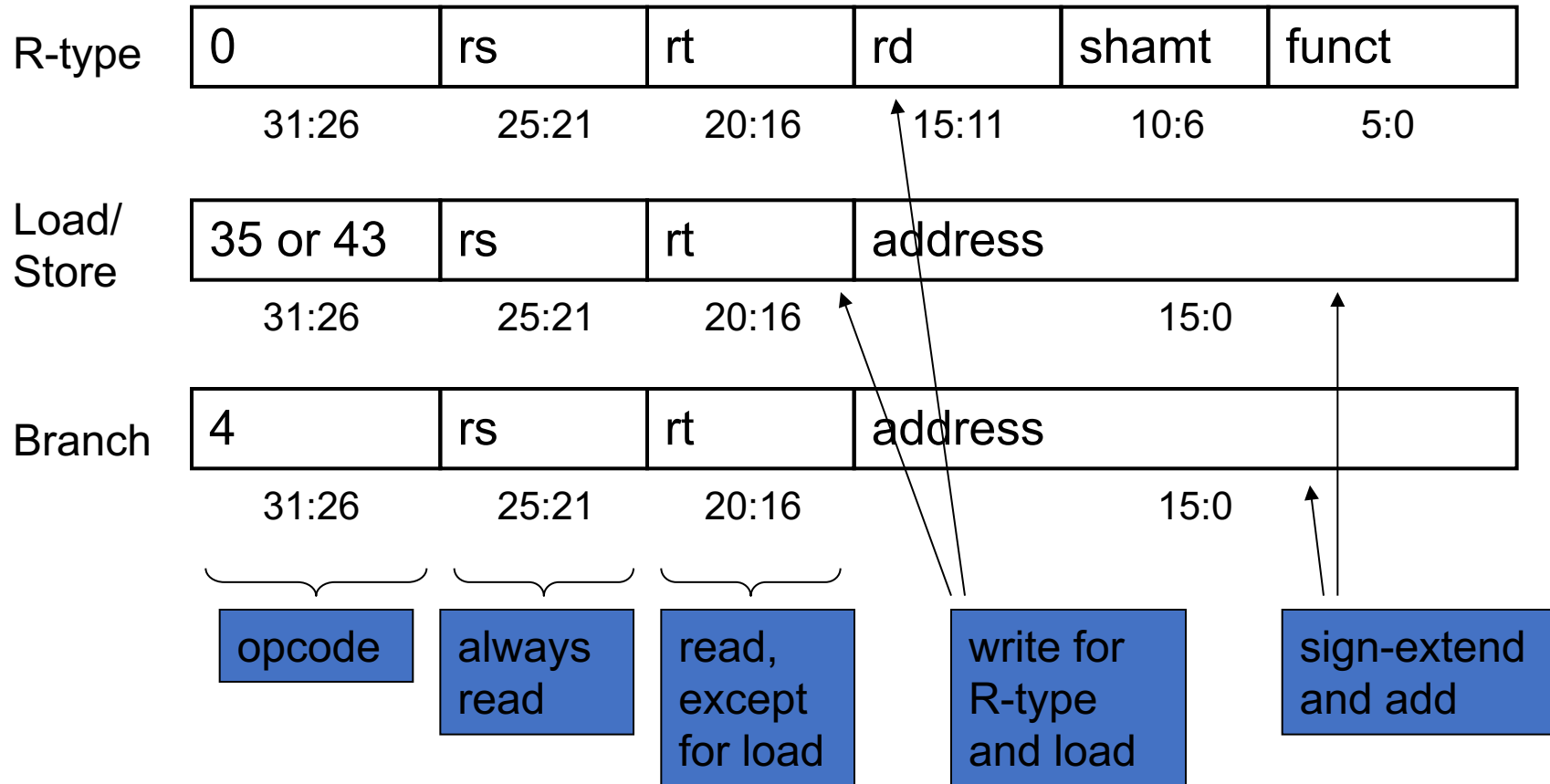| ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

ALU Control

- Assume 2-bit ALUOp derived from opcode
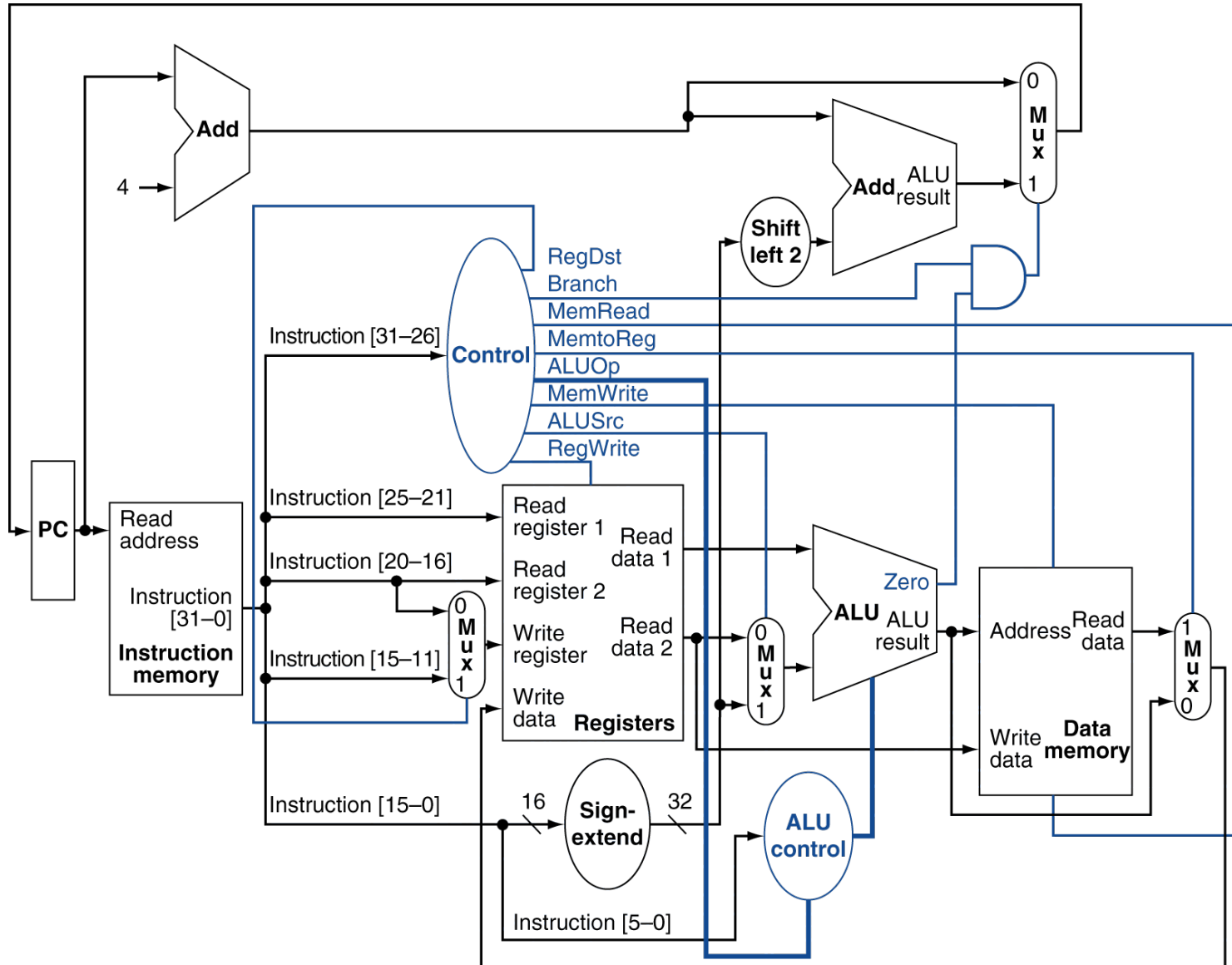 - Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

The Main Control Unit

- Control signals derived from instruction

R-type

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

Load/Store

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

Branch

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

opcode

always read

read, except for load

write for R-type and load
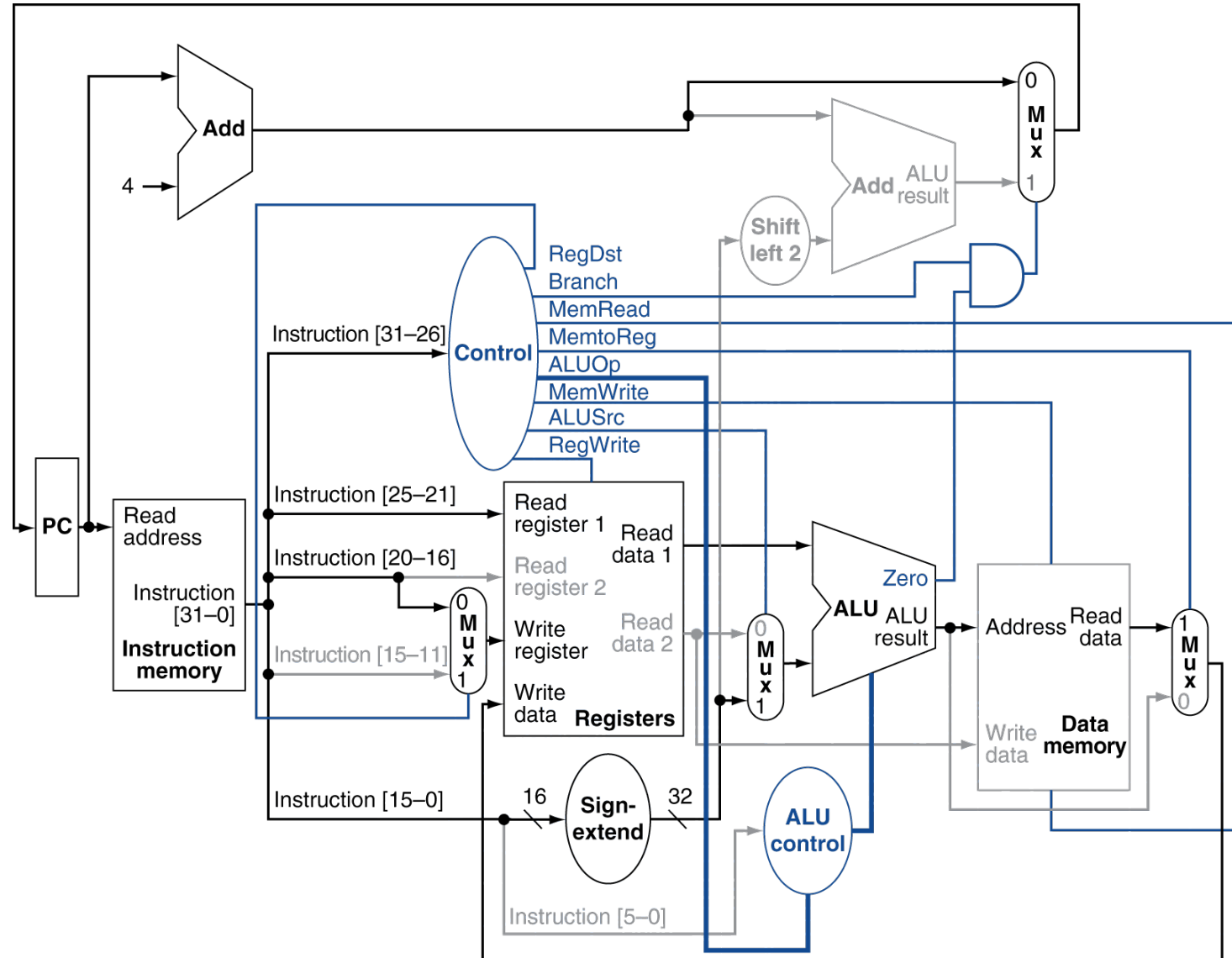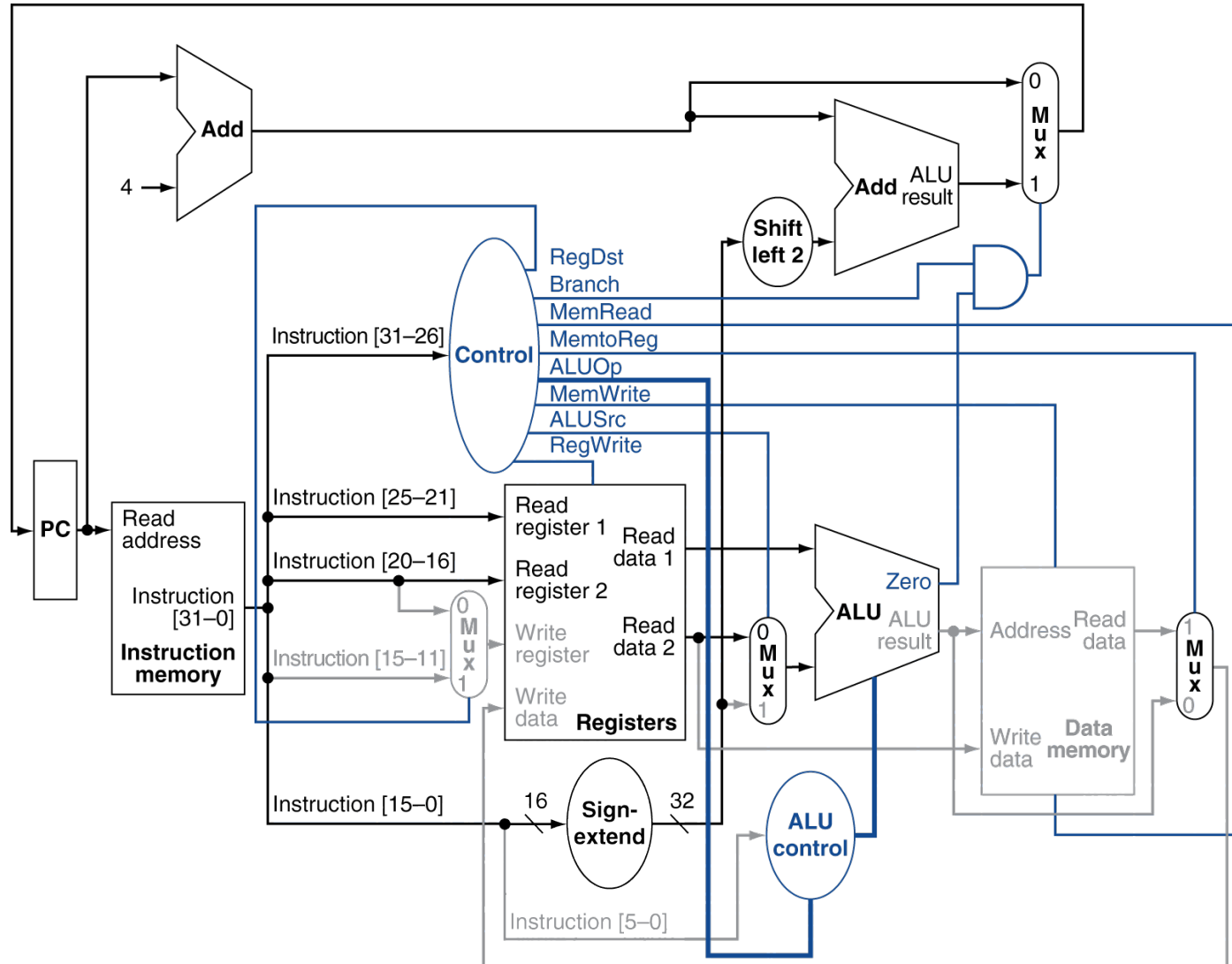
sign-extend and add

Datapath With Control

R-Type Instruction
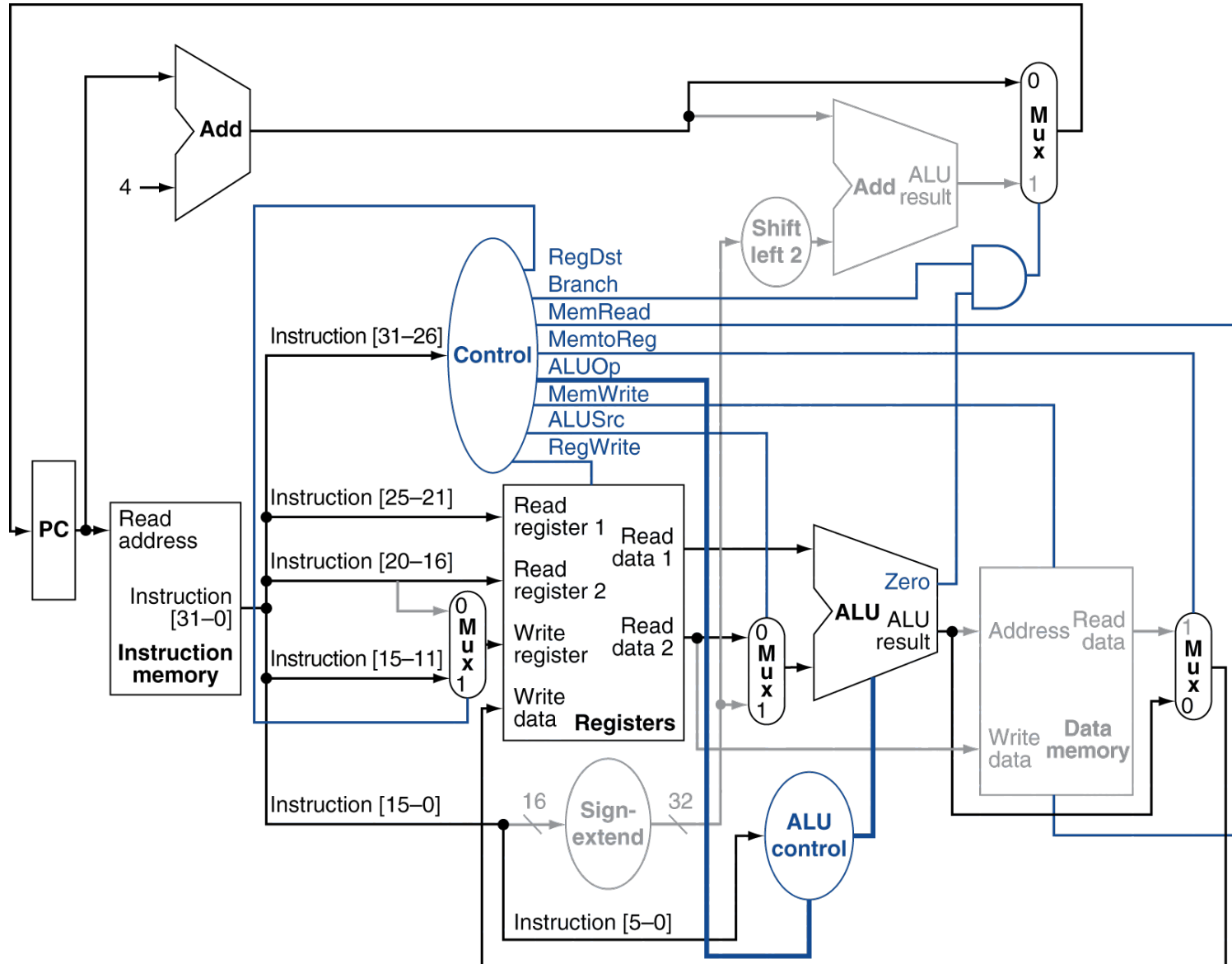
Load Instruction

Branch-on-Equal Instruction

Implementing Jumps

Jump

2	address

31:26   25:0
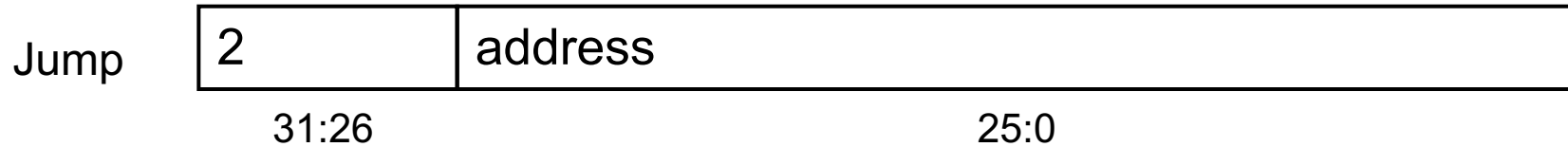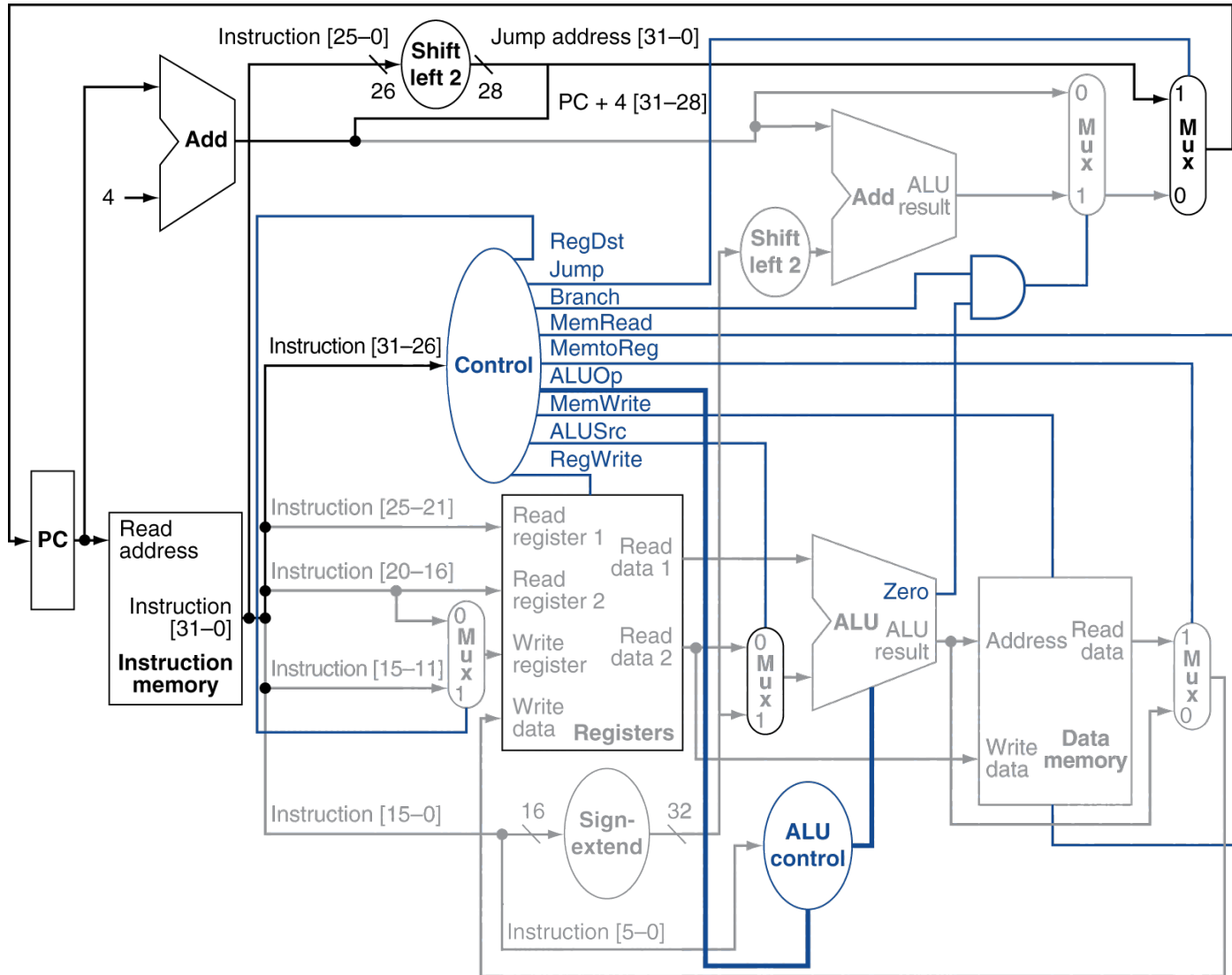
- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
  - 00
- Need an extra control signal decoded from opcode
  - Called a jump control
  - Asserted (set) only when opcode has the value of 2

Datapath With Jumps Added

Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory $\rightarrow$ register file $\rightarrow$ ALU $\rightarrow$ data memory $\rightarrow$ register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Making the common case fast
- A solution to improve performance is by pipelining

Pipelining Analogy

- Pipelined laundry: overlapping execution
  - Parallelism improves performance
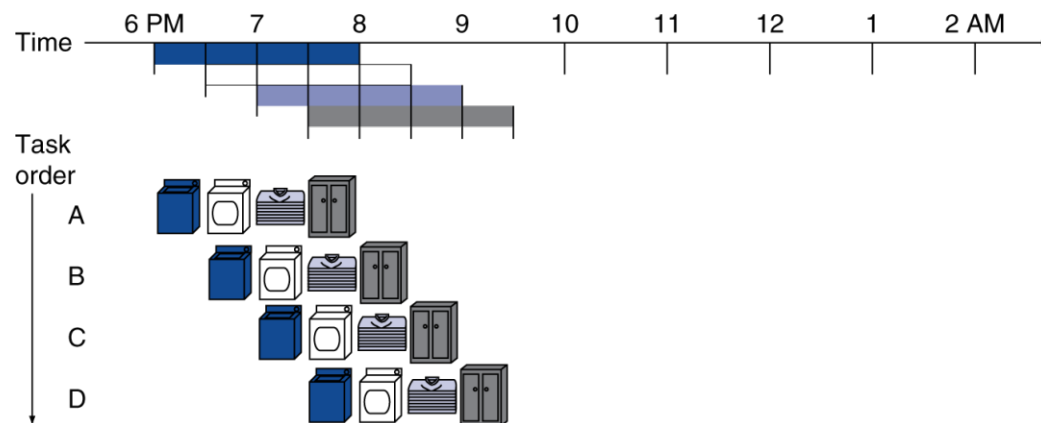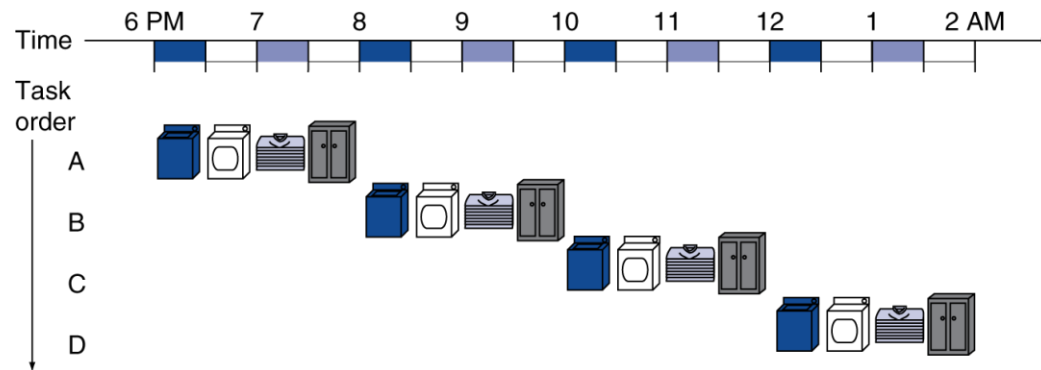
- Four loads:
  - Sequential execution
- Non-stop:
  - Speedup achieved with pipelining
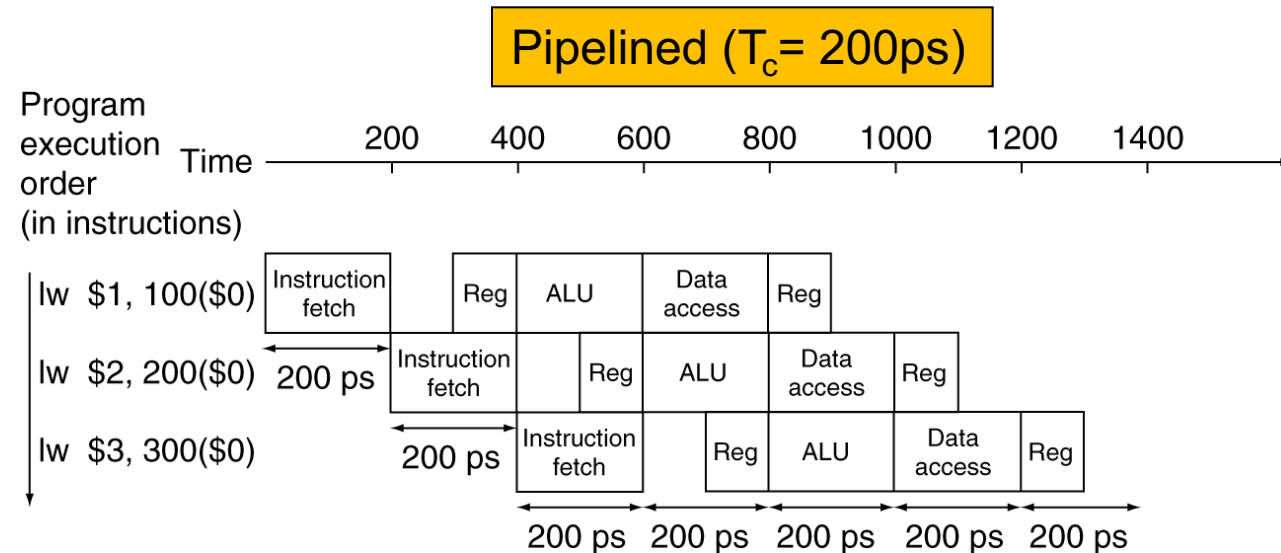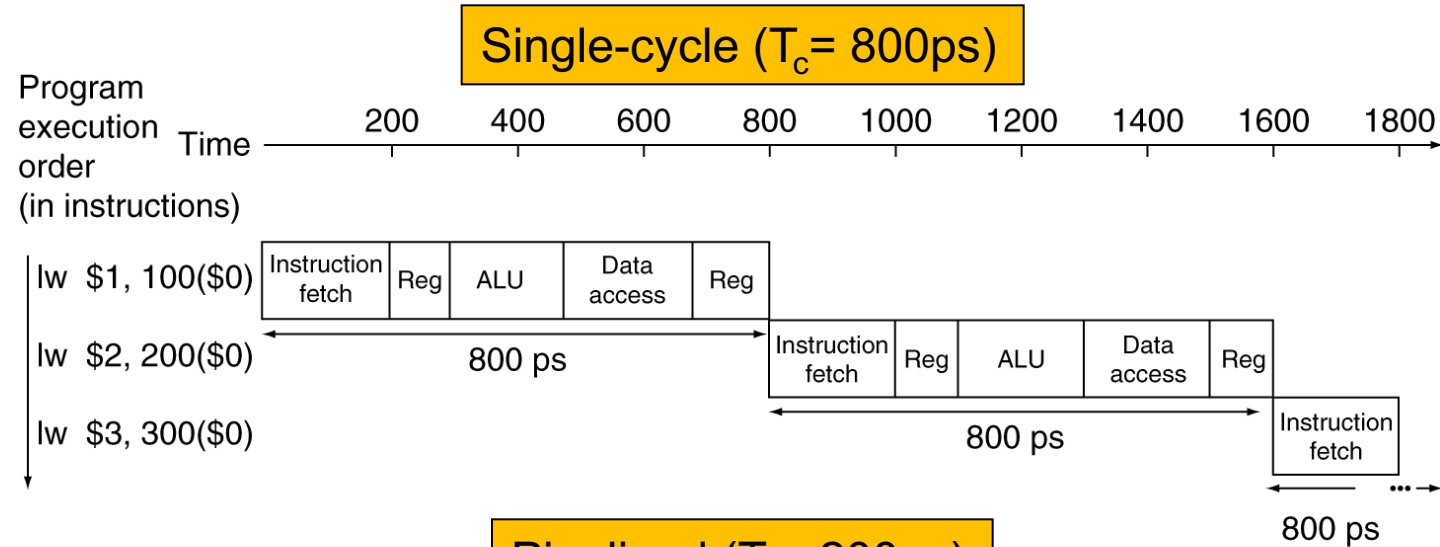
MIPS Pipeline

- Five stages, one step per stage
    1. IF: Instruction fetch from memory
    2. ID: Instruction decode & register read
    3. EX: Execute operation or calculate address
    4. MEM: Access memory operand
    5. WB: Write result back to register

Pipeline Performance

- Assume time for stages is
    - 100ps for register read or write
    - 200ps for other stages
- Table below shows the clock-cycle time for single-cycle datapath

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance Comparison

Pipeline Speedup

- If all stages are balanced
    - i.e., all take the same time
    - Time between instructions$_{pipelined}$

    $$= \frac{\text{Time between instructions}_{nonpipelined}}{\text{Number of stages}}$$

- If not balanced, speedup is less

- Speedup due to increased throughput
    - Latency (time for each instruction) does not decrease

Pipelining and ISA Design

- MIPS ISA designed for pipelining
  - All instructions are 32-bits
    - Easier to fetch and decode in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands
    - Memory access takes only one cycle

Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

Structure Hazards

- Conflict for use of a resource

- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline "bubble"

- Hence, pipelined datapaths require separate instruction/data memories
  - Or separate instruction/data caches