ECE 341

Lecture # 10

Instructor: Zeshan Chishti
zeshan@ece.pdx.edu

October 29, 2014

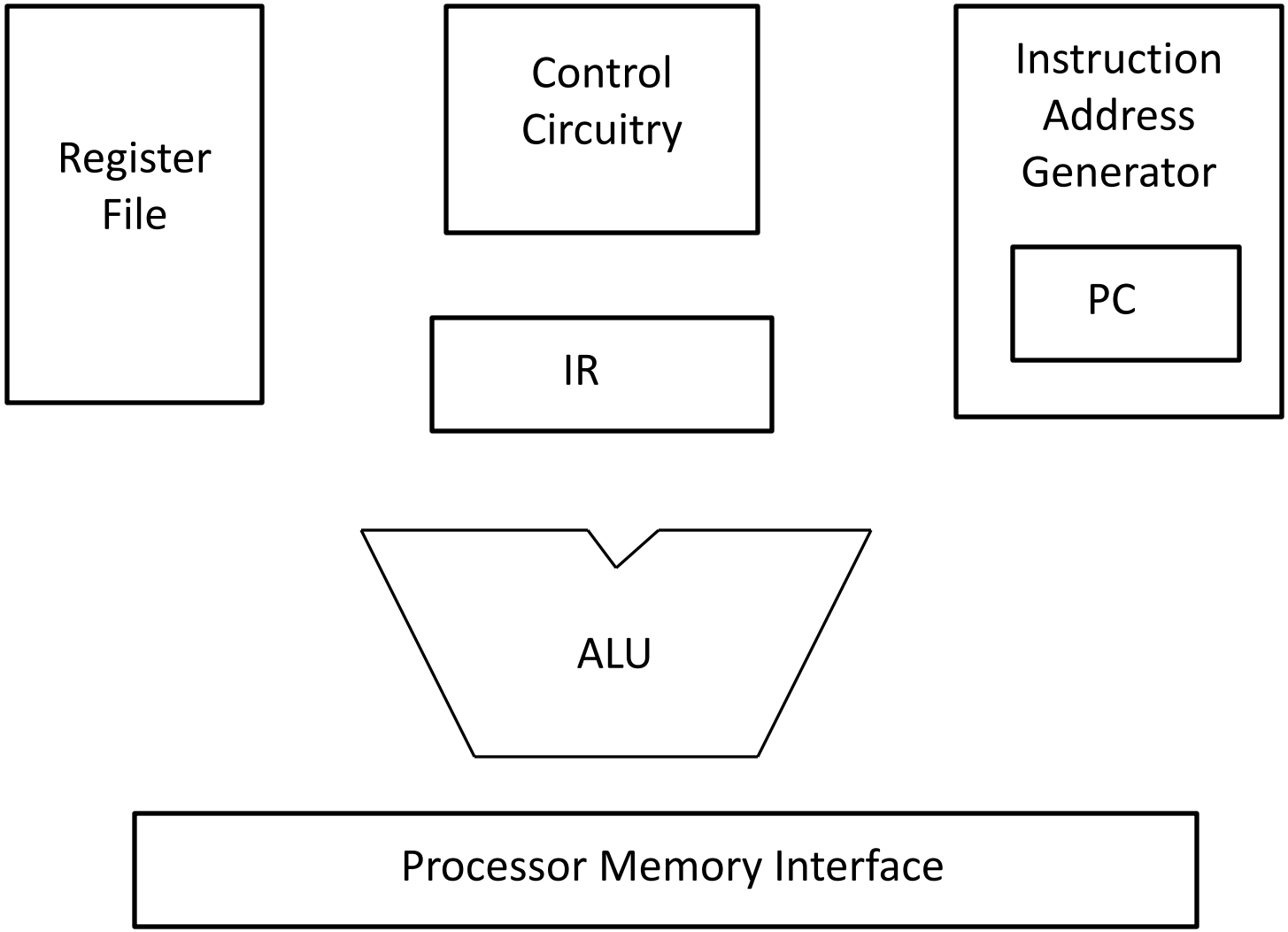
Portland State University

Lecture Topics

- Basic Processing Unit
  - Hardware Components
    - Register File
    - ALU
    - Datapath
    - Instruction Fetch Section
  - Instruction Fetch and Execution Steps
    - Instruction Encoding
    - Examples

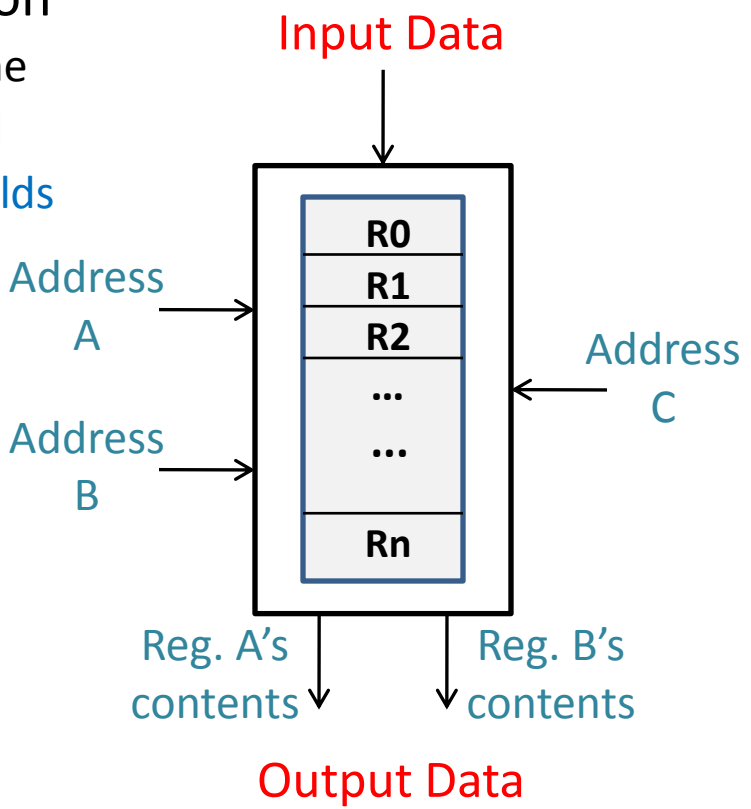
- Reference:
  - Chapter 5: Sections 5.3 and 5.4 (Pages 161-168 of textbook)

Hardware Components of a Processor



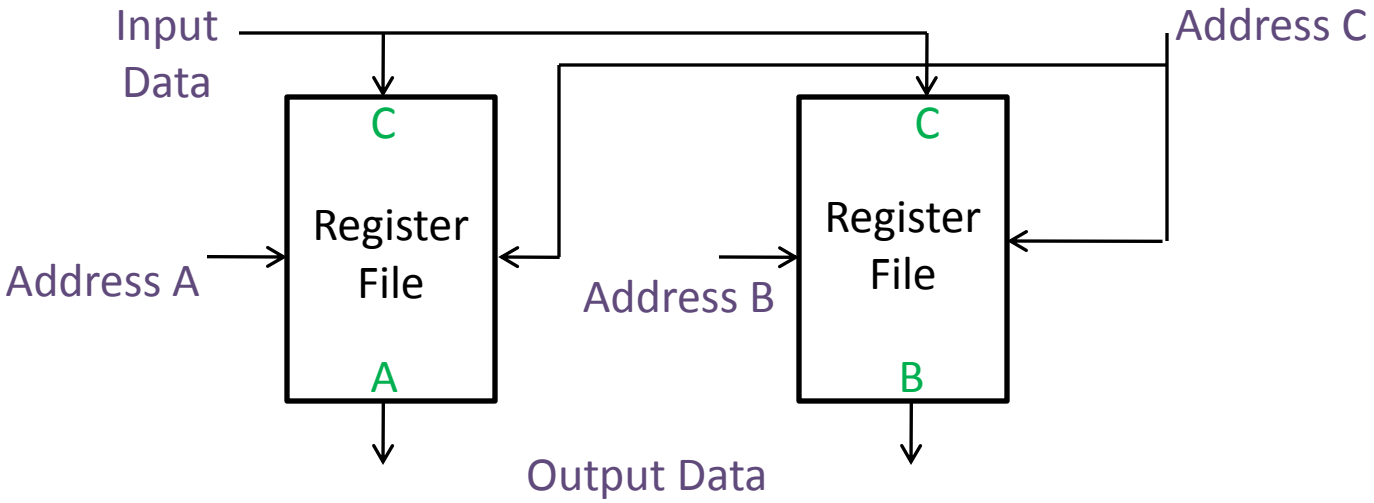
Register File

- 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



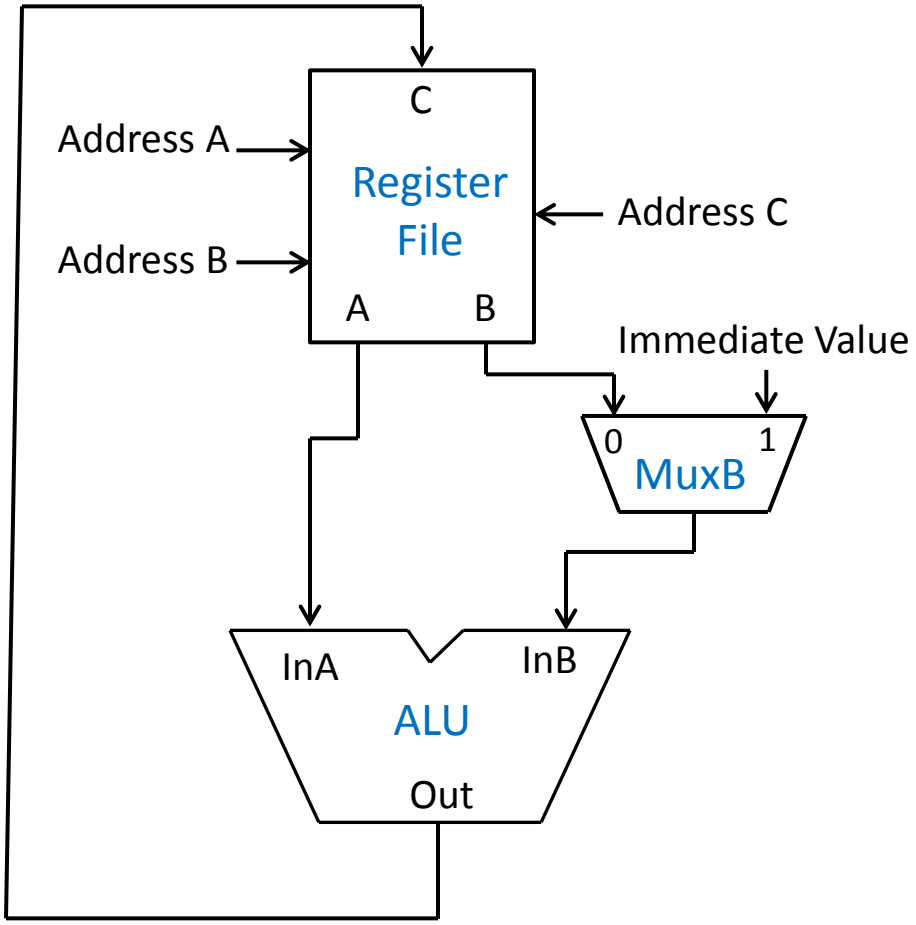
ALU

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

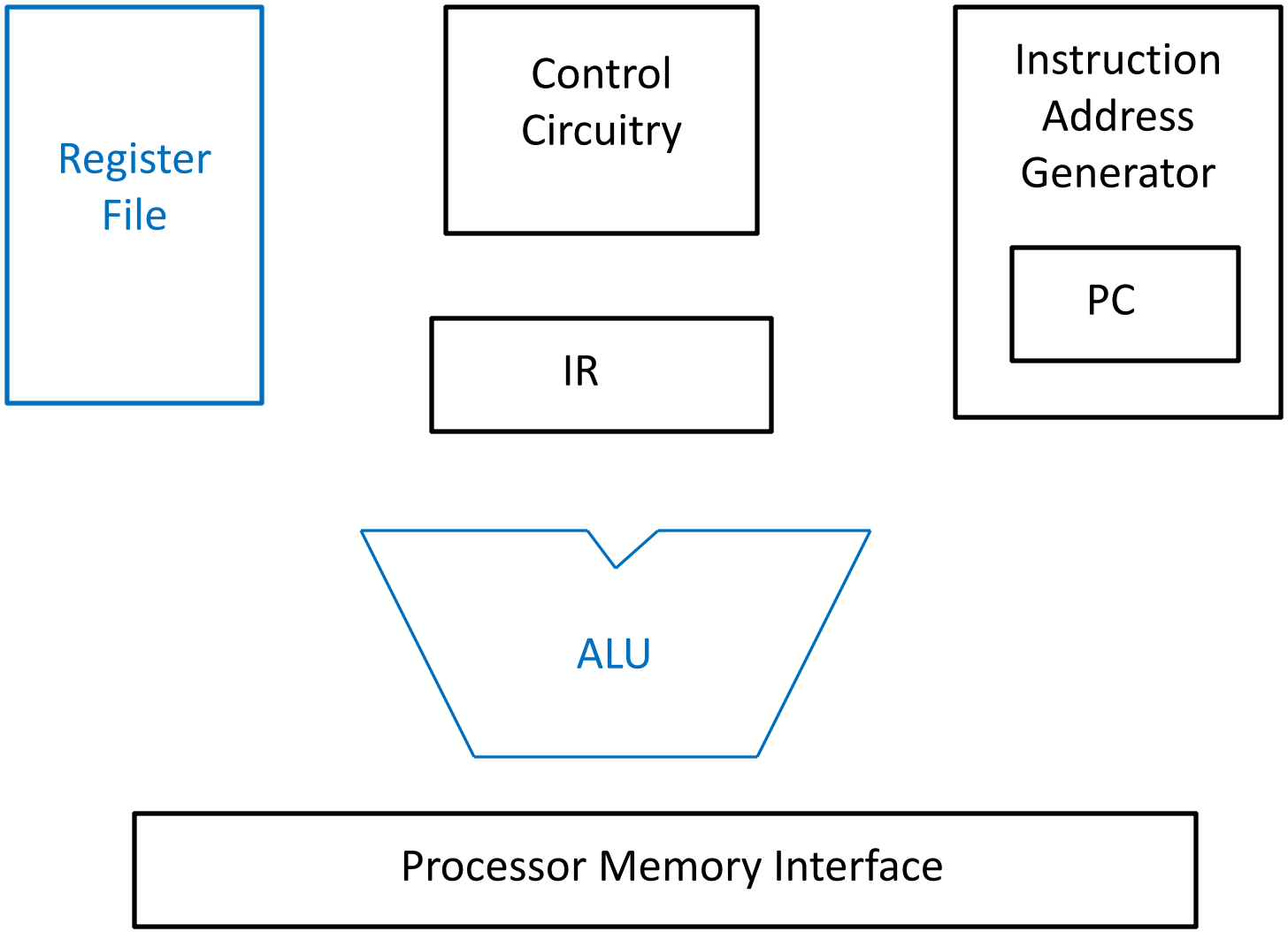
- Register file and ALU are connected with each other so that
  - Source operands of an instruction, after being read from the register file are directly fed into the ALU
  - Result computed by the ALU can be loaded into the destination register specified by the instruction

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

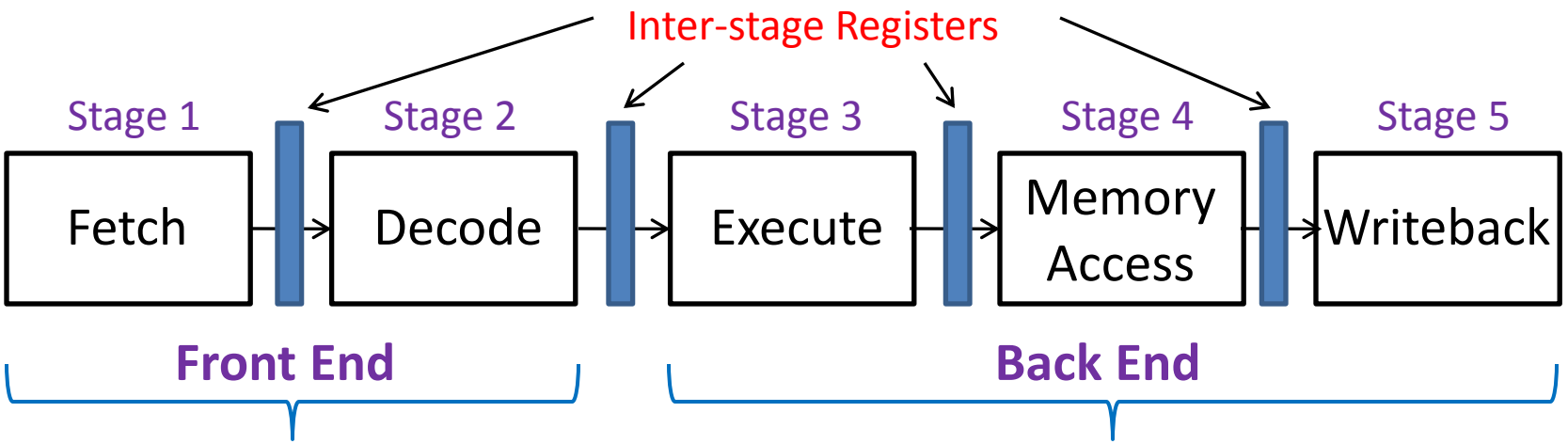


Hardware Components of a Processor



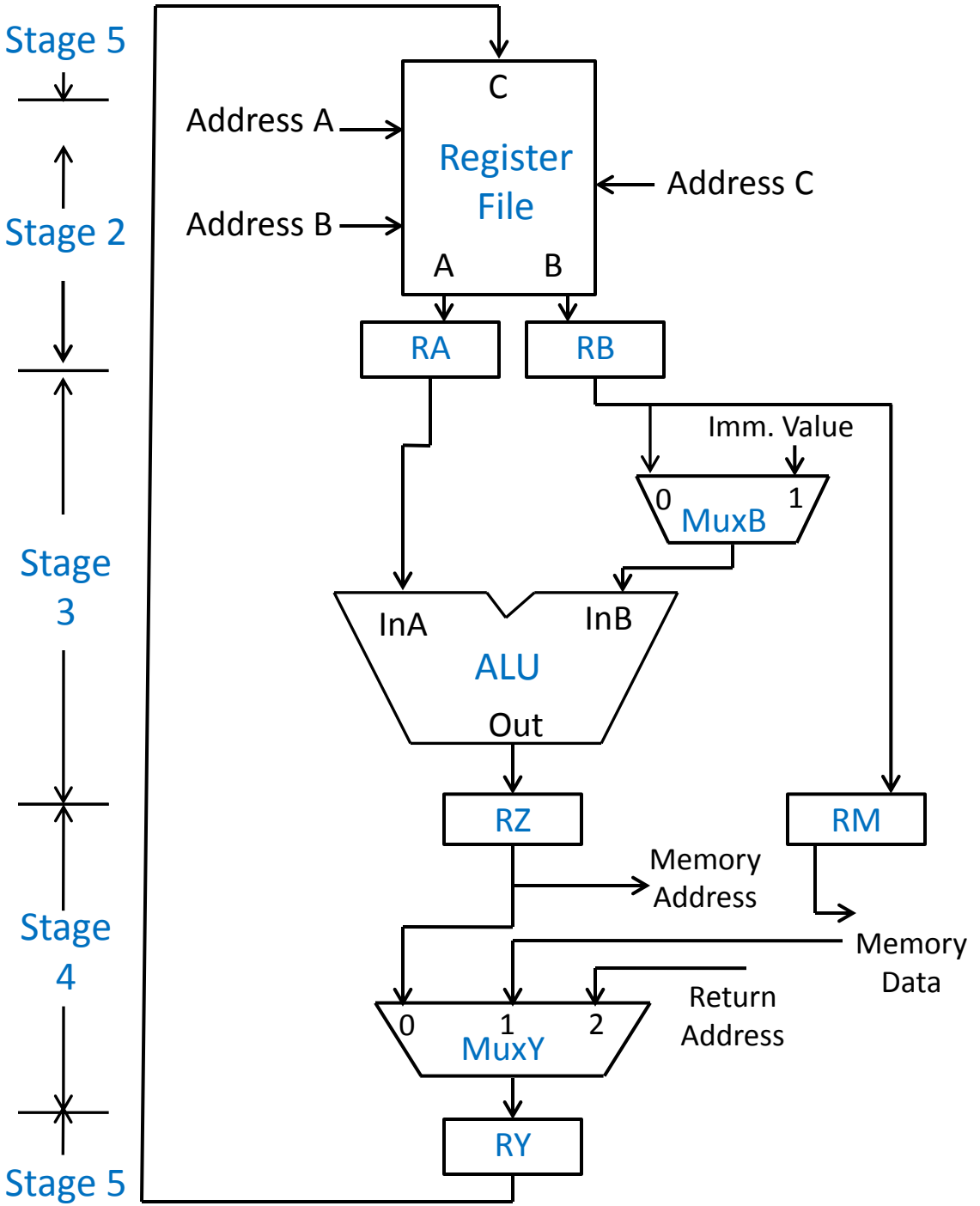
Datapath

- Recall that instruction processing consists of *fetch phase* & *execution phase*
- Processor hardware divided into two corresponding sections:
  - **Front end (fetch section)**:
    - Includes the fetch and decode stages
  - **Back end (execute section)**:
    - Includes the ALU, Memory Access and Writeback stages
- **Datapath** is a combination of register read, back-end stages and inter-stage registers



RA, RB, RZ, RM and RY are **inter-stage registers**

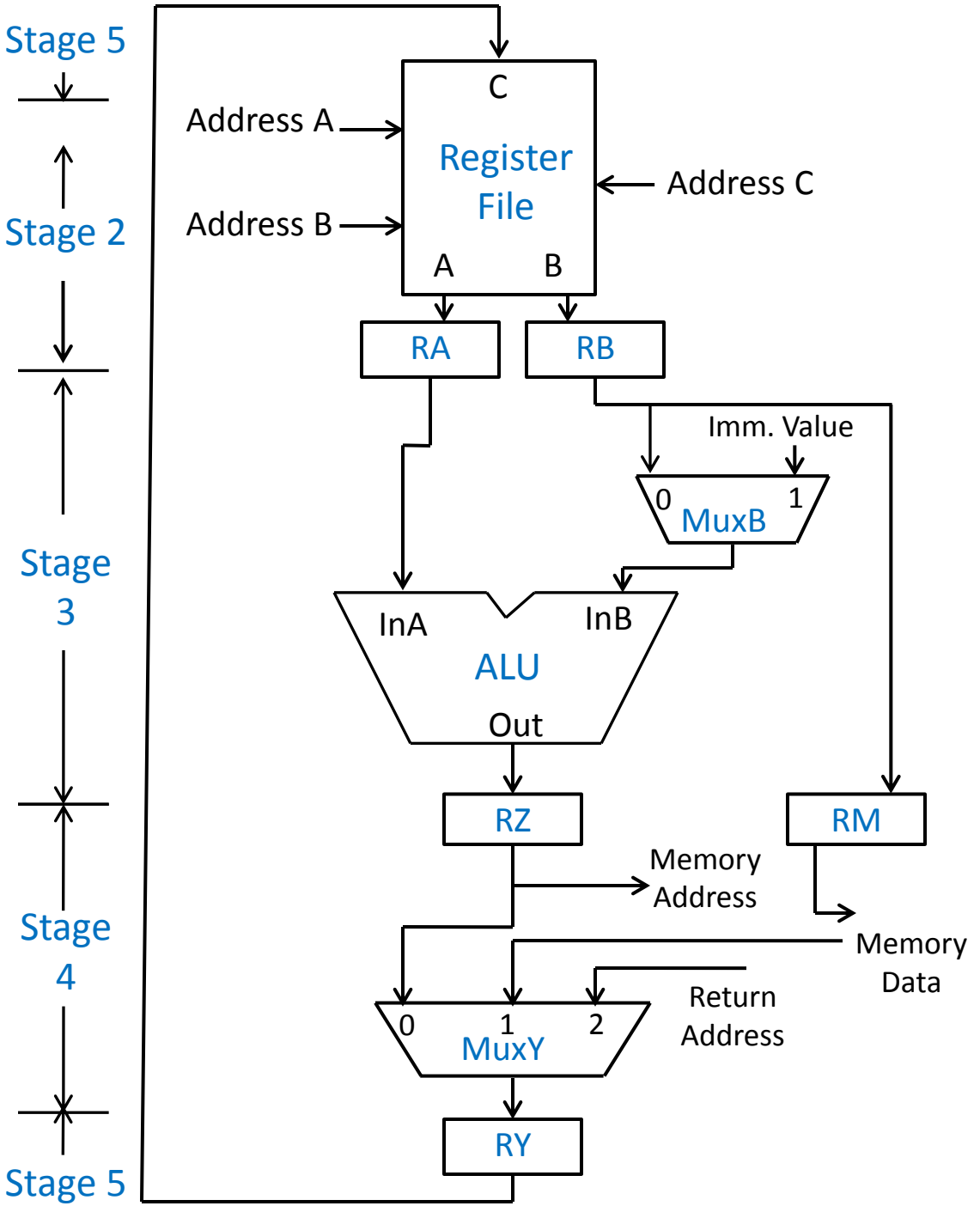
Register file is in both stage 2 and stage 5, because stage 2 (Decode) reads source registers and stage 5 (Writeback) writes into destination registers



Arithmetic/Logic Instructions:

Instruction Type	MuxB selection	MuxY selection
1 source register, 1 immediate operand	1	0
2 source registers	0	0

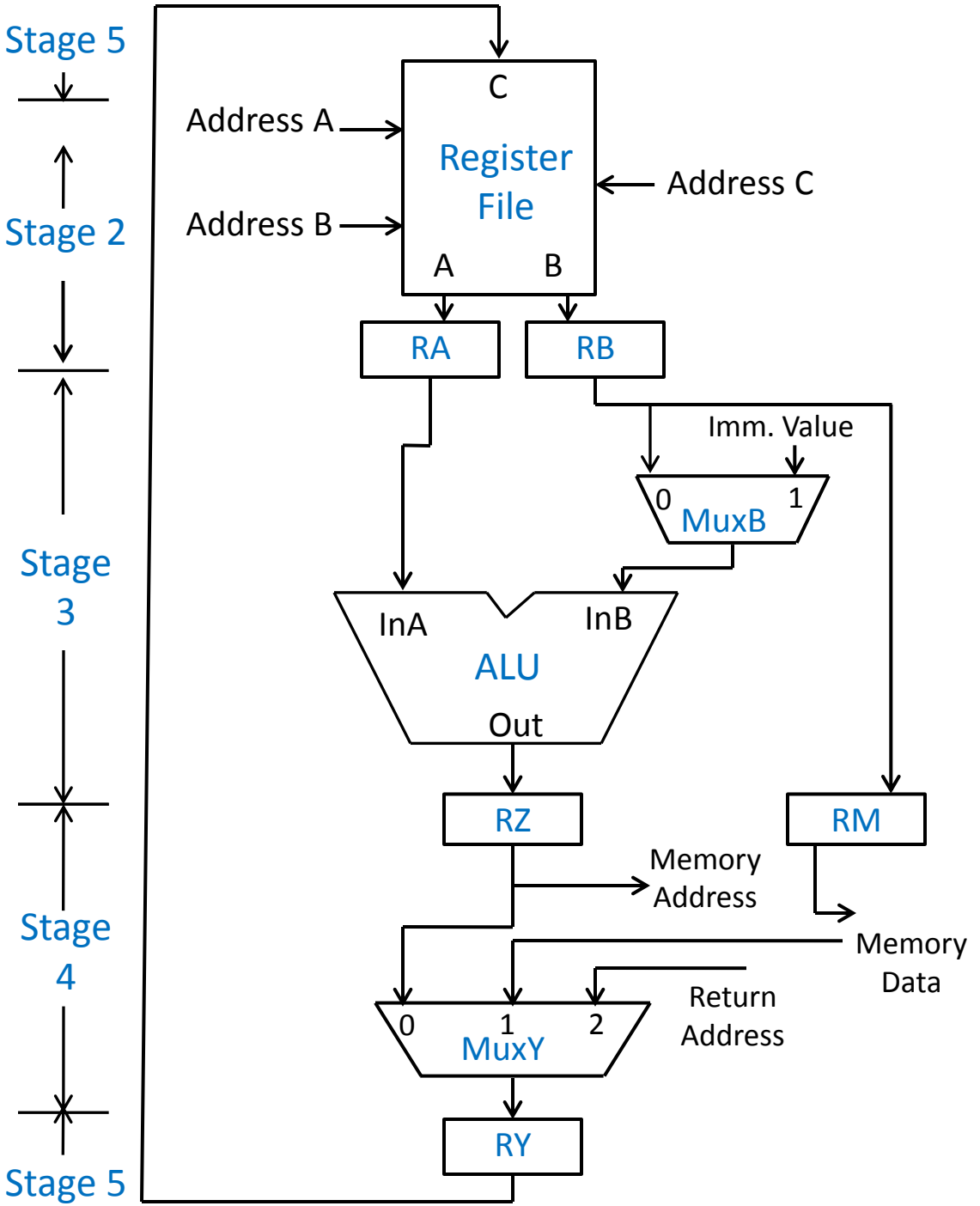
- Stage 3: ALU carries out the desired arithmetic/logic operation on the source operands , result is loaded into RZ
- Stage 4: Result computed in stage-3 moves from RZ to RY
- Stage 5: Data transferred from RY into destination



Load Instructions:

Instruction Type	MuxB selection	MuxY selection
Load	1	1

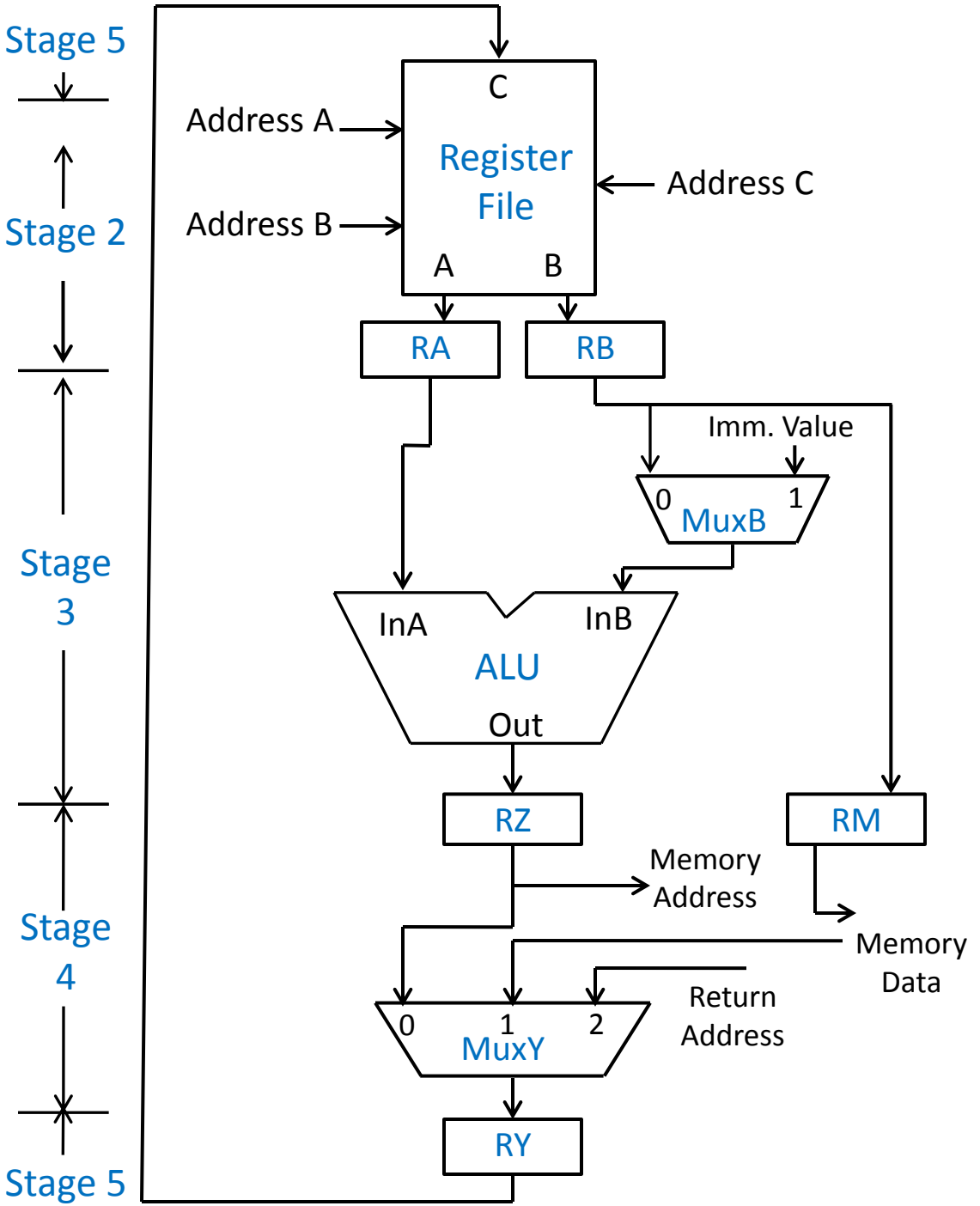
- Stage 3: Effective address computed and loaded into RZ
- Stage 4: Address sent from RZ to memory, data returned by memory loaded into RY
- Stage 5: Data transferred from RY into destination



Store Instruction:

Instruction Type	MuxB selection	MuxY selection
Store	1	x

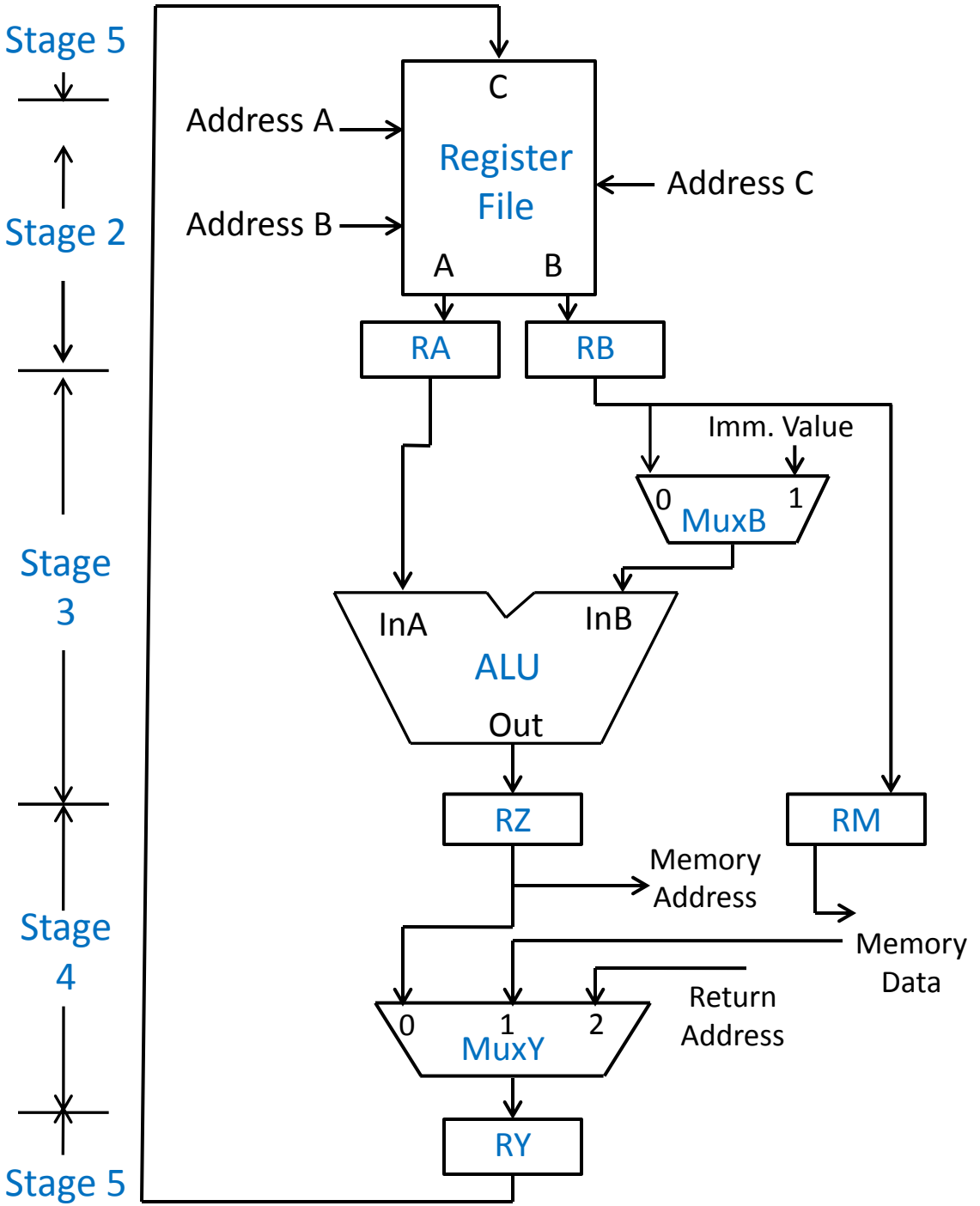
- Stage 3: Effective address computed and loaded into RZ, data to be stored is moved from RB to RM
- Stage 4: Address sent from RZ to memory, data sent from RM to memory, write signal asserted
- Stage 5: No action



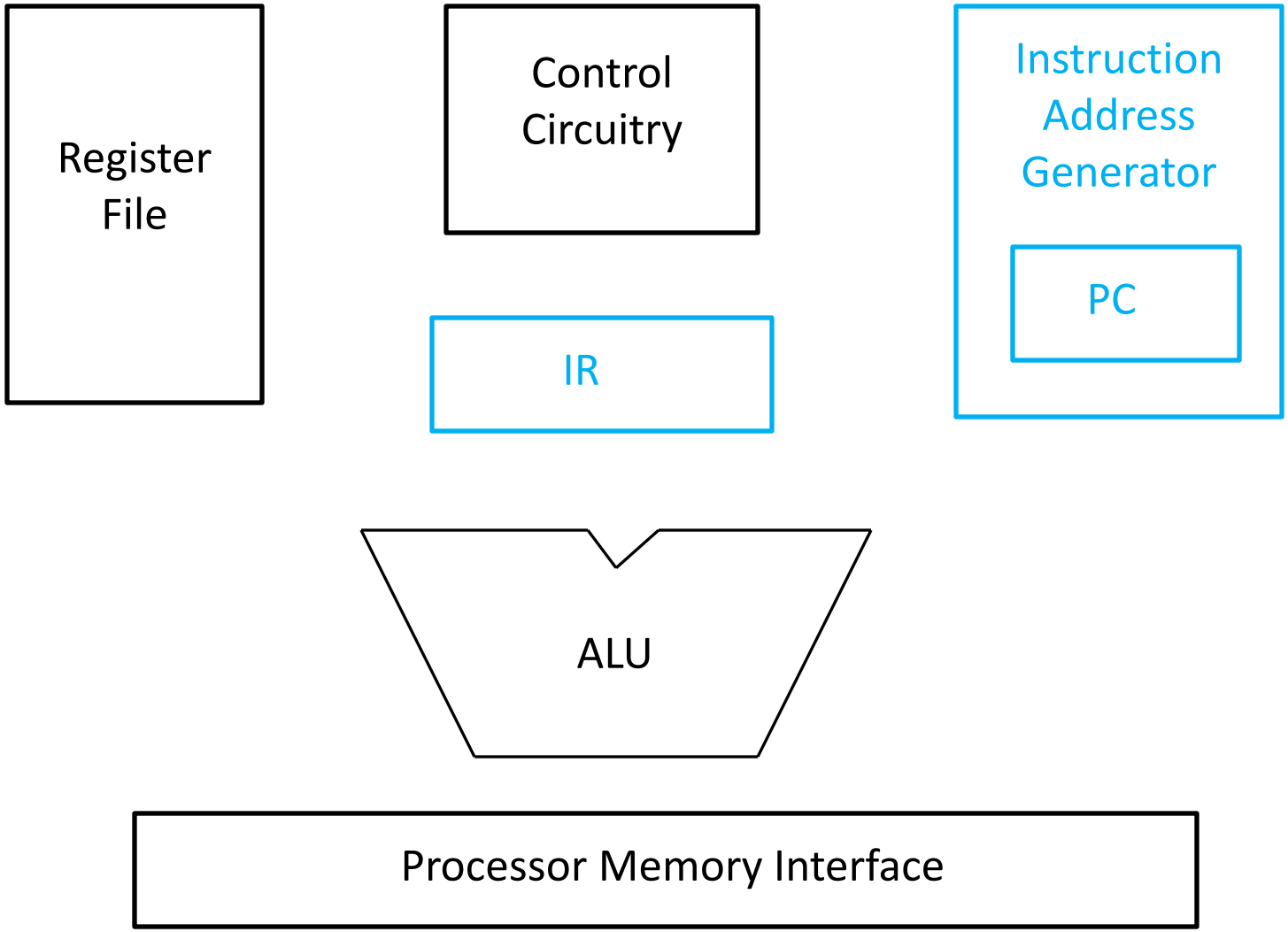
Subroutine Call Instruction:
saves the subroutine return address (current contents of PC) in a destination register

Instruction Type	MuxB selection	MuxY selection
Subroutine Call	x	2

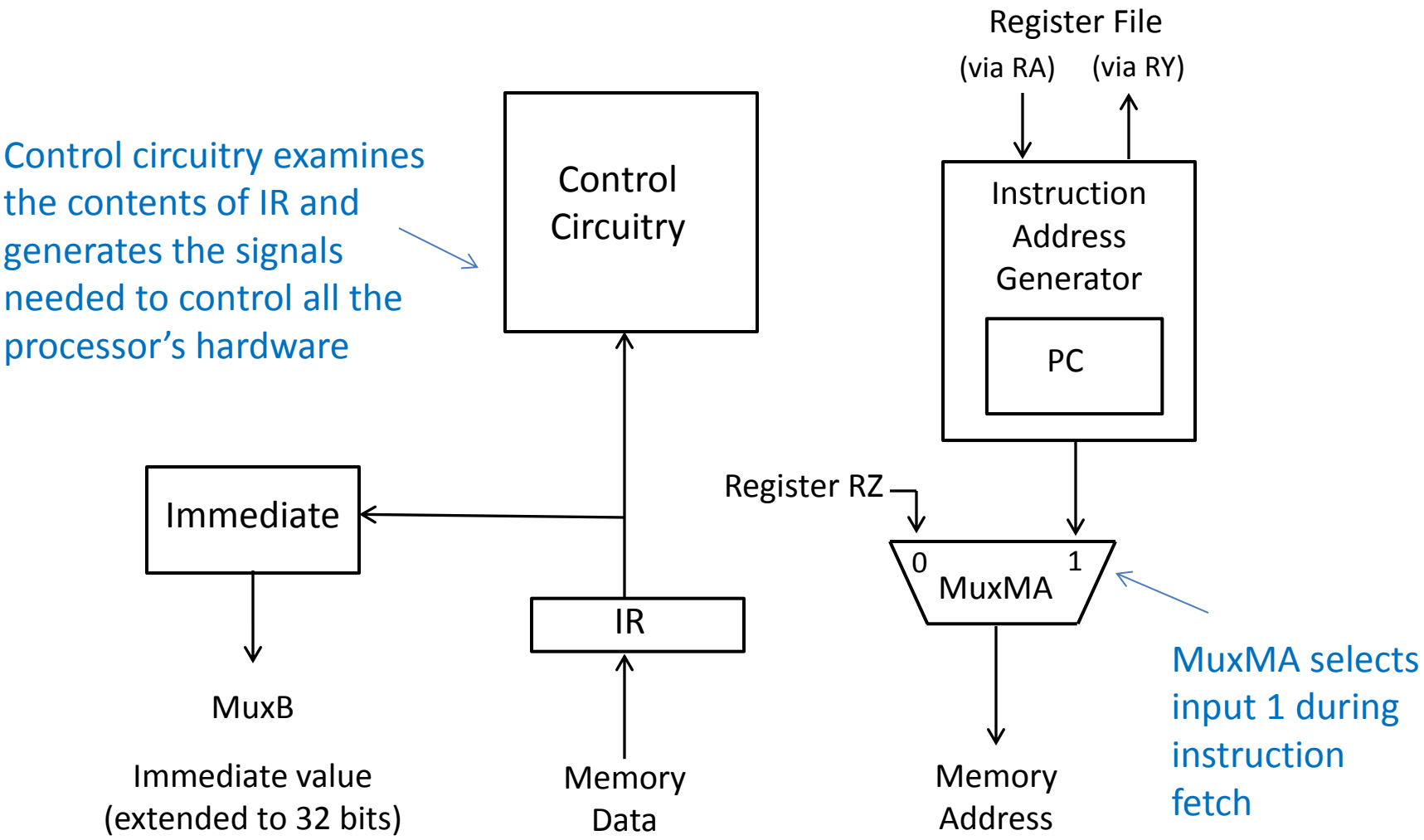
- Stage 3: No action
- Stage 4: Return address sent from PC to RY
- Stage 5: Contents of RY sent to the destination register



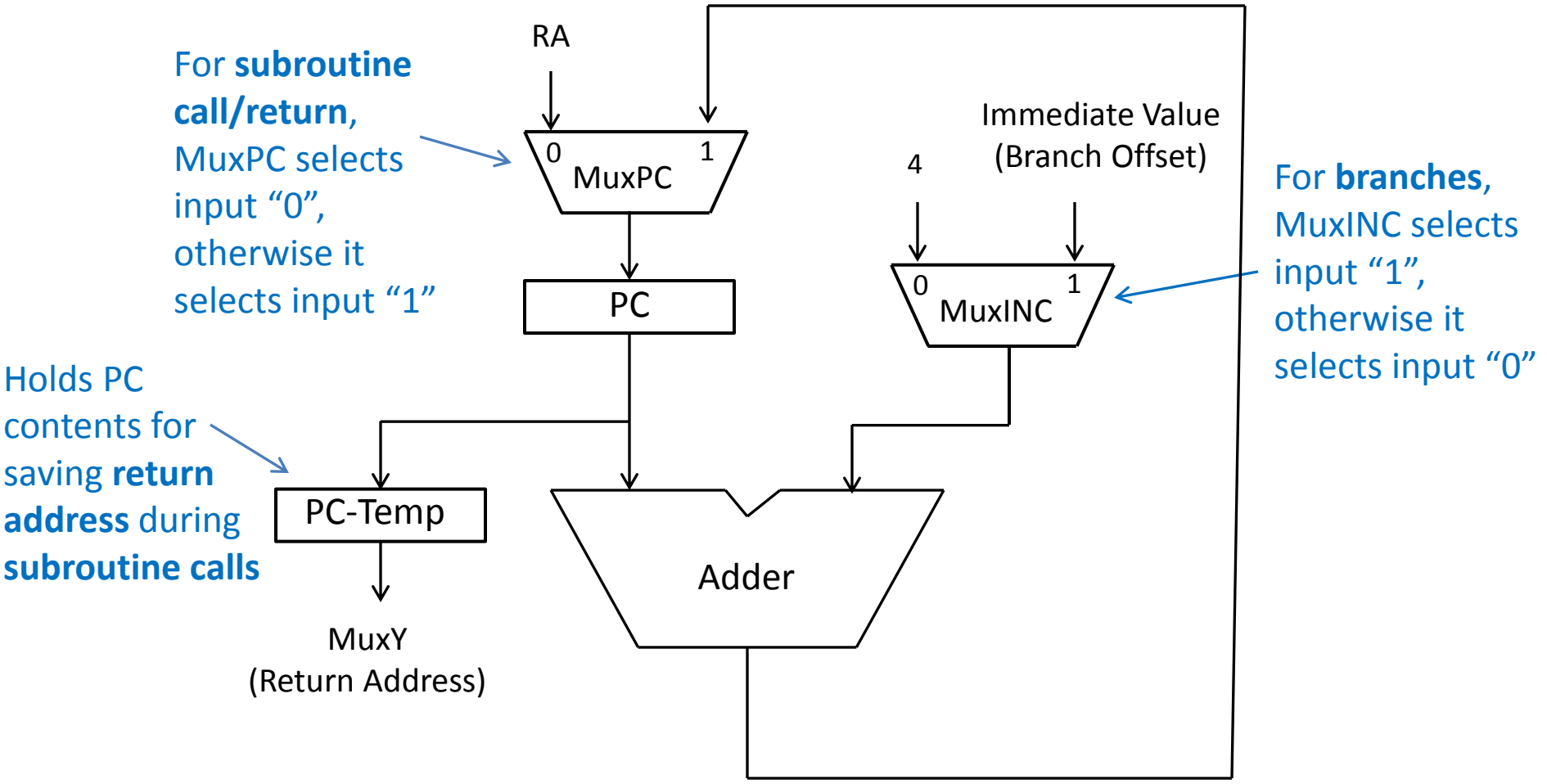
Hardware Components of a Processor



Instruction Fetch Step



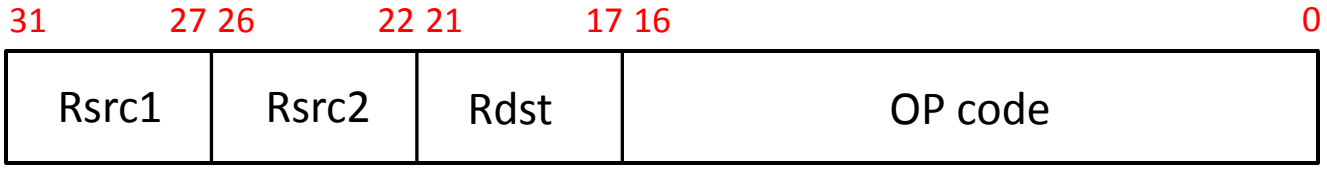
Instruction Address Generator



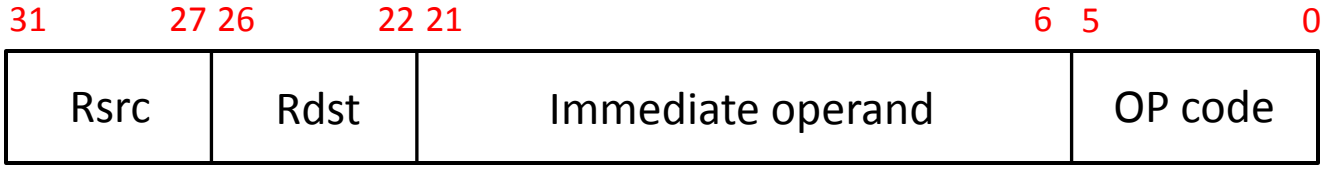
Immediate Field Extension

- Immediate operand needs to be represented as a 32-bit number before being used as ALU's input
- "Immediate" block extends the *immediate* field in IR and sends the extended value to MuxB
- For logical instructions (AND, OR etc.)
  - Immediate value is padded with zeroes to extend from 16 to 32 bits
- For arithmetic instructions (add, mult etc.) and branches
  - Immediate value is sign-extended from 16 to 32 bits

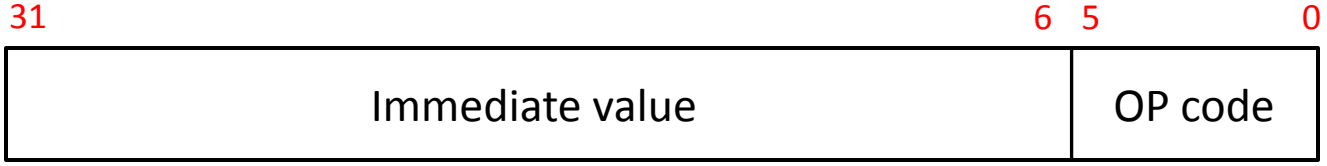
Instruction Encoding



(a) Register-operand format



(b) Immediate-operand format



(c) Call format

Using a few standard instruction formats simplifies the instruction decoding process

Instruction Encoding (cont.)

- The **OP code** field specifies the type of instruction
  - Control circuitry examines the op code to decide which control signals to assert in subsequent stages

- Can the reading of source registers proceed, while the control circuitry is yet to decode the instruction op code?
- YES, because source register addresses are specified using the same bit positions in all instructions (bits 31-27 and bits 26-22)
  - These two fields in IR (IR[31-27] and IR[26-22]) are connected to address inputs A and B of register file
  - **Contents of these two registers are loaded into RA and RB at the end of stage 2, irrespective of the type of instruction**
  - If these contents are not needed, subsequent stages can ignore them
  - If needed, these contents will be available as operands in stage 3

Example 1: **Add R3, R4, R5**

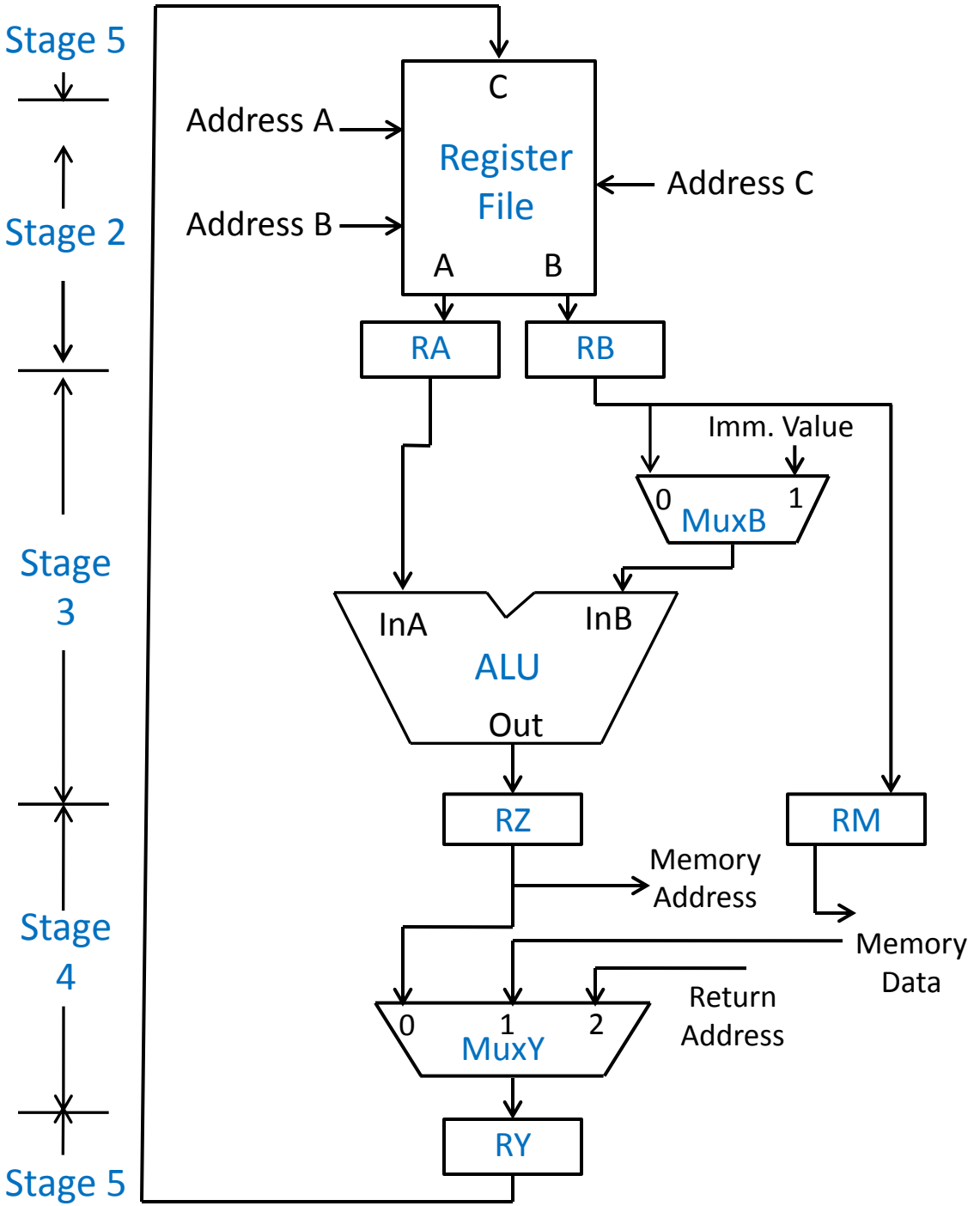
Stage 1:
Memory address ← [PC],
Read memory,
IR ← Memory data,
PC ← [PC] + 4

Stage 2:
Decode instruction,
RA ← [R4], RB ← [R5]

Stage 3:
RZ ← [RA] + [RB]

Stage 4:
RY ← [RZ]

Stage 5:
R3 ← [RY]



Example 2: **Load R5, X(R7)**

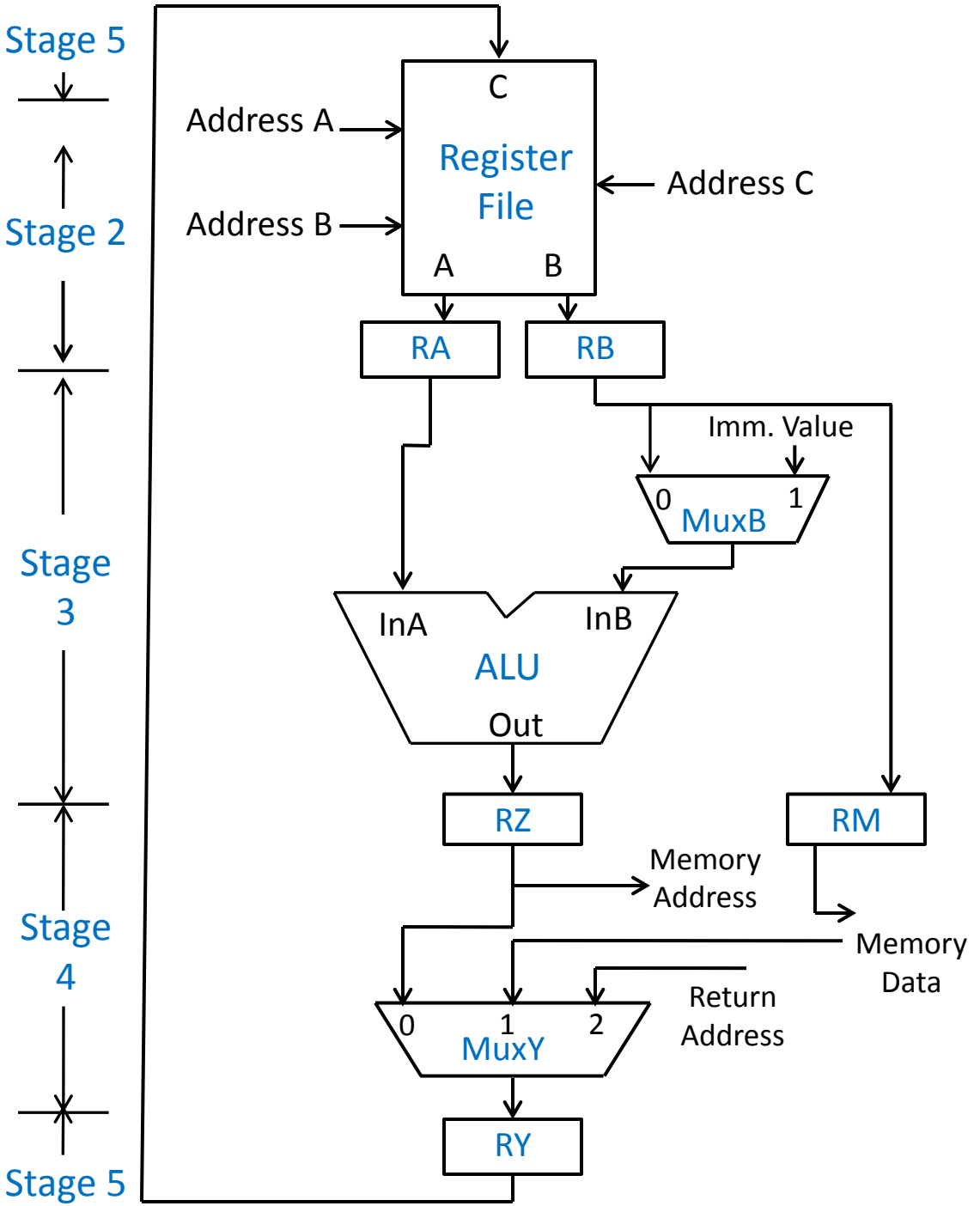
Stage 1:
Memory address ← [PC],
Read memory,
IR ← Memory data,
PC ← [PC] + 4

Stage 2:
Decode instruction,
RA ← [R7]

Stage 3:
RZ ← [RA] + Immediate value X

Stage 4:
Memory address ← [RZ],
Read memory
RY ← Memory data

Stage 5:
R5 ← [RY]



Example 3: **Store R6,X(R8)**

Stage 1:
Memory address ← [PC],
Read memory,
IR ← Memory data,
PC ← [PC] + 4

Stage 2:
Decode instruction,
RA ← [R8], RB ← [R6]

Stage 3:
RZ ← [RA] + Immediate value X
RM ← [RB]

Stage 4:
Memory address ← RZ,
Memory data ← [RM],
Write memory

Stage 5:
No action