ECE 341

Lecture # 9

Instructor: Zeshan Chishti
zeshan@pdx.edu

October 27, 2014

Portland State University

Lecture Topics

- Basic Processing Unit
  - Fundamental Concepts
    - Instruction Processing Steps
    - Basic Processing Hardware
    - RISC Processors
  - Instruction Processing in a RISC processor
    - Load Instructions
    - Arithmetic and Logic Instructions
    - Store Instructions
  - Hardware Components
    - Register File
    - ALU

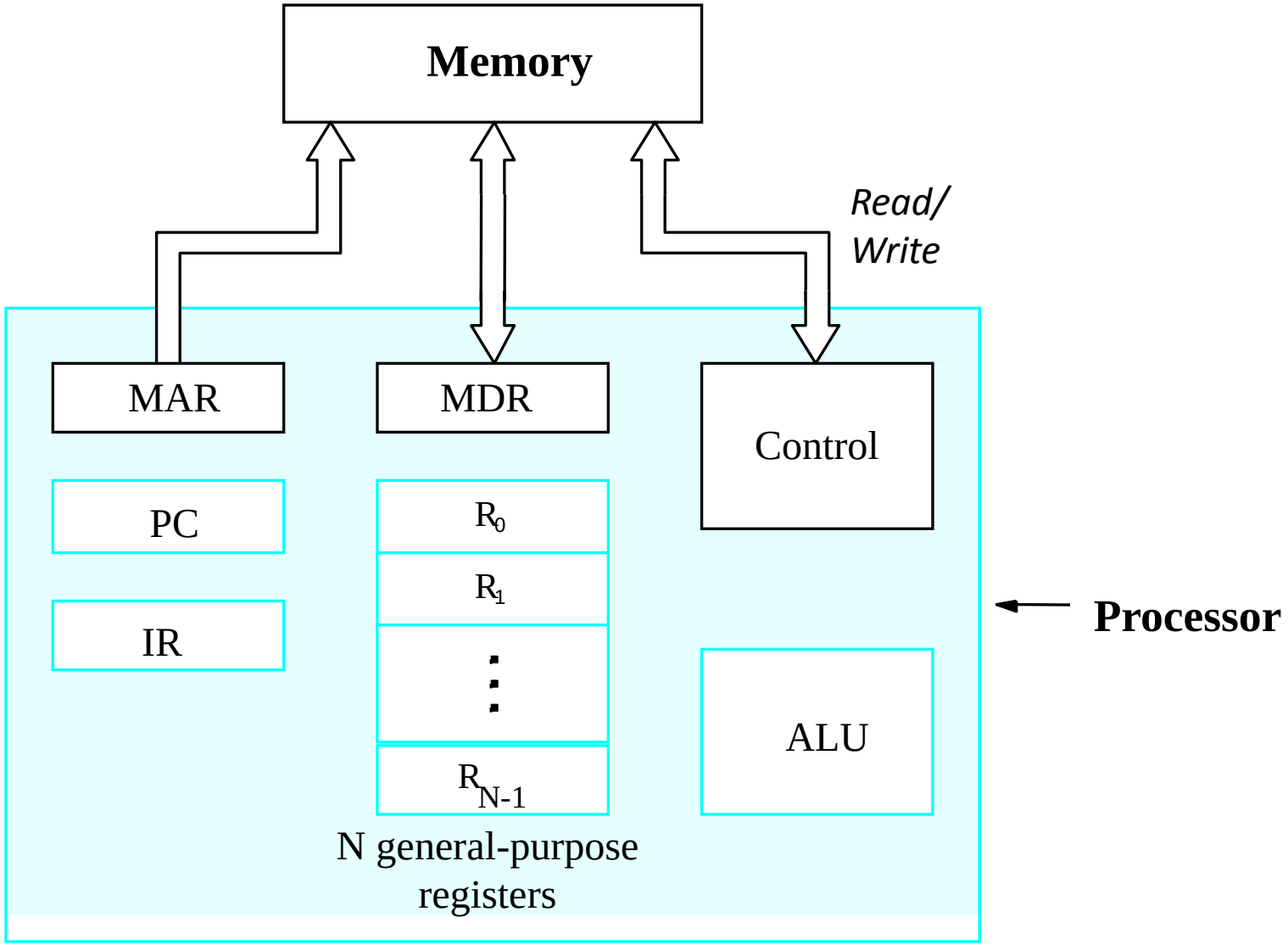
- Reference:
  - Chapter 5: Sections 5.1, 5.2 and 5.3

Overview

- A typical computing task consists of a a sequence of *instructions* that constitute a *program*
- Processing Unit (*processor*) carries out these instructions
- An instruction is *processed* by performing a sequence of more rudimentary operations, such as:
  - *Fetching* the instruction from memory
  - *Decoding* the instruction to understand its semantics
  - *Executing* the instruction
- Different processor architectures differ in the way they perform the above operations

We'll focus on architectural principles and hardware components that are common to (almost) all the existing processors

Processor Organization



Processor Registers

- Instruction register (IR)
  - Holds the instruction that is currently being executed
- Program counter (PC)
  - Contains address of the next instruction to be fetched/executed
- General-purpose register ($R_0 – R_{n-1}$)
  - Hold operands that are currently being (or soon to be) processed
- Memory address register (MAR)
  - Contains address of the word being read/written from memory
- Memory data register (MDR)
  - Contains contents of the data word being read/written from memory

Fundamental Concepts

- Processor fetches Instructions from successive memory addresses until a branch/jump instruction is encountered
- Processor keeps track of the address of memory location containing the next instruction in a register called **program counter (*PC*)**
- After every instruction fetch, the **instruction address generator** updates the contents of *PC* to point to the next instruction
- While being processed, the instruction is kept in **instruction register (*IR*)**. *IR* holds the instruction until its execution is completed
- **Control circuitry** decodes (interprets) the instruction in *IR* and orchestrates the necessary control signals and inputs needed to execute the instruction

Fundamental Concepts (cont.)

- **Register file** is a fast memory whose storage locations are organized to form the processor's general purpose registers
- The contents of the registers named in an instruction are sent to the **Arithmetic and Logic Unit (ALU)**, which executes the instruction
- The results computed by the ALU are stored in a register in the register file
- The processor communicates with the memory through the **processor memory interface**

ISA and Register Transfer Notation

Instruction set Architecture (ISA) specifies the operations that a computer is able to perform. ISA must include four types of operations:

1. Data transfers between memory and processor registers (load, store)
2. Arithmetic and logic operations on data (add, mult, and, or etc.)
3. Program sequencing and control (branch, jump)
4. I/O transfers

Register Transfer Notation (RTN) describes how each operation transfers data

- The contents of a location are denoted by placing square brackets around its name
- Right hand side of a RTN expression always denotes a value
- Left hand side of a RTN is the name of a location where the value is to be placed
- Examples:
  - R2 ← [LOC] means that the contents of memory location LOC are transferred into register R2
  - R4 ← [R2] + [R3] means that the contents of registers R2 and R3 are added and the result is transferred into the register R4

Instruction Processing

Carrying out an instruction in any processor involves two distinct phases:

Instruction Fetch Phase:

- Fetch the contents of memory location pointed to by PC. Load the instruction returned from memory into IR:

$$IR \leftarrow [[PC]]$$

- Increment the PC to point to the next instruction:

$$PC \leftarrow [PC] + 4$$

Instruction Execution Phase:

- Carry out the operation specified by the instruction in the IR. This involves performing one or more of the following steps:
  - Read the contents of a given memory location into a register
  - Read data from one or more processor registers
  - Perform an arithmetic/logic operation and place the result into a register
  - Store data from a register into a given memory location

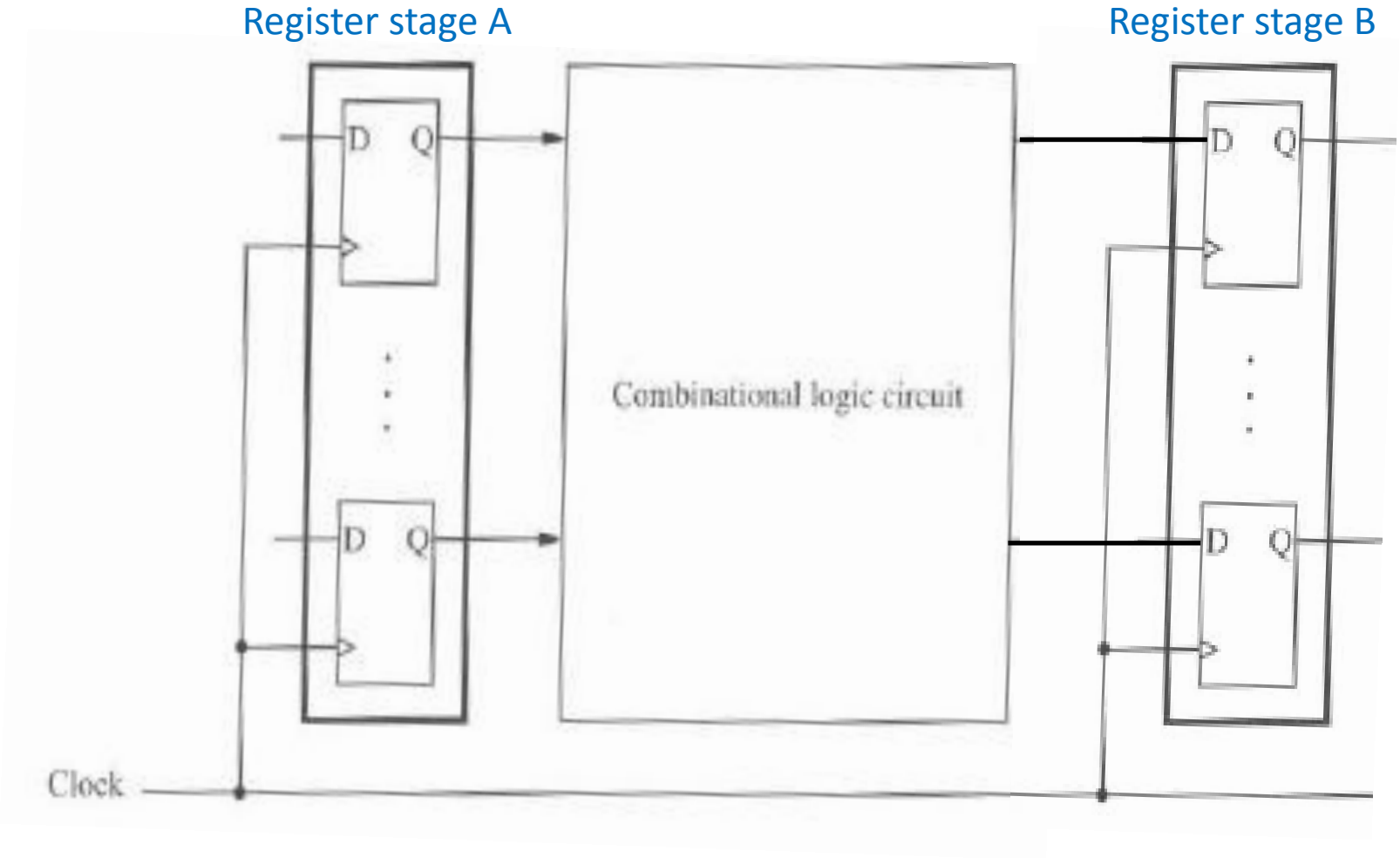
Typical Instruction Processing Steps

Initial State: PC is set to point to the first instruction

Instruction steps executed in a loop

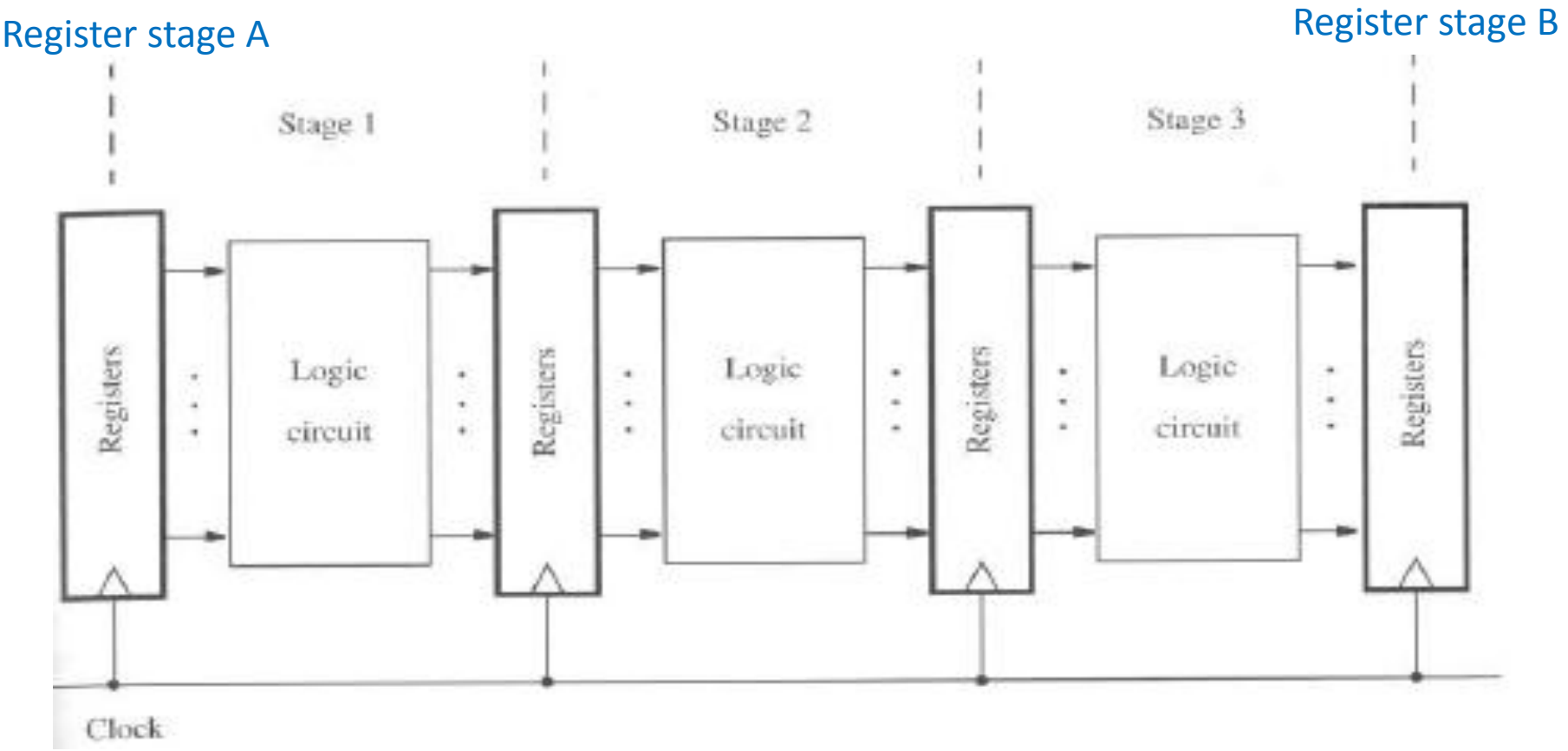
- **Instruction Fetch:**
  - MAR ← [PC], *Read* signal sent to memory
  - Instruction read out from memory and loaded into MDR
  - IR ← [MDR], PC ← [PC] + 4
- **Instruction Decode:**
  - Instruction in IR decoded by control logic, instruction type and operands determined
  - Source operand registers read from general purpose register file
- **Execute:**
  - ALU performs the desired arithmetic operation for arithmetic instructions
  - ALU performs address calculation for load/store instructions
- **Memory Read:**
  - For load instrs., data read from memory (address sent to MAR, *Read* asserted)
- **Writeback:**
  - Results sent back to
    - Destination register, or
    - Memory, in case of store instrs., (address sent to MAR, data to MDR, assert *Write*)

Basic Processing Hardware



Clock period must be longer than the delay of the combinational logic circuit

Multi-stage Processing Hardware



- Combinational circuit broken down into multiple stages, each taking one clock cycle
- If there are *n* stages, clock period is 1/*n* of the clock period on previous slide
- Multiple instructions may proceed in parallel in different stages (**pipelining**)
- Inter-stage registers hold a stage's result, so they can be used as next stage's inputs

RISC Architectures

Reduced Instruction Set Computers (RISC) have two key characteristics:

- Each instructions fits in exactly one memory *word*
- A *load/store* architecture is used
  - Memory operands are accessed using **only** Load and Store instructions
  - All operands in an arithmetic/logic operation **must** either be in processor registers, or one operand may be specified explicitly (*immediate* operand) within the instruction word

RISC instructions have a regular structure => simpler to implement in a multi-stage fashion and amenable to *pipelined* operation

RISC Example

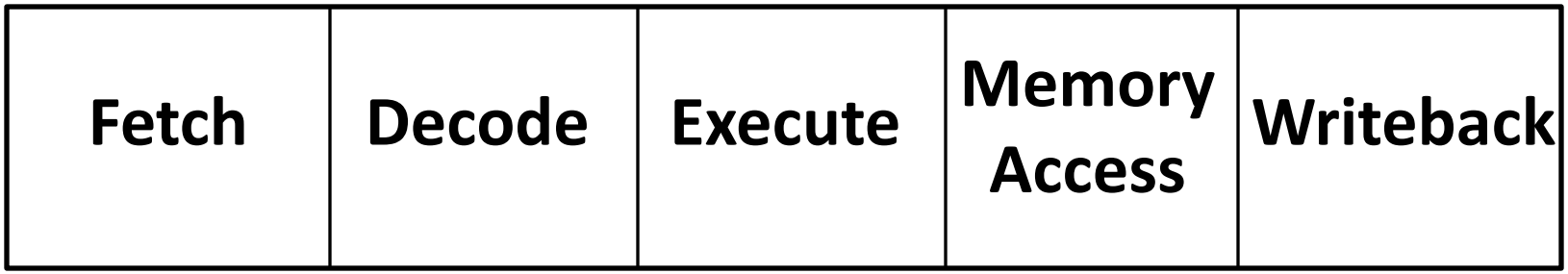
Objective:

- Add the contents of memory words at addresses 4 and 36 and store the result in register R5

Program: Assuming that register R0 has the contents “0”

Add R1, R0, #4	R1←[R0]+4
Add R2, R0, #36	R2←[R0]+36
Load R3, 0(R1)	R3←[[R1]]
Load R4, 0(R2)	R4←[[R2]]
Add R5, R3, R4	R5←[R3]+[R4]

5-stage RISC Processor



FETCH: Fetch an instruction and increment the PC

DECODE: Decode the instruction (determine instruction type, source & destination registers) and read source registers from register file

EXECUTE: Perform an ALU operation

MEMORY ACCESS: Read or write memory data if the instruction involves a memory operation

WRITEBACK: Write the result into the destination register, if needed

Load Instructions

Example

- Load R5, X(R7) R5←[[R7]+X]

Source register: R7 Destination register: R5

The immediate value X is given in the instruction word

Instruction steps:

- Fetch: Fetch the instruction and increment the program counter
- Decode: Decode the instruction in IR to determine the operation to be performed (load). Read the contents of register R7.
- Execute: Add the immediate value X to the contents of R7
- Memory Access: Use the sum X+[R7] as the effective address of the source operand, read the contents of that location from memory
- Writeback: Load the data received from memory into register R5

Arithmetic and Logic Instructions

Example with *only* register operands

- Add R3, R4, R5 R3←[R4]+[R5]

Source registers: R4, R5 Destination register: R3

Instruction steps:

- Fetch: Fetch the instruction and increment the program counter
- Decode: Decode the instruction in IR to determine the operation to be performed (add). Read the contents of registers R4 and R5
- Execute: Compute the sum [R4] + [R5]
- Memory Access: No action, since there are no memory operands
- Writeback: Load the result into register R3

Arithmetic and Logic Instructions

Example with *immediate* operand:

- Add R3, R4, #1000 R3←[R4]+1000

Source register: R4 Destination register: R3

The immediate value is given in the instruction word

Instruction steps:

- Fetch: Fetch the instruction and increment the program counter
- Decode: Decode the instruction in IR to determine the operation to be performed (add immediate). Read the register R4
- Execute: Compute the sum [R4] + 1000
- Memory Access: No action, since there are no memory operands
- Writeback: Load the result into register R3

Store Instructions

Example

- Store R6, X(R8) X+[R8]←[R6]

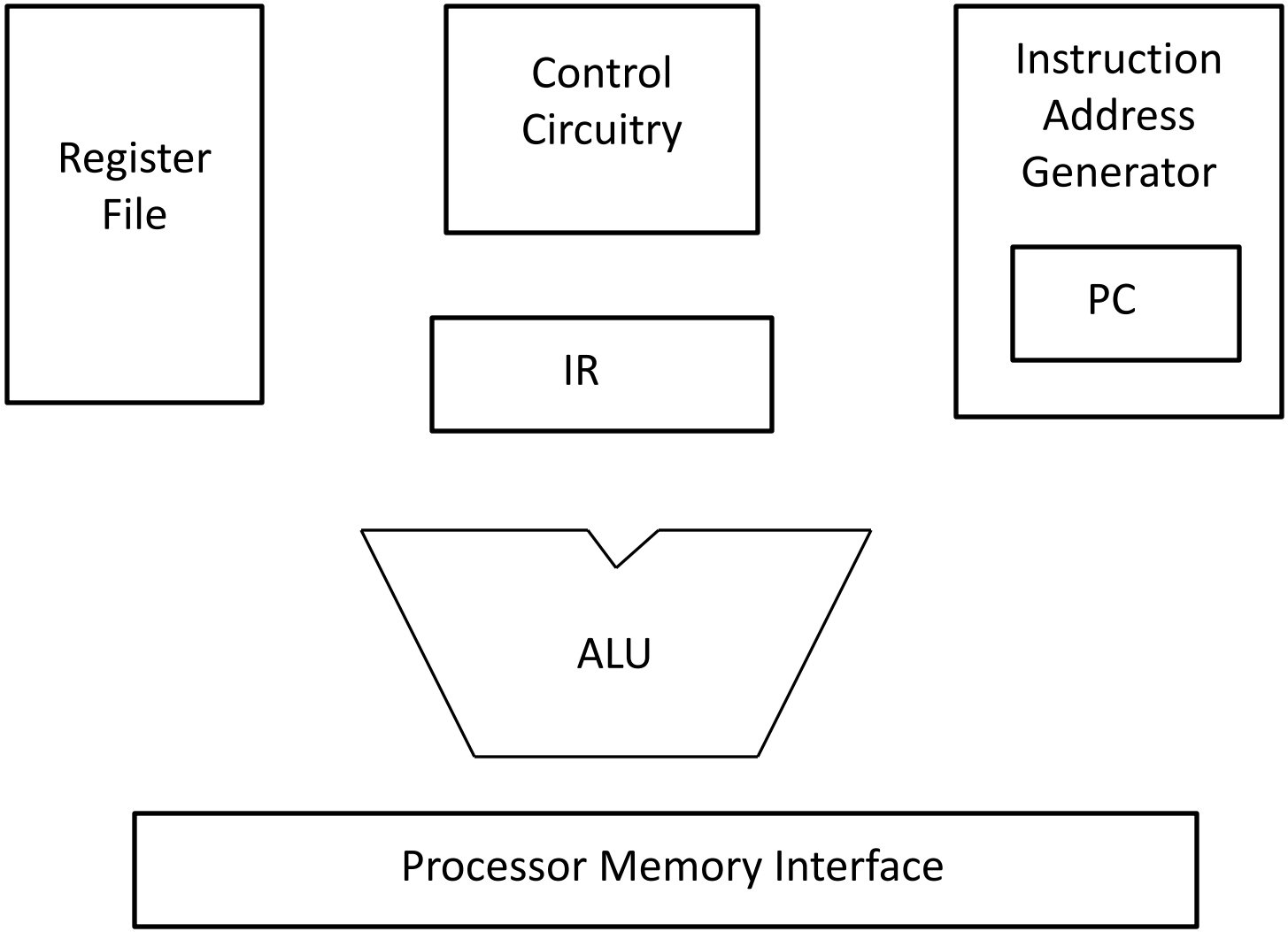
Source registers: R6, R8 Destination register: None

The immediate value X is given in the instruction word

Instruction steps:

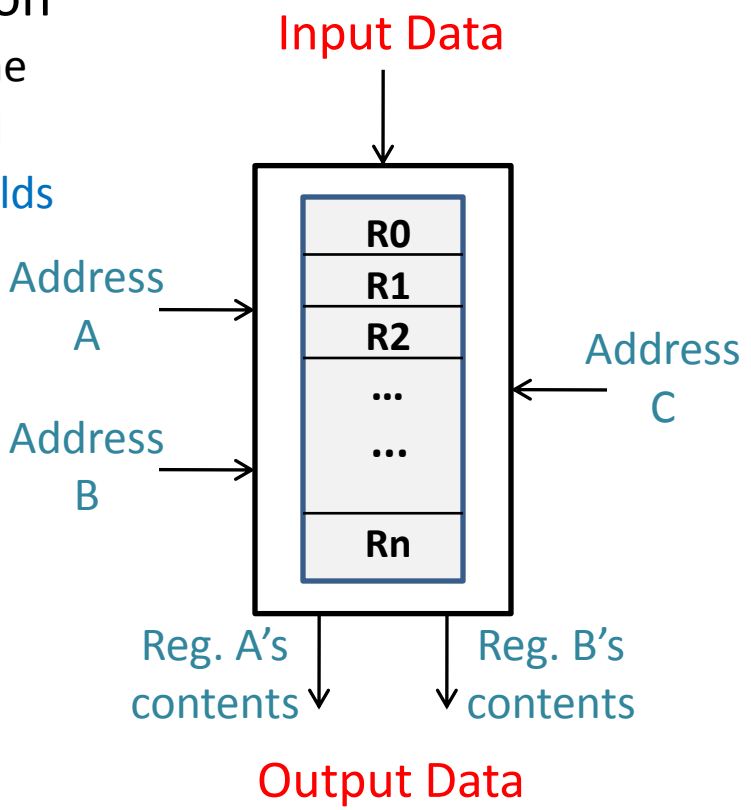
- Fetch: Fetch the instruction and increment the program counter
- Decode: Decode the instruction in IR to determine the operation to be performed (store). Read the contents of registers R6 and R8.
- Execute: Compute the effective address X + [R8]
- Memory Access: Store the contents of register R6 into memory location X + [R8]
- Writeback: No action

Hardware Components of a Processor



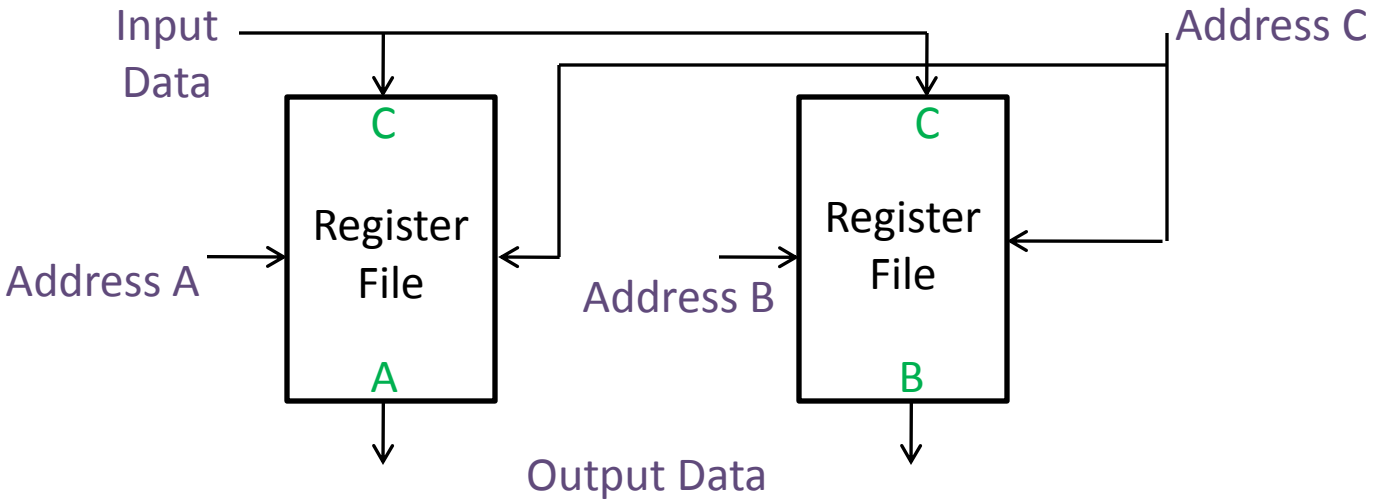
Register File

- General purpose registers are usually implemented as a *register file*
- Register file contains
  - Array of storage elements to hold register contents
  - Access circuitry to read/write data from/into any register
- To support two source operands per instruction
  - Access circuitry able to read 2 registers at same time
  - Address inputs *A* & *B* select the registers to be read
  - Address inputs connected to IR's source register fields
  - Register contents available at separate outputs
- To support a destination operand
  - Address input *C* selects the register to be written
  - *C* is connected to IR's destination register field
  - Data input used to specify the data to be written
- For example, for instruction add R3, R4, R5
  - A = 4, B = 5 and C = 3



Register File (cont.)

- A memory unit with two output ports is said to be *dual ported*
- Two ways to implement a **dual-ported register file**
- **True ports:** Single set of registers with duplicate data paths and access circuitry that enables two registers to be read at a time
- **Two copies:** Use 2 memory blocks each containing one copy of the register file
  - To read two registers, one register can be accessed from each file
  - To write a register, data needs to be written to both the copies of that register



ALU

- Arithmetic and Logic Unit (ALU) performs:
  - Arithmetic operations (adition, subtraction, multiplication etc.)
  - Logic operations (bitwise AND, OR, XOR etc.)

- Register file and ALU are connected with each other so that
  - Source operands of an instruction, after being read from the register file are directly fed into the ALU
  - Result computed by the ALU can be loaded into the destination register specified by the instruction

ALU + Register File Connection

- Source register *A* connected to *InA* input of ALU
- Multiplexer MuxB connected to *InB* input of ALU and the *immediate value* field in IR
- MuxB's select input depends on the instruction being executed
  - MuxB selects *InB*, for operations that require only register operands
  - MuxB selects *immediate value* for operations that require an immediate operand