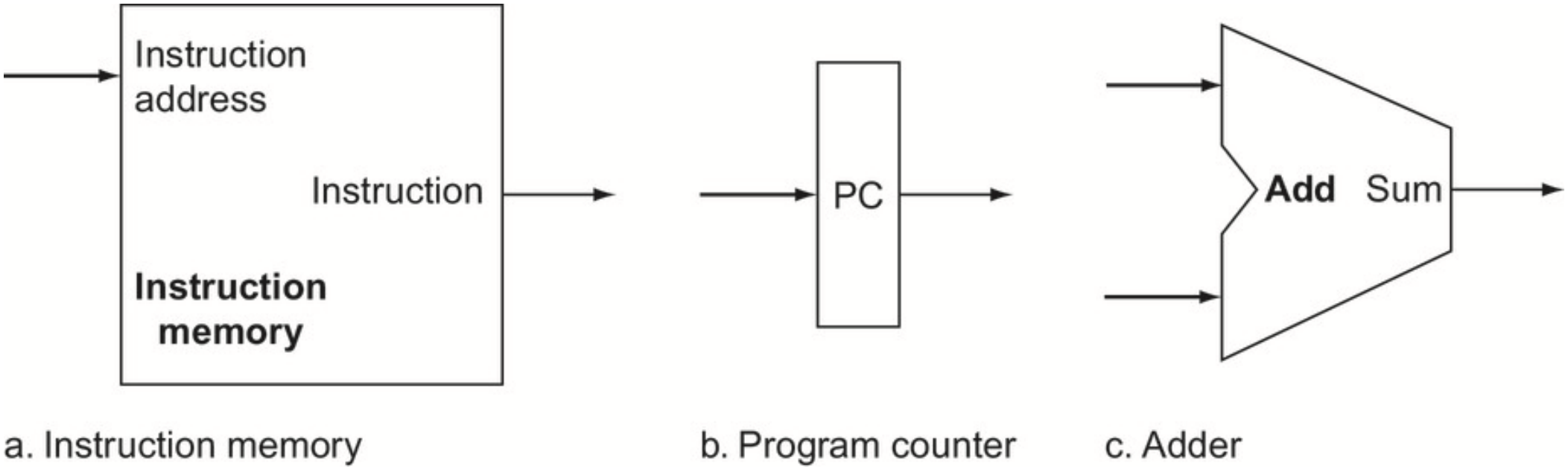
Lesson 10

Processors Continued

Building a datapath

- ***Datapath element is a*** unit used to operate on or hold data within a processor.
- In the MIPS implementation, the datapath elements include the instruction and data memories, the register file, the ALU, and adders.
- *Program Counter (PC) is the first element we need in a datapath.*
- *PC* is a register that holds the address of the current instruction.
- ***Program counter*** (***PC***): The register containing the address of the next instruction in the program to be executed.
- Next, we need an adder to increment the PC to the address of the next instruction

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

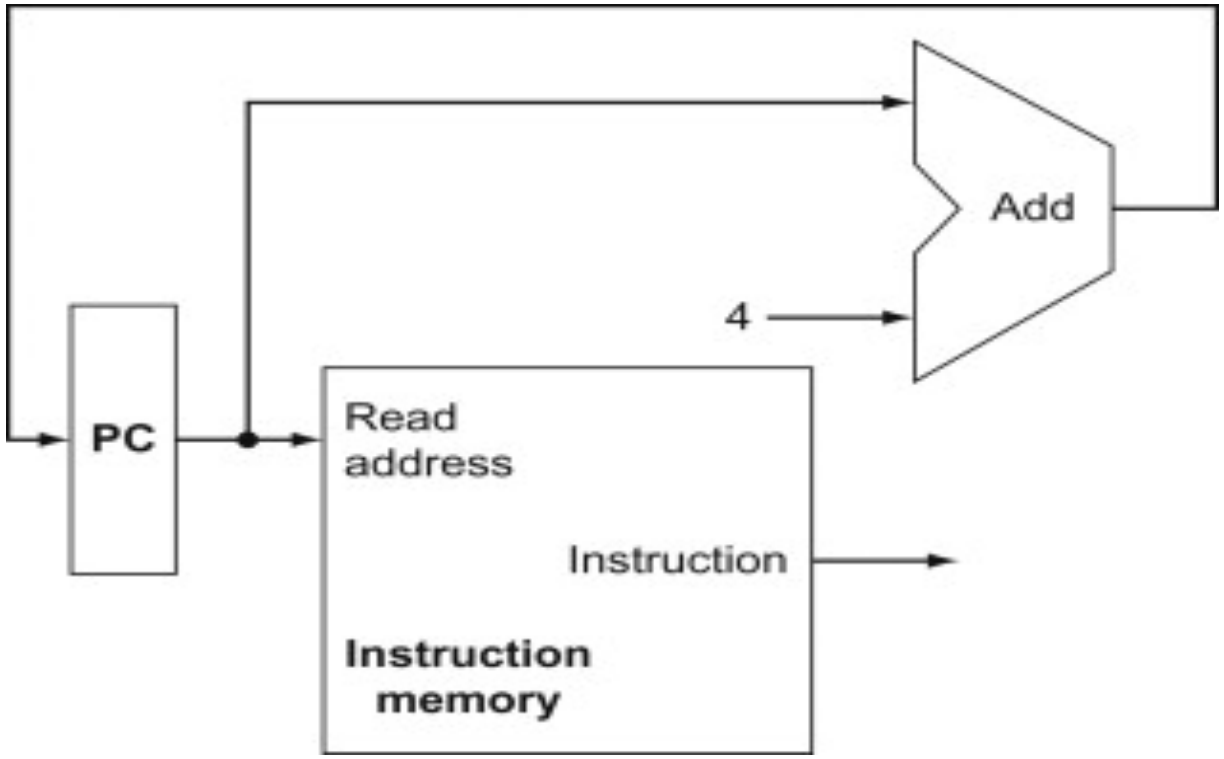


a. Instruction memory

b. Program counter

c. Adder

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

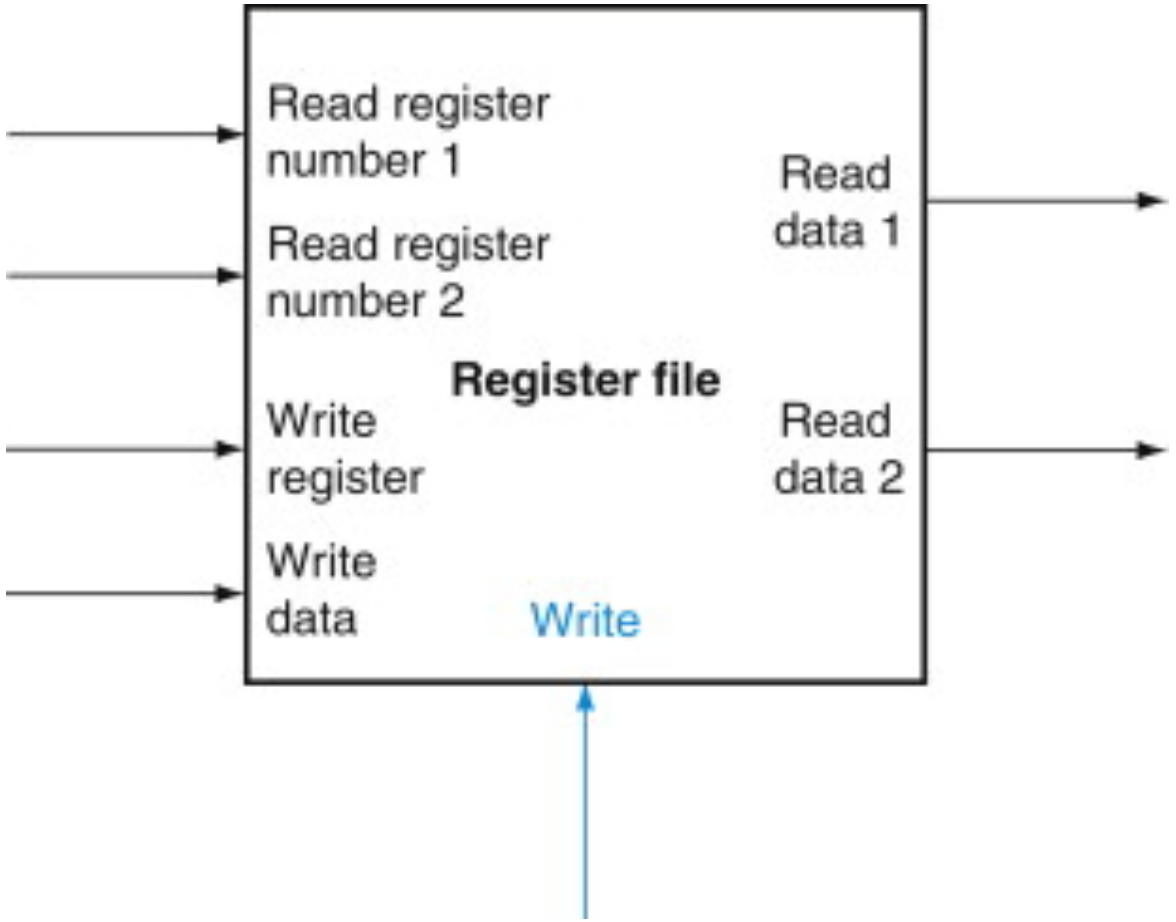


To execute any instruction, we must start by fetching the instruction from memory.

Register files

- A register file consists of a set of registers that can be read and written by supplying a register number to be accessed.
- A register file can be implemented with a decoder for each read or write port and an array of registers built from D flip-flops
- R-format instructions read two registers, perform an ALU operation on the contents of the registers, and write the result to a register.
- This instruction class includes add, sub, AND, OR, and slt
- R-format instructions have three register operands, so we will need to read two data words from the register file and write one data word into the register file for each instruction.

A register file with two read ports and one write port has five inputs and two outputs



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

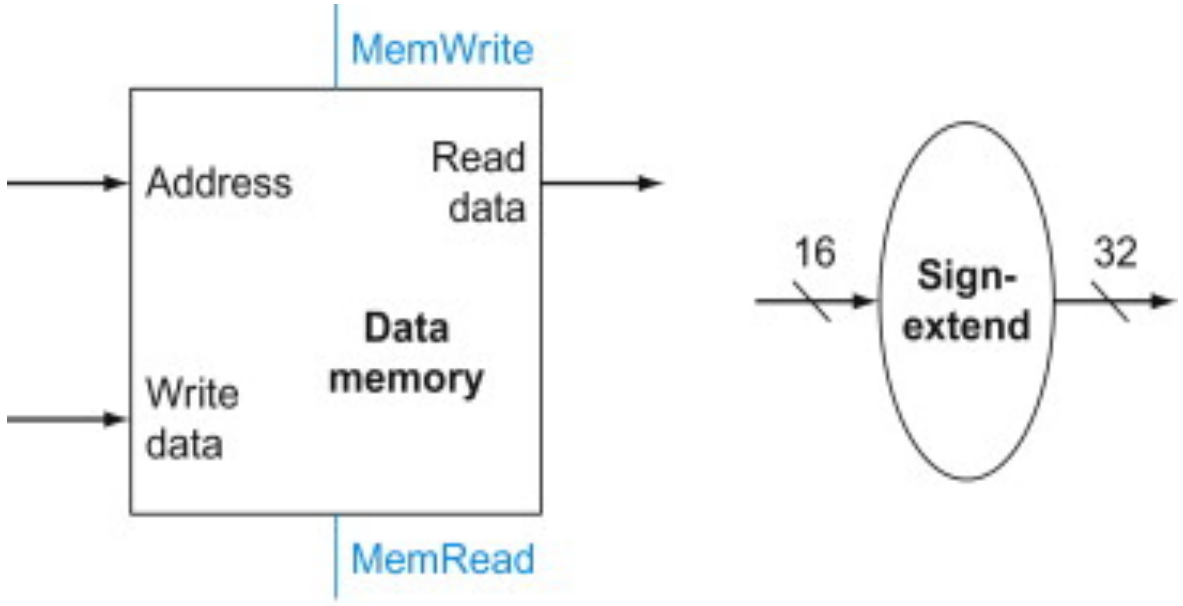
a. Registers

b. ALU

Sign-extend:

- To increase the size of a data item by replicating the high-order sign bit of the original data item in the high-order bits of the larger, destination data item.

Two units needed to implement loads and stores

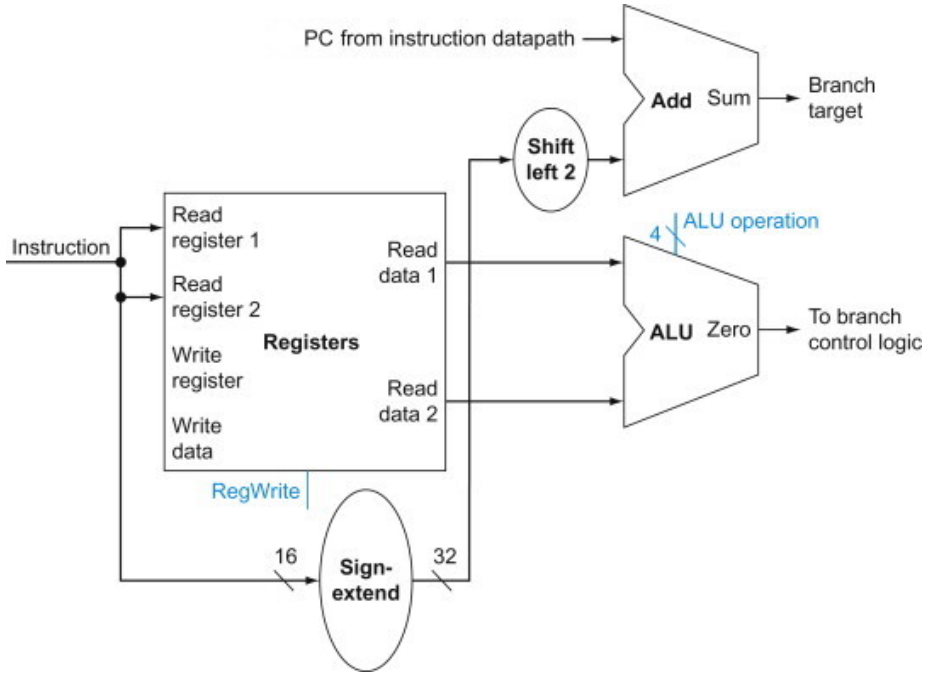


a. Data memory unit

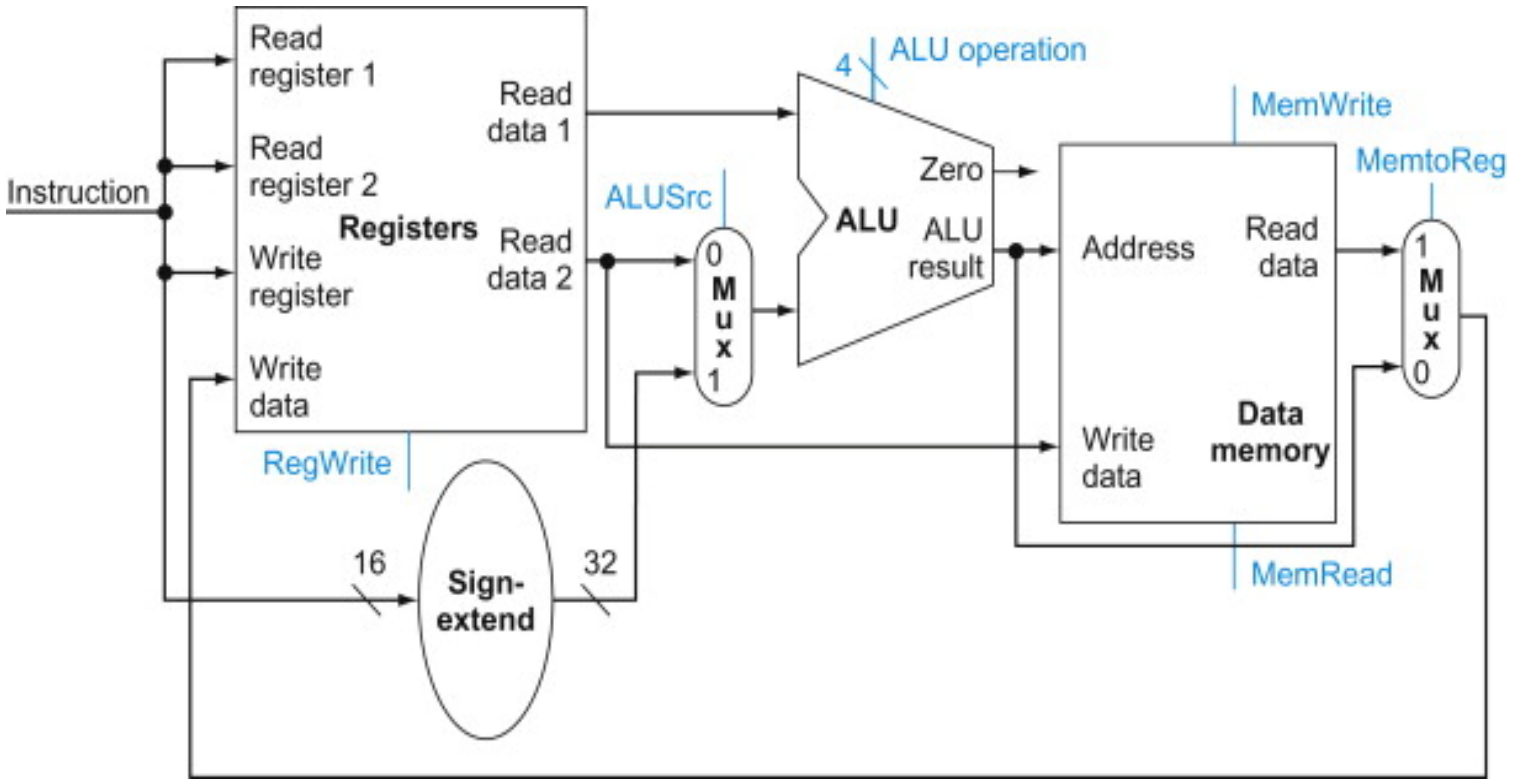
b. Sign extension unit

- ***Branch target address***: The address specified in a branch, which becomes the new program counter (PC) if the branch is taken. In the MIPS architecture the branch target is given by the sum of the offset field of the instruction and the address of the instruction following the branch.
- ***Branch taken***: A branch where the branch condition is satisfied and the program counter (PC) becomes the branch target. All unconditional branches are taken branches.
- ***Branch not taken*** or (***untaken branch***): A branch where the branch condition is false and the program counter (PC) becomes the address of the instruction that sequentially follows the branch.

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



The datapath for the memory instructions and the R-type instructions



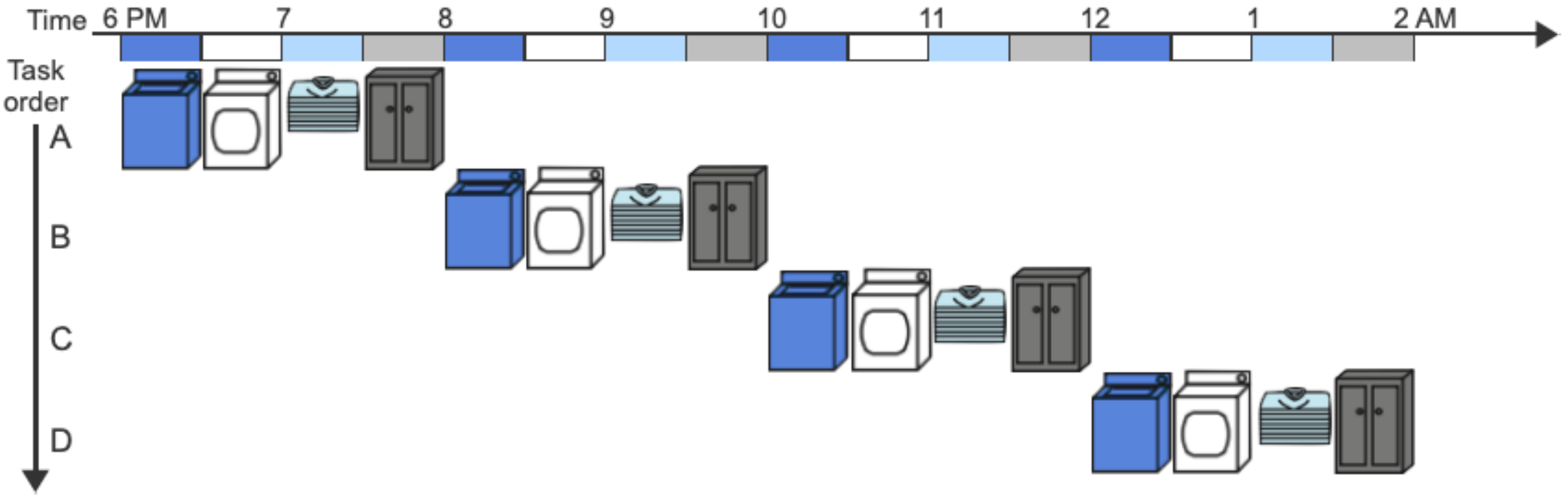
Pipelining

- ***Pipelining***: An implementation technique in which multiple instructions are overlapped in execution, much like an assembly line.

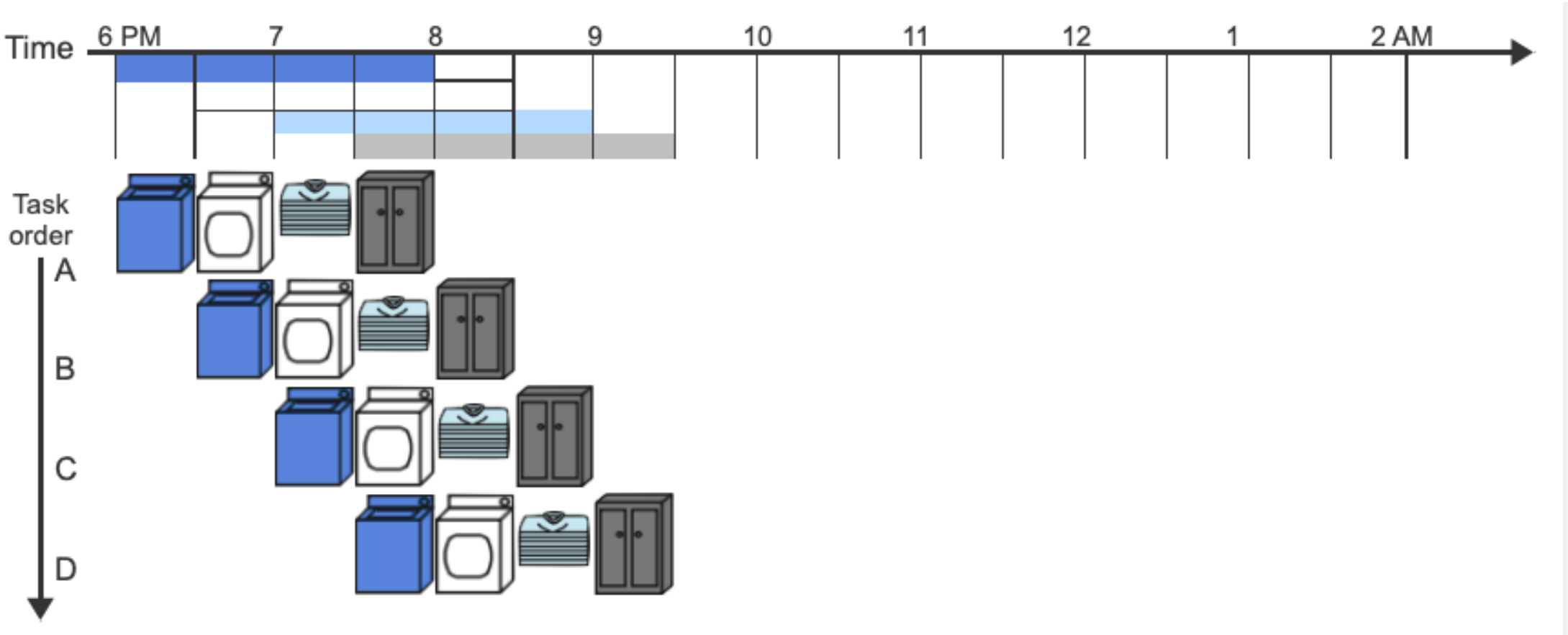
non-pipelined approach to laundry

1. Place one dirty load of clothes in the washer.
2. When the washer is finished, place the wet load in the dryer.
3. When the dryer is finished, place the dry load on a table and fold.
4. When folding is finished, ask your roommate to put the clothes away.

non-pipelined approach to laundry



pipelined approach to laundry



Pipelining in Processors

- The same principles apply to processors where we pipeline instruction-execution. MIPS instructions classically take five steps:
  1. Fetch instruction from memory.
  2. Read registers while decoding the instruction. The regular format of MIPS instructions allows reading and decoding to occur simultaneously.
  3. Execute the operation or calculate an address.
  4. Access an operand in data memory.
  5. Write the result into a register.

Total time for each instruction calculated from the time for each component

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `AND`, `OR`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Pipelining Speed-up Formula

Pipelining improves performance by *increasing instruction throughput*

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

Pipeline hazards

- These are situations in which the next instruction cannot execute in the following clock cycle
- We will look at:
  1. Structural hazard
  2. Data hazard
  3. Conttrol hazard

Structural hazard

- When a planned instruction cannot execute in the proper clock cycle because the hardware does not support the combination of instructions that are set to execute.

Data hazards

- Data hazards occur when the pipeline must be stalled because one step must wait for another to complete.
- Data hazard is also called a ***pipeline data hazard***.
- Data hazard is when a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available.
- Example:

```
add $s0, $t0, $t1
sub $t2, $s0, $t3
```

Forwarding:

- It is a method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer-visible registers or memory.
- Forwarding is also called **bypassing**

Data Hazard continued

- *Load-use data hazard is a* specific form of data hazard in which the data being loaded by a load instruction has not yet become available when it is needed by another instruction.
- Pipeline stall, which is also called bubble, is a stall initiated in order to resolve a hazard.

Data Hazard

- Does the following cause a data hazard for the 5-stage MIPS pipeline?

  i1: add $s0, $s1, $s2
  i2: add $s3, $s0, $s4

- The answer is yes

Data Hazard

- Does the following cause a data hazard for the 5-stage MIPS pipeline?

  i1: add $s0, $s1, $s2
  i2: add $s3, $s1, $s4

- The answer is No

Control Hazards

- **Control hazard**: Also called **branch hazard**.
- It is when the proper instruction cannot execute in the proper pipeline clock cycle because the instruction that was fetched is not the one that is needed; that is, the flow of instruction addresses is not what the pipeline expected.
- ***Branch prediction***: A method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome

Parallelism via instructions

- **Pipelining** exploits the potential **parallelism** among instructions
- ***Instruction-level parallelism***: The parallelism among instructions.
- ***Multiple issue***: A scheme whereby multiple instructions are launched in one clock cycle.

Two major ways to implement a multiple-issue processor

- ***Static multiple issue***: An approach to implementing a multiple-issue processor where many decisions are made by the compiler before execution.
- ***Dynamic multiple issue***: An approach to implementing a multiple-issue processor where many decisions are made during execution by the processor.

Reading

- Hennessy and Patterson Chapter 4.3, 4.4, 4.5, 4.7, 4.8 and 4.10