

Computer Organization & Architecture
**Chapter 5 – Instruction
Execution**

Zhang Yang 张杨

cszyang@scut.edu.cn

Autumn 2025



Content of this lecture

- 5.1 Some Fundamental Concepts
 - Processing Unit
 - Functions of Processing Unit
 - Register Organization of Processor
 - Instruction Execution
 - Processor's Building Blocks
 - A Digital Processing System
 - A Multi-stage Digital Processing System



Processing Unit

- Instruction Set Processor or Central Processing Unit (CPU)
- A **processor** is responsible for reading program instructions from the computer's memory and executing them.
 - It fetches one instruction at a time.
 - It decodes (interprets) the instruction.
 - Then, it carries out the actions specified.



Functions of Processing Unit

- Control program sequences
- Control instruction operations
- Control timing
- Data processing

Register Organization of Processor (1)

■ User-Visible Registers

- A user-visible register is one that may be referenced by means of the machine language that the CPU executes.
- General-purpose registers
- Data registers
 - Hold data and cannot be employed in the calculation of an operand address.
- Address registers
 - Segment pointers
 - In a machine with segmented addressing, a segment register holds the address of the base of the segment.

Register Organization of Processor (2)

■ User-Visible Registers (ctd.)

□ Address registers (ctd.)

- Index registers
- Stack pointer
 - Pentium 4's ESP register
- ...

□ Condition codes register

- At least partially visible to the user.
- Generally, machine instructions allow condition code bits to be read by implicit reference, but they cannot be altered by the programmer.

Register Organization of Processor (3)

■ Control and Status Registers

- Most of these registers, on most machines are not visible to the user.

- Some of them may be visible to machine instructions executed in a control or operating system mode.

- Program Counter (PC)

- Instruction Register (IR)

- Memory Address Register (MAR)

- Memory Data Register (MDR)

} Essential to instruction execution

Register Organization of Processor (4)

■ Control and Status Registers (ctd.)

□ Program Status Word (PSW)

- Include a register or a set of registers.
- The PSW typically contains condition codes plus other status information:
 - N, Z, C, V
 - Equal: set if a logical compare result is equality
 - Interrupt enable/disable: used to enable or disable interrupts.
 - Supervisor: indicates whether the CPU is executing in supervisor or user mode.

□ Interrupt vector register

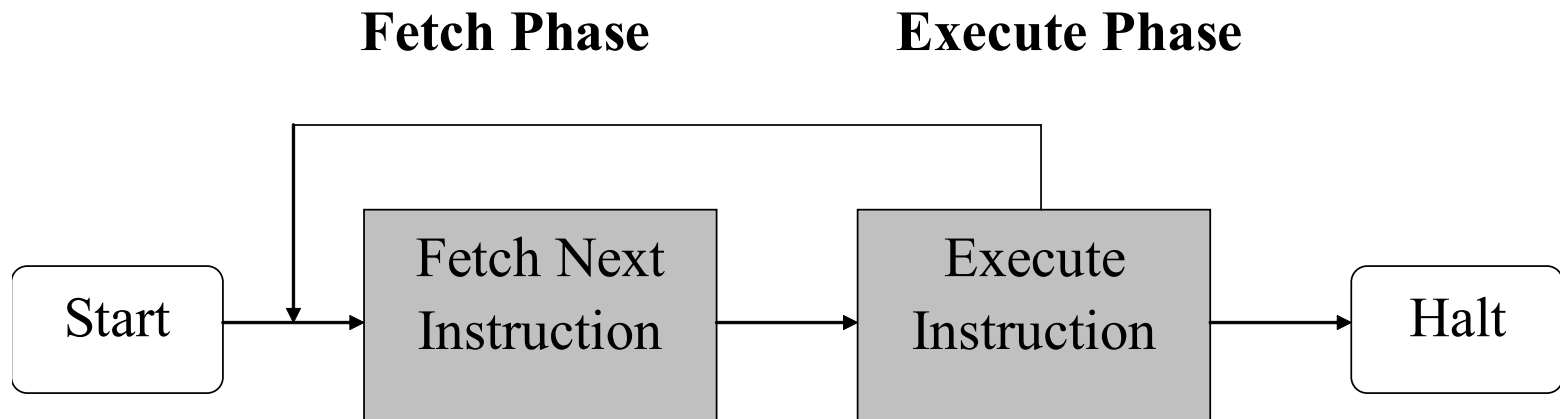
□ Page table pointer

□ ...

} Related to a particular processor design

Instruction Execution (1)

- To execute an instruction, the processor has to perform the following steps:



- 1. Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR. $IR \leftarrow [[PC]]$

Instruction Execution (2)

■ Instruction Execution (ctd.)

- To execute an instruction, the processor has to perform the following steps:
 - 2. Increment the contents of the PC by 4. $PC \leftarrow [PC] + 4$
 - Assuming that the memory is byte addressable, and each instruction comprises 4 bytes.
 - 3. Carry out the actions specified by the instruction in the IR.
 - Note
 - If an instruction occupies more than one memory word, steps 1 and 2 must be repeated as many times as necessary to fetch the complete instruction.

Processor's Building Blocks

■ Figure 5.1

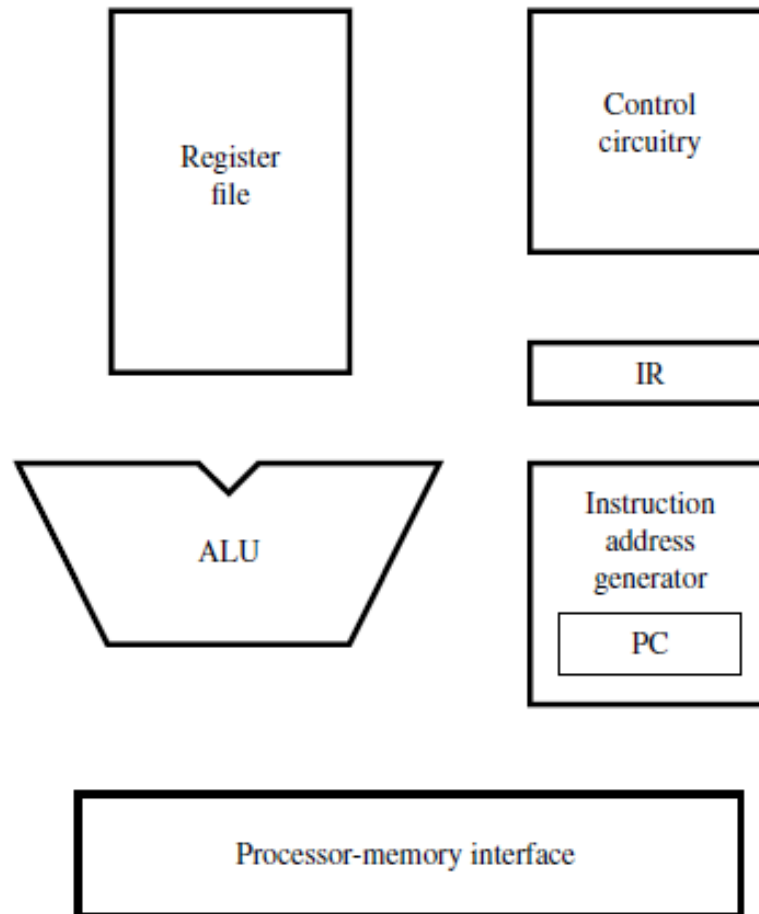


Figure 5.1 Main hardware components of a processor.

A Digital Processing System

- Contents of register A are processed and deposited in register B.

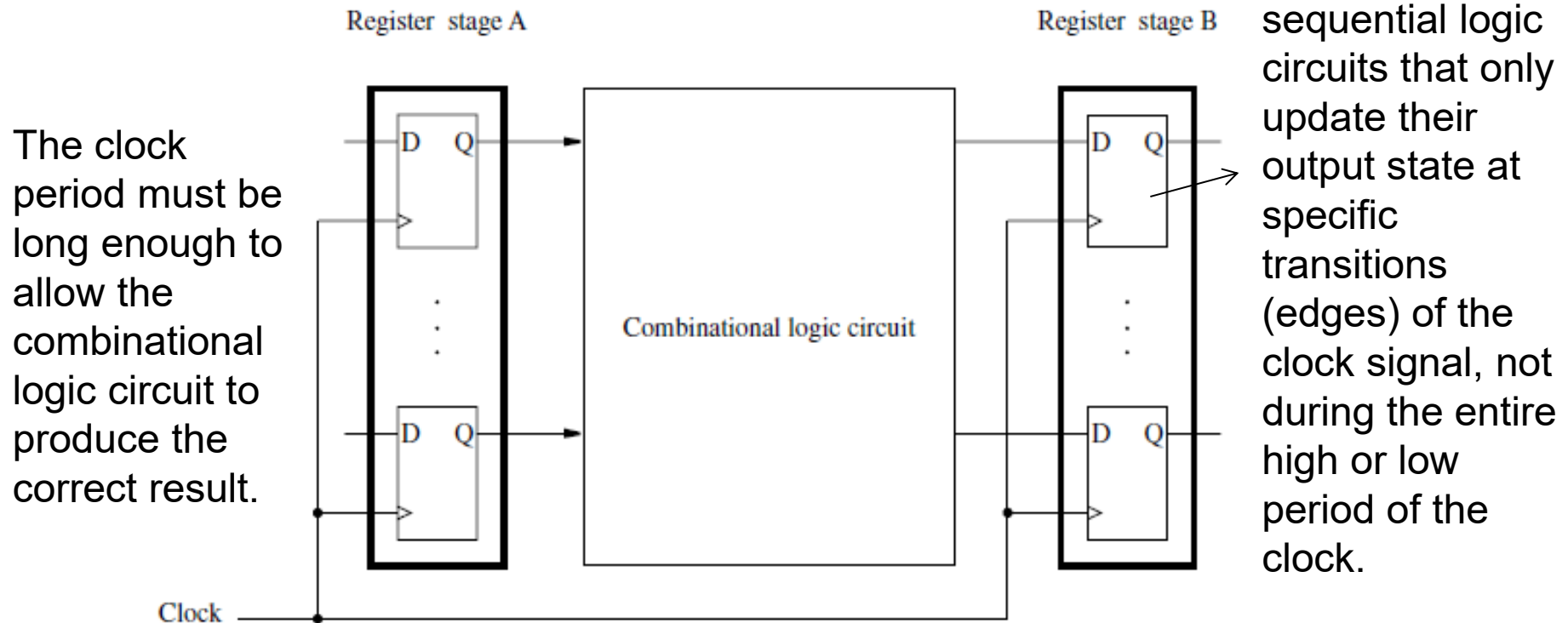
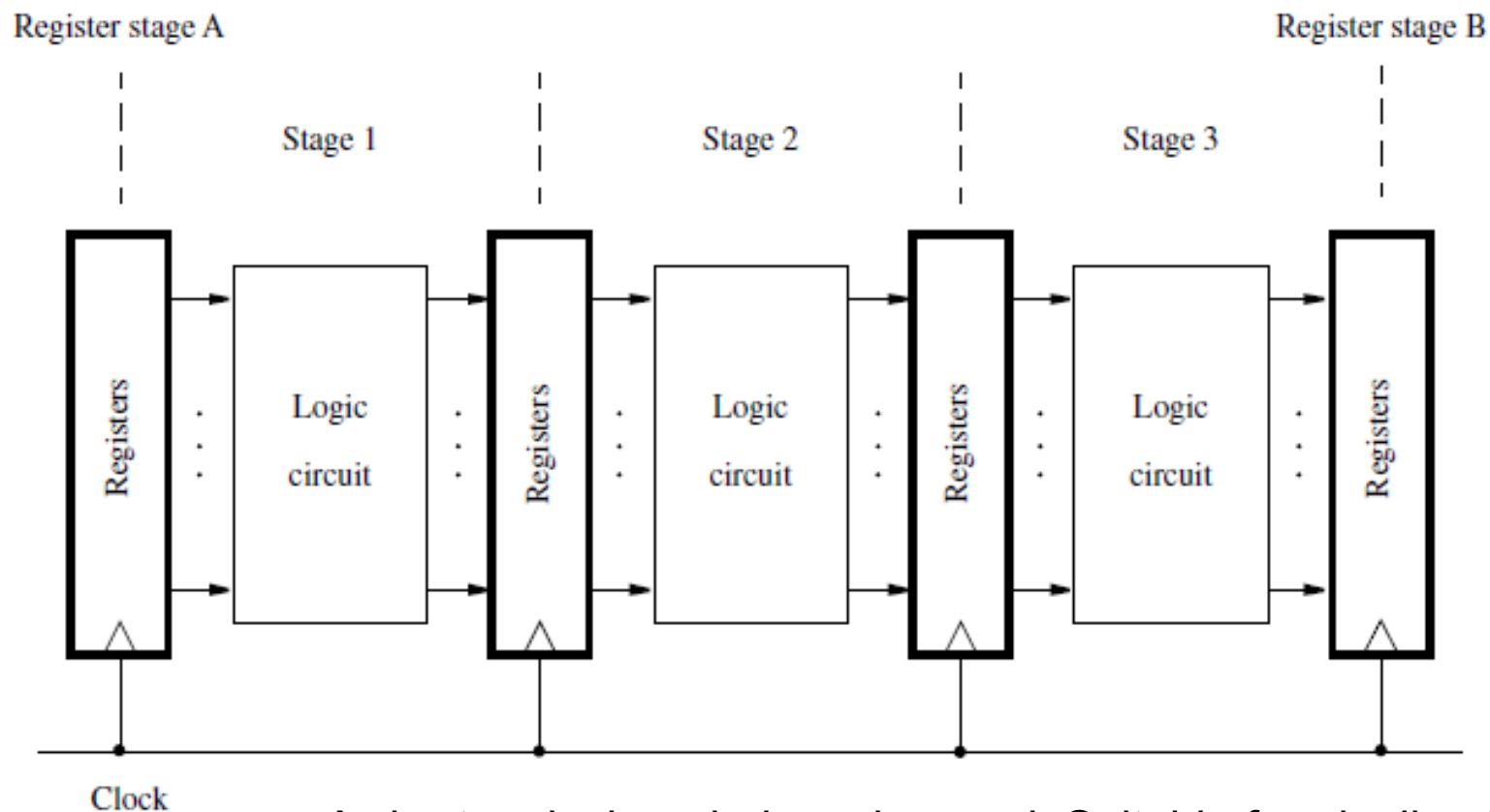


Figure 5.2 Basic structure for data processing.

A Multi-Stage Digital Processing System (1)

■ A Hardware Structure with Multiple Stages



A shorter clock period can be used. Suitable for pipelined operation.

Figure 5.3 A hardware structure with multiple stages.

A Multi-Stage Digital Processing System (2)

■ Why Multi-Stage?

- Processing moves from one stage to the next in each clock cycle.
- Such a multi-stage system is known as a **pipeline**.
- High-performance processors have a pipelined organization.
- Pipelining enables the execution of successive instructions to be overlapped.
- Pipelining will be discussed later.



Content of this lecture

■ 5.2 Instruction Execution

- Instruction Execution
- Load Instruction
- Arithmetic and Logic Instruction
- Store Instruction
- Five-Step Sequence of Actions



Instruction Execution

- Pipelined organization is most effective if all instructions can be executed in the same number of steps.
- Each step is carried out in a separate hardware stage.
- Processor design will be illustrated using five hardware stages.
- How can instruction execution be divided into five steps?

Load Instruction (1)

■ Example: Load R5,X(R7)

□ Actions required for execution of the instruction

- Fetch the instruction from the memory.
- Increment the program counter.
- Decode the instruction to determine the operation to be performed.
- Read register R7.
- Add the immediate value X to the contents of R7.
- Use the sum $X+[R7]$ as the effective address of the source operand, and read the contents of that location in the memory.
- Load the data received from the memory into the destination register, R5.

Load Instruction (2)

■ Example: Load R5,X(R7) (ctd.)

- Depending on how the hardware is organized, some of these actions can be performed at the same time.
 - 1. Fetch the instruction and increment the program counter.
 - 2. Decode the instruction and read the contents of register R7 in the register file.
 - 3. Compute the effective address.
 - 4. Read the memory source operand.
 - 5. Load the operand into the destination register, R5.

Arithmetic and Logic Instructions

- Differ from the Load instruction
 - Two source operand
 - No access to memory operand
- Example Add R3, R4, R5
 - 1. Fetch the instruction and increment the program counter.
 - 2. Decode the instruction and read the contents of source registers R4 and R5.
 - 3. Compute the sum $[R4] + [R5]$.
 - 4. No action.
 - 5. Load the result into the destination register, R3.
 - *Stage 4 (memory access) is not involved in this instruction.*

Store Instruction

■ Example: Store R6, X(R8)

- 1. Fetch the instruction and increment the program counter.
- 2. Decode the instruction and read registers R6 and R8.
- 3. Compute the effective address $X + [R8]$.
- 4. Store the contents of register R6 into memory location $X + [R8]$.
- 5. No action.

Five-Step Sequence of Actions (1)

■ For RISC Computers

- This sequence determines the hardware stages needed.

Step	Action
1	Fetch an instruction and increment the program counter.
2	Decode the instruction and read registers from the register file.
3	Perform an ALU operation.
4	Read or write memory data if the instruction involves a memory operand.
5	Write the result into the destination register, if needed.

Figure 5.4 A five-step sequence of actions to fetch and execute an instruction.

Five-Step Sequence of Actions (2)

■ Comments

- The five-step sequence is suitable for all instructions in a RISC-style instruction set.
- Instructions that perform computations use data that are either stored in general-purpose registers or given as immediate data in the instruction.
- The five-step sequence is suitable for all Load and Store instructions, because the addressing modes that can be used in these instructions are special cases of the Index mode.
 - R0 as index register



Content of this lecture

- 5.3 Hardware Components
 - Register File
 - ALU
 - Datapath
 - Instruction Fetch Section

Register File (1)

- General-purpose registers are implemented in the form of a register file.
- Register File
 - A small and fast memory block: an array of storage elements with access circuitry.
- Port: The inputs and outputs of any memory unit are often called input and output ports.
- A **2-port register file** is needed to read the two source registers at the same time.
- A memory unit that has two output ports is said to be *dual-ported*.
 - It may be implemented using a 2-port memory.
 - Or using two single-ported memory blocks.

Register File (2)

Two Alternatives

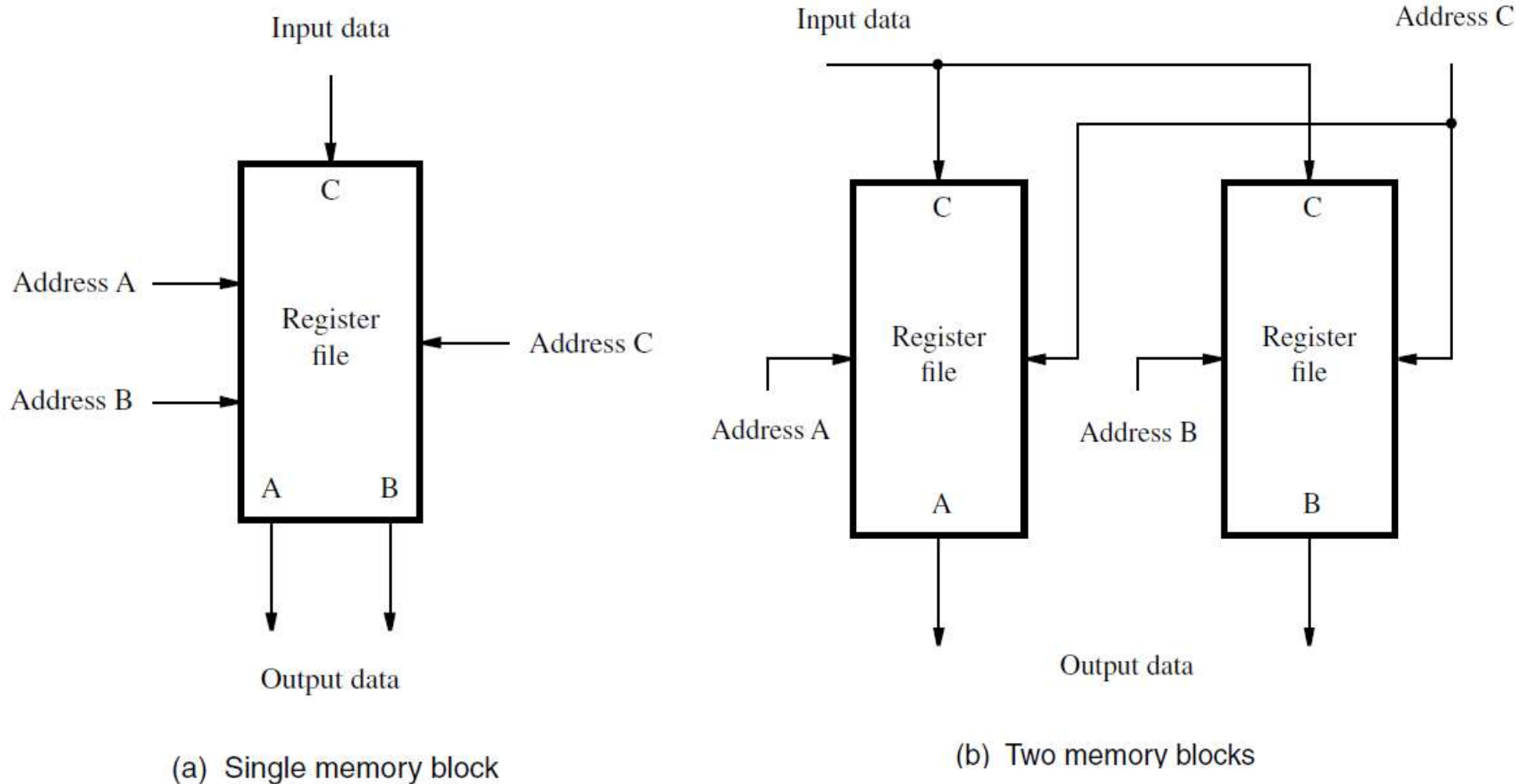


Figure 5.5 Two alternatives for implementing a dual-ported register file.

ALU (1)

■ Figure 5.6

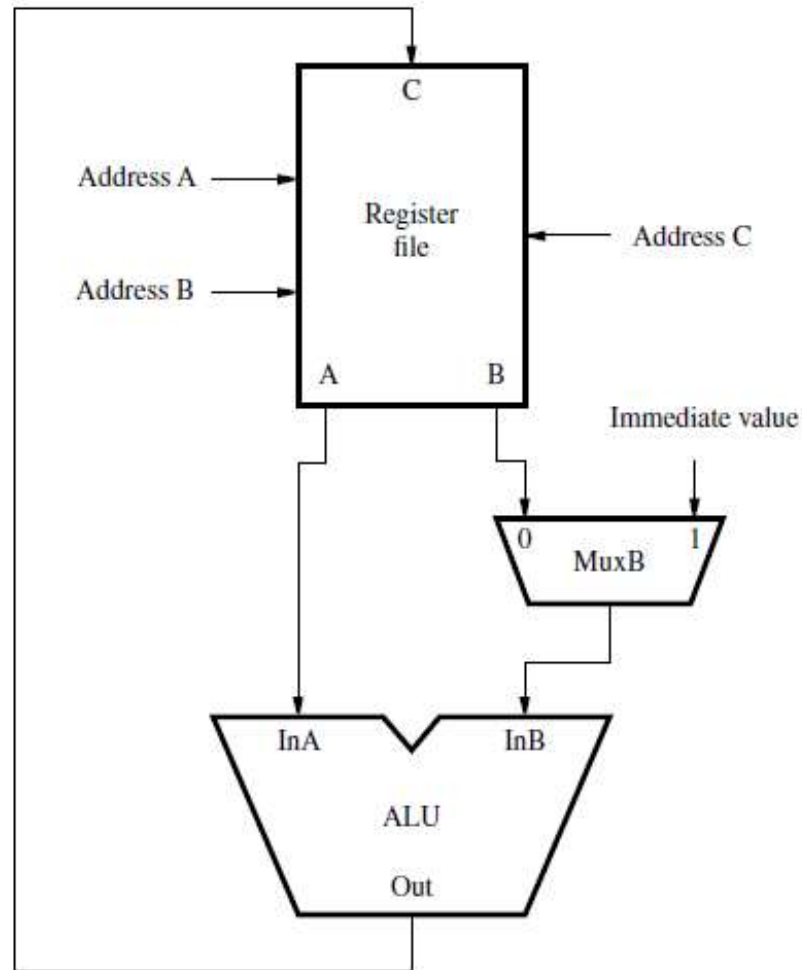
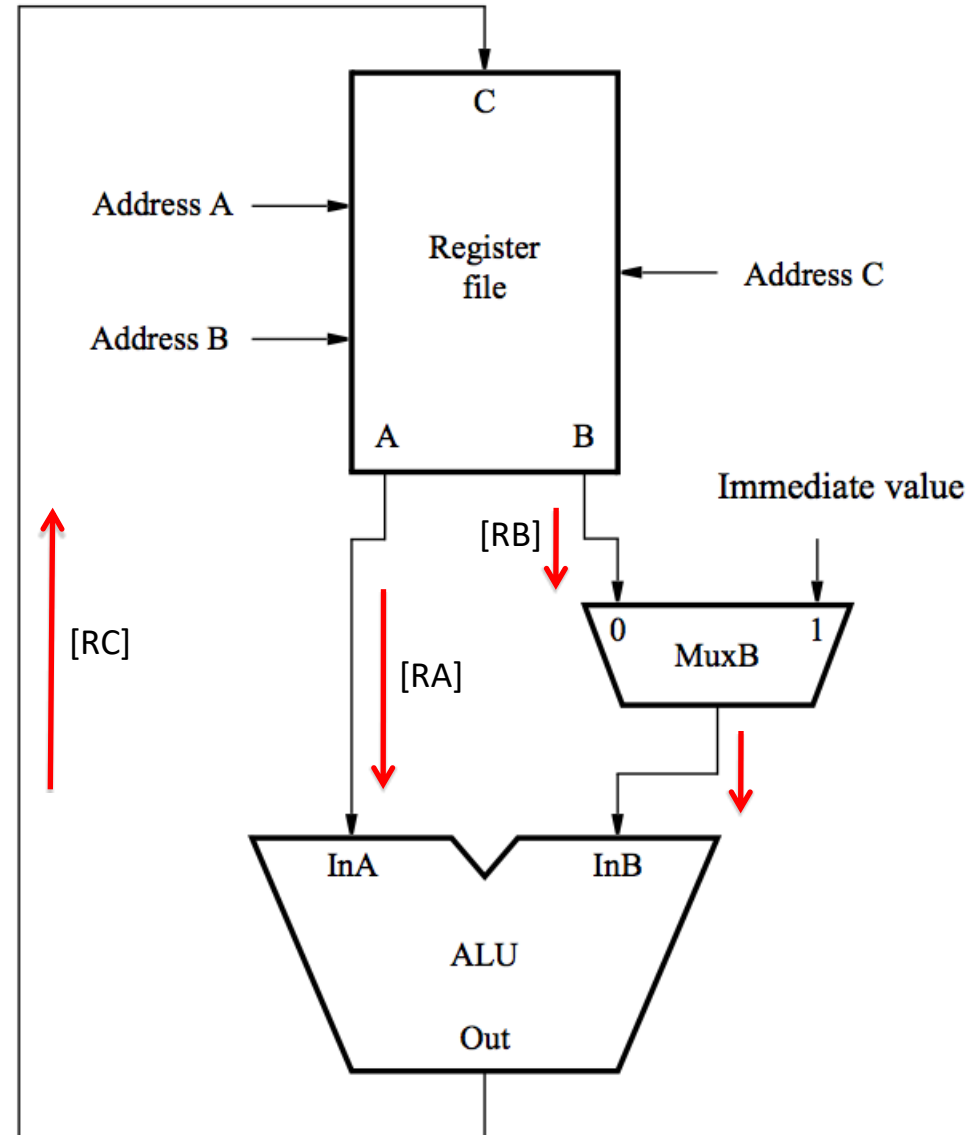


Figure 5.6 Conceptual view of the hardware needed for computation.

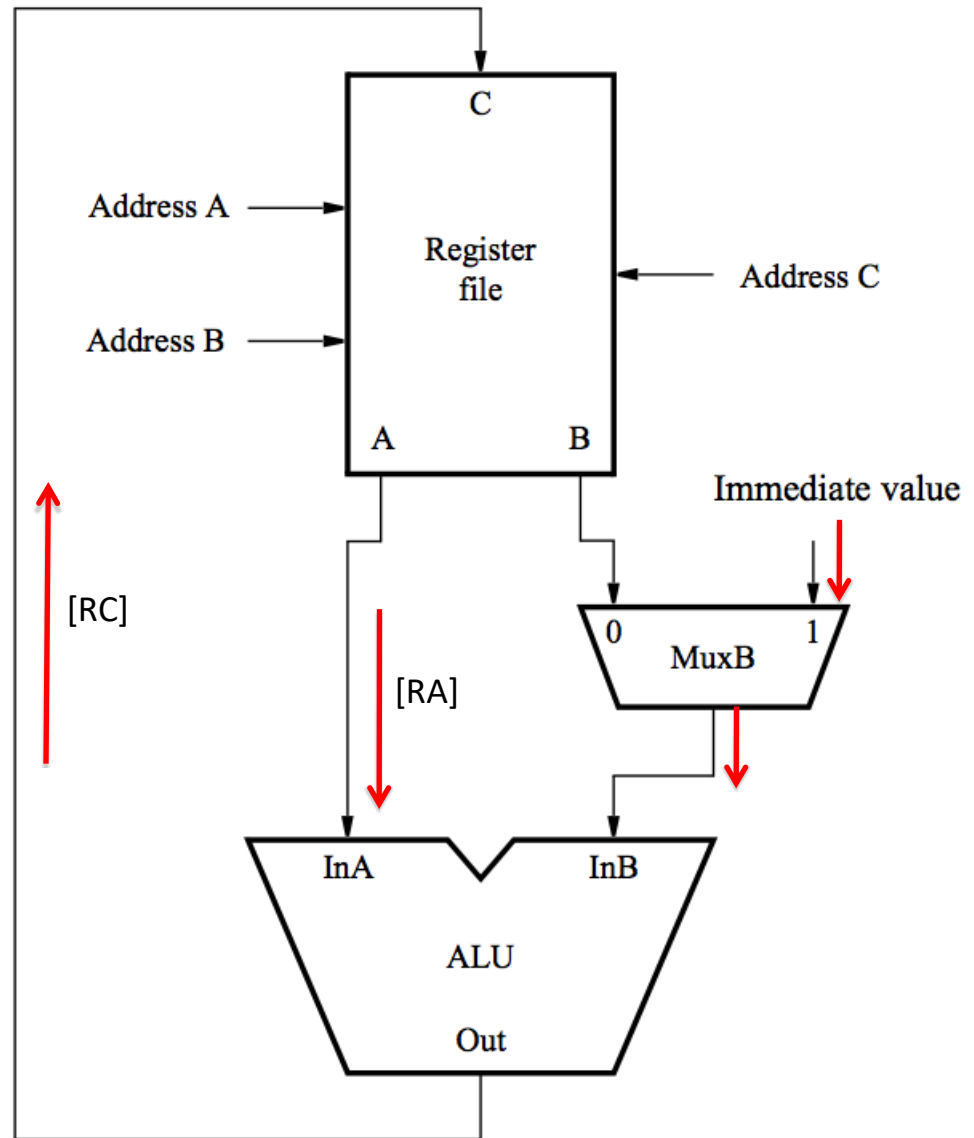
ALU (2)

- A conceptual view of computational instructions
 - Both source operands and the destination location are in the register file.



ALU (3)

- A conceptual view of immediate instructions:
 - One of the source operands is the immediate value in the IR.



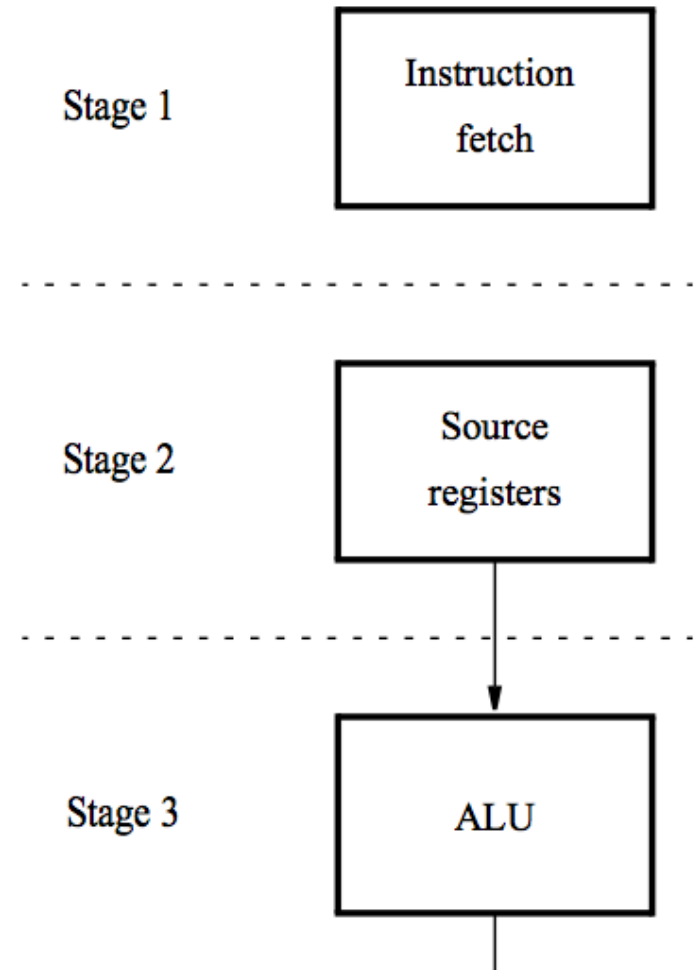
Datapath (1)

- Two Phase of Instruction Processing
 - Fetch Phase
 - Execution Phase
- Divide the processor hardware into two corresponding sections:
 - Fetch Section: fetching instruction, decoding instruction, generating control signals
 - Execution Section: reading operands, performing computation, storing results
- A datapath is a collection of functional units, such as arithmetic logic units or multipliers, that perform data processing operations.

Datapath (2)

■ A Five-Stage Organization

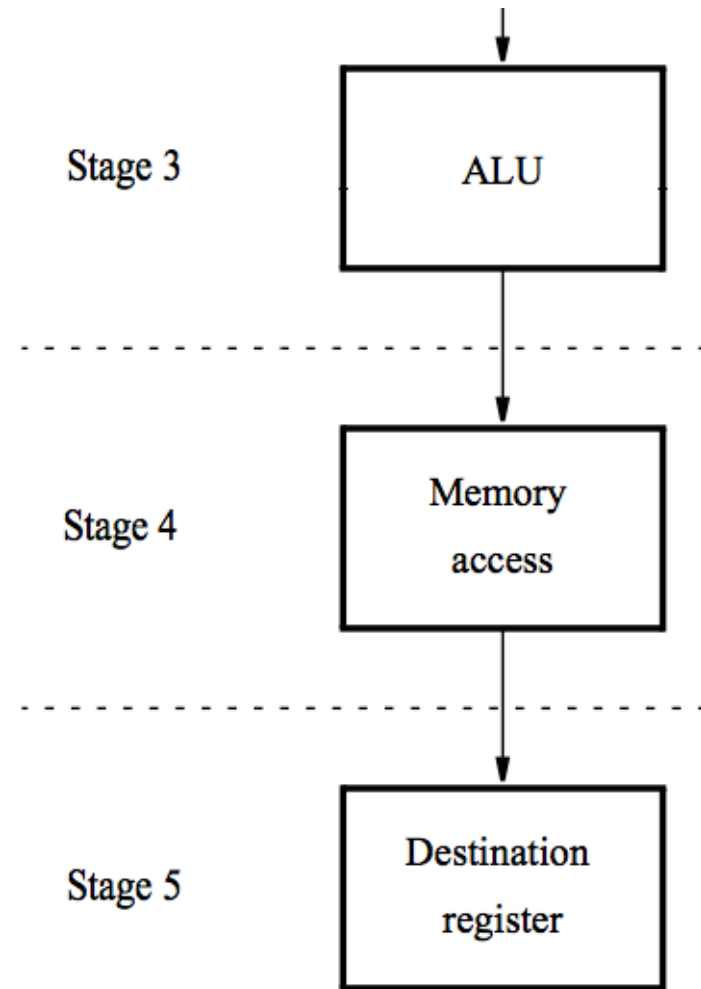
- Instruction processing moves from each stage to the next in every clock cycle.
- The instruction is decoded and the source registers are read in stage 2.
- Computation takes place in the ALU in stage 3.



Datapath (3)

■ A Five-Stage Organization (ctd.)

- If a memory operation is involved, it takes place in stage 4.
- The result of the instruction is stored in the destination register in stage 5.



Datapath (4)

■ The Datapath-Stages 2 to 5

□ Register file

Inter-stage registers needed to carry data from one stage to the next.

□ ALU stage

□ Memory stage

□ Back to the register file

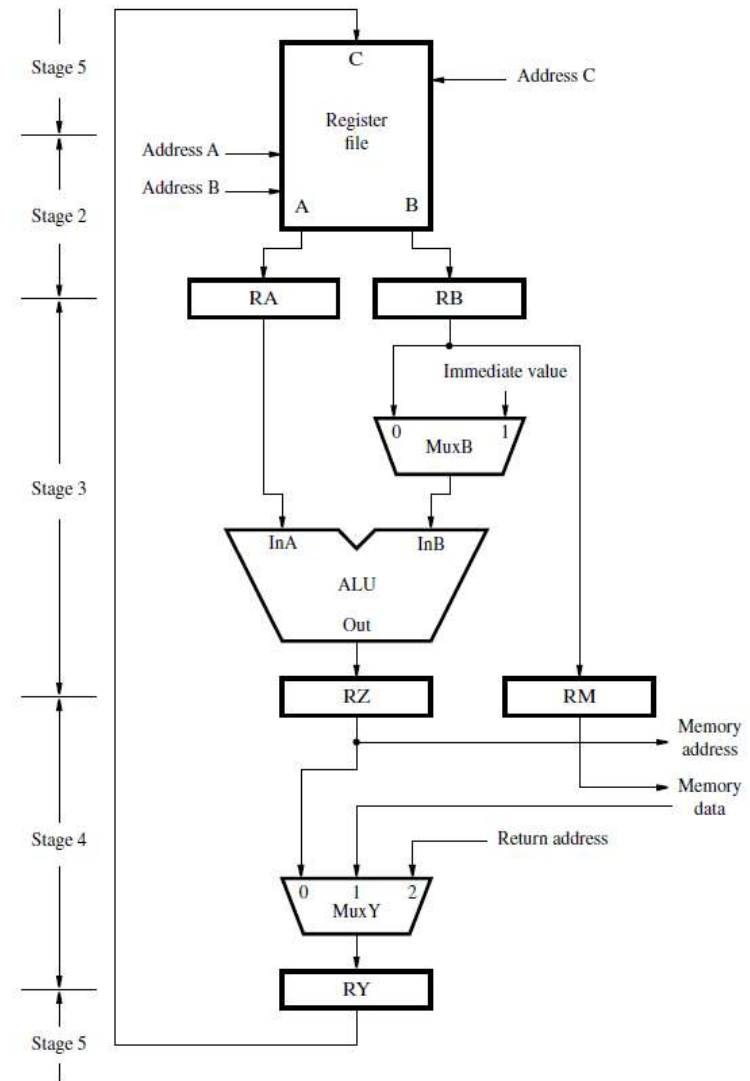
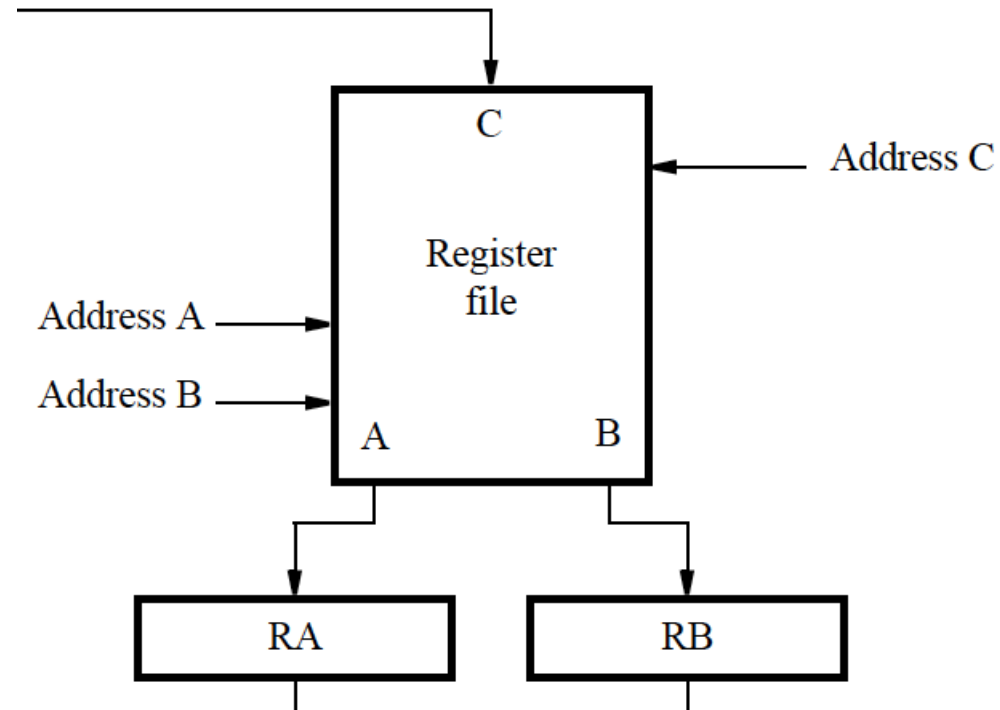


Figure 5.8 Datapath in a processor.

Datapath (5)

■ Register File-Stages 2&5

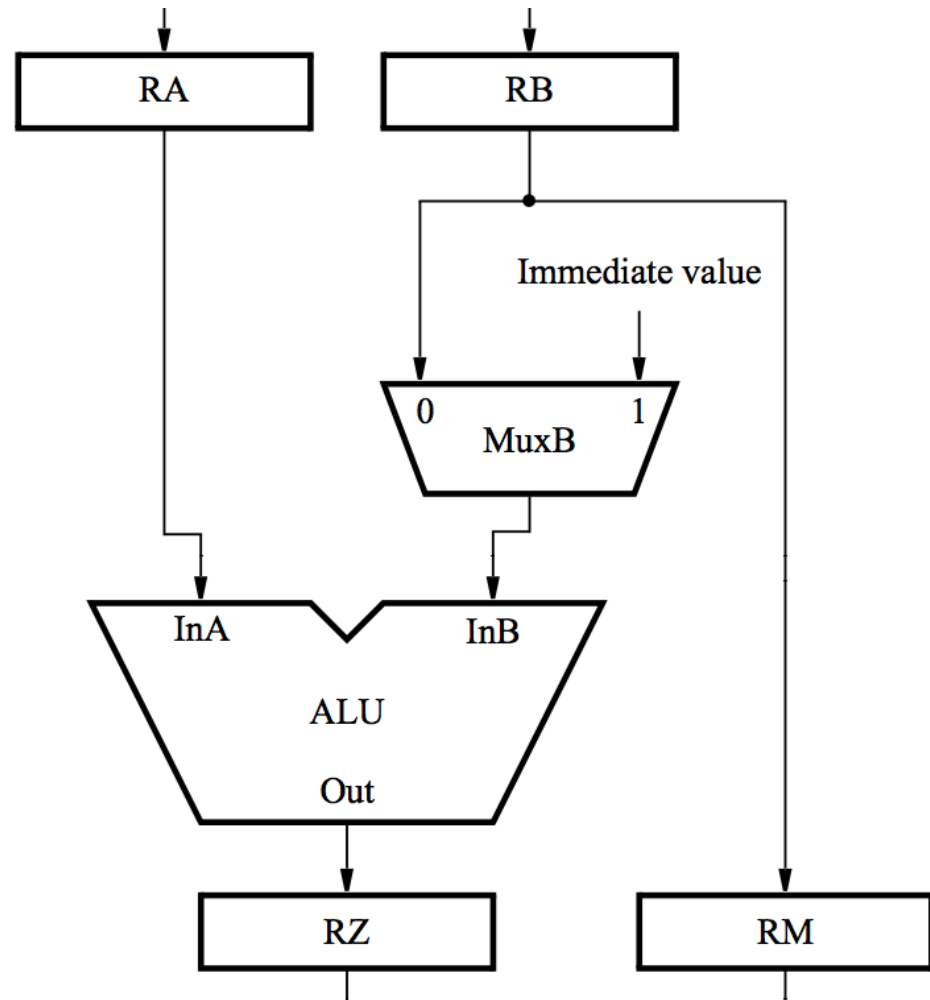
- Address inputs connected to corresponding fields in IR.
- Source registers are read and their contents stored in RA and RB.
- The result of the instruction is stored in the destination register selected by address C.



Datapath (6)

ALU Stage

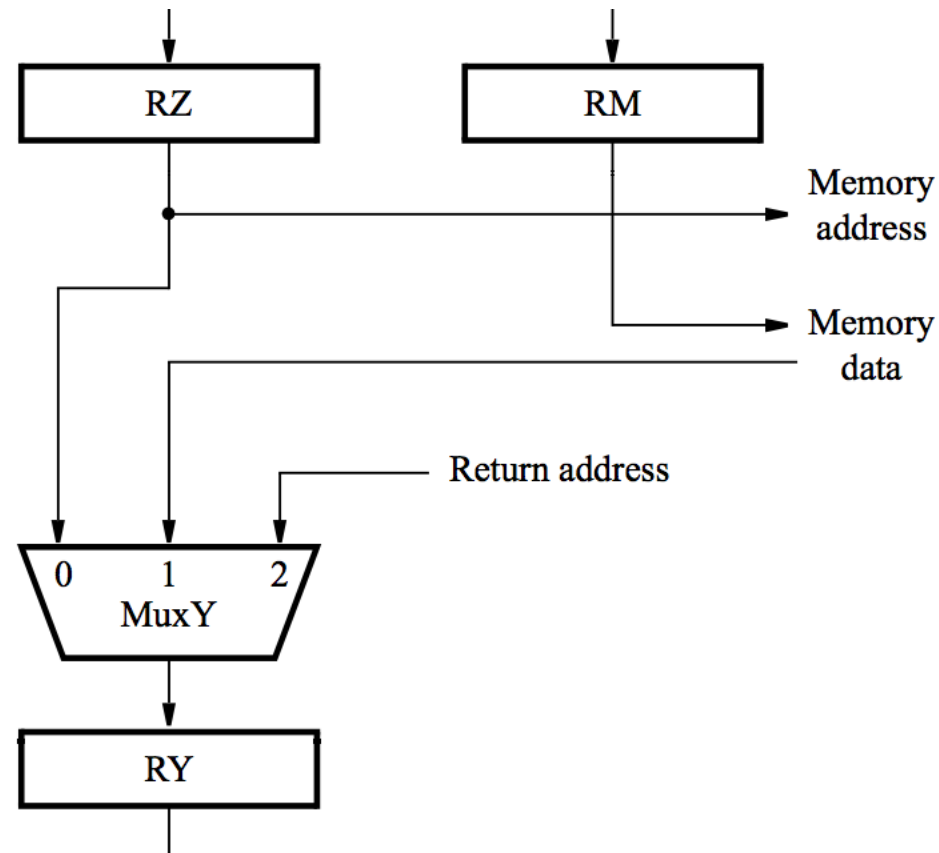
- ALU performs calculation specified by the instruction.
- Multiplexer MuxB selects either RB or the Immediate field of IR.
- Results stored in RZ.
- Data to be written in the memory are transferred from RB to RM.



Datapath (7)

■ Memory Stage

- For a memory instruction, RZ provides memory address, and MuxY selects data read to be placed in RY.
- RM provides data for a memory write operation.
- For a calculation instruction, MuxY selects [RZ] to be placed in RY.
- Input 2 of MuxY is used in subroutine returns or interrupt returns.



Instruction Fetch Section (1)

■ Figure 5.9

- Processor Control Section
- Memory Address Generation Section

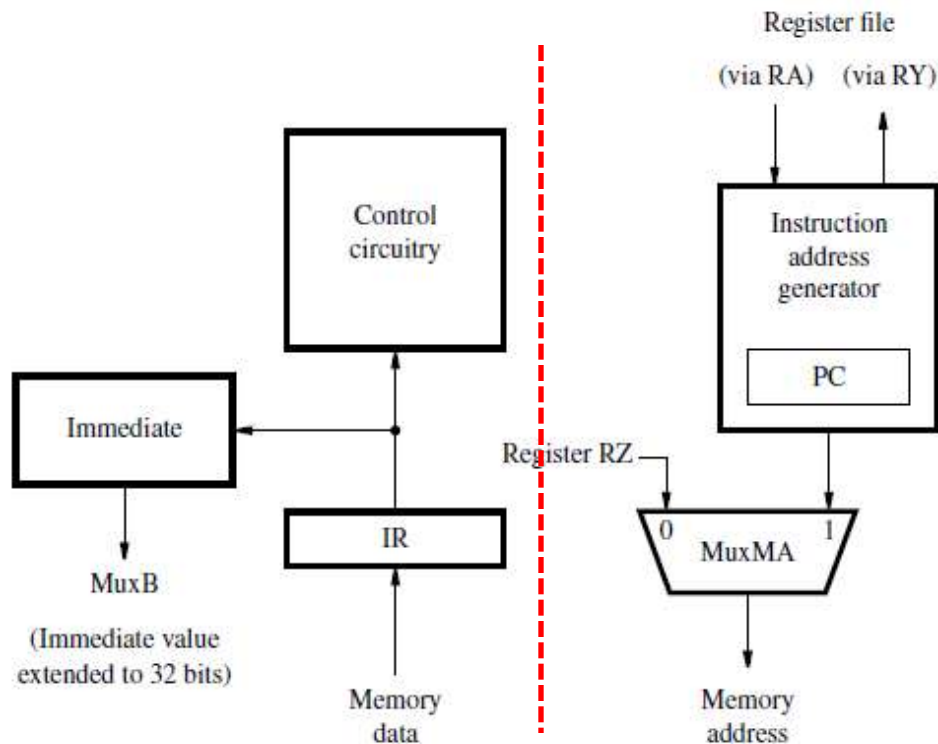
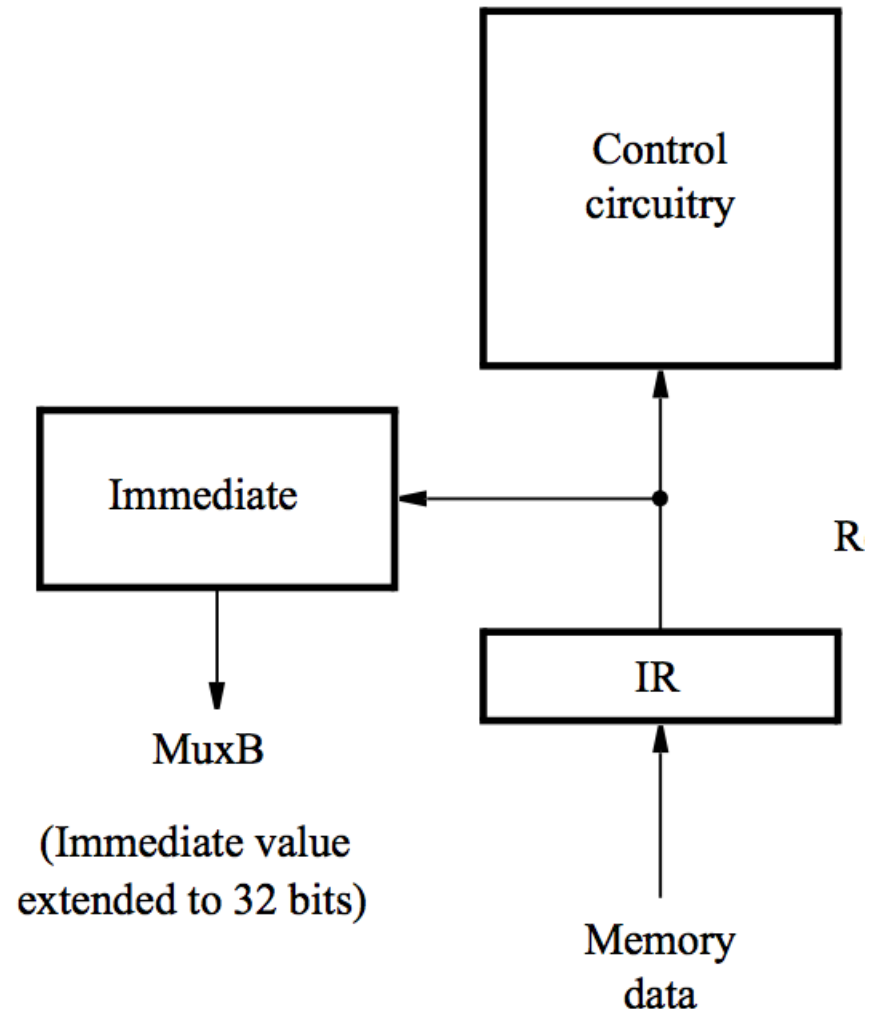


Figure 5.9 Instruction fetch section of Figure 5.7.

Instruction Fetch Section (2)

■ Processor Control Section

- When an instruction is read, it is placed in IR.
- The control circuitry decodes the instruction.
- It generates the control signals that drive all units.
- The Immediate block extends the immediate operand to 32 bits as specified in the instruction.

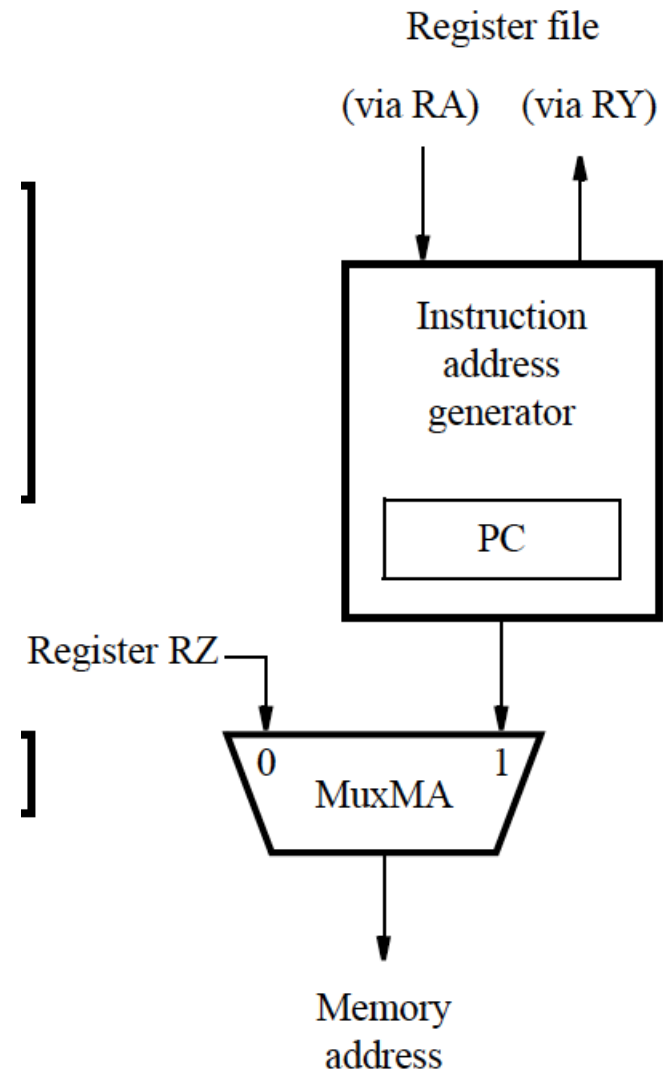


Instruction Fetch Section (3)

■ Memory Address

Generation Section

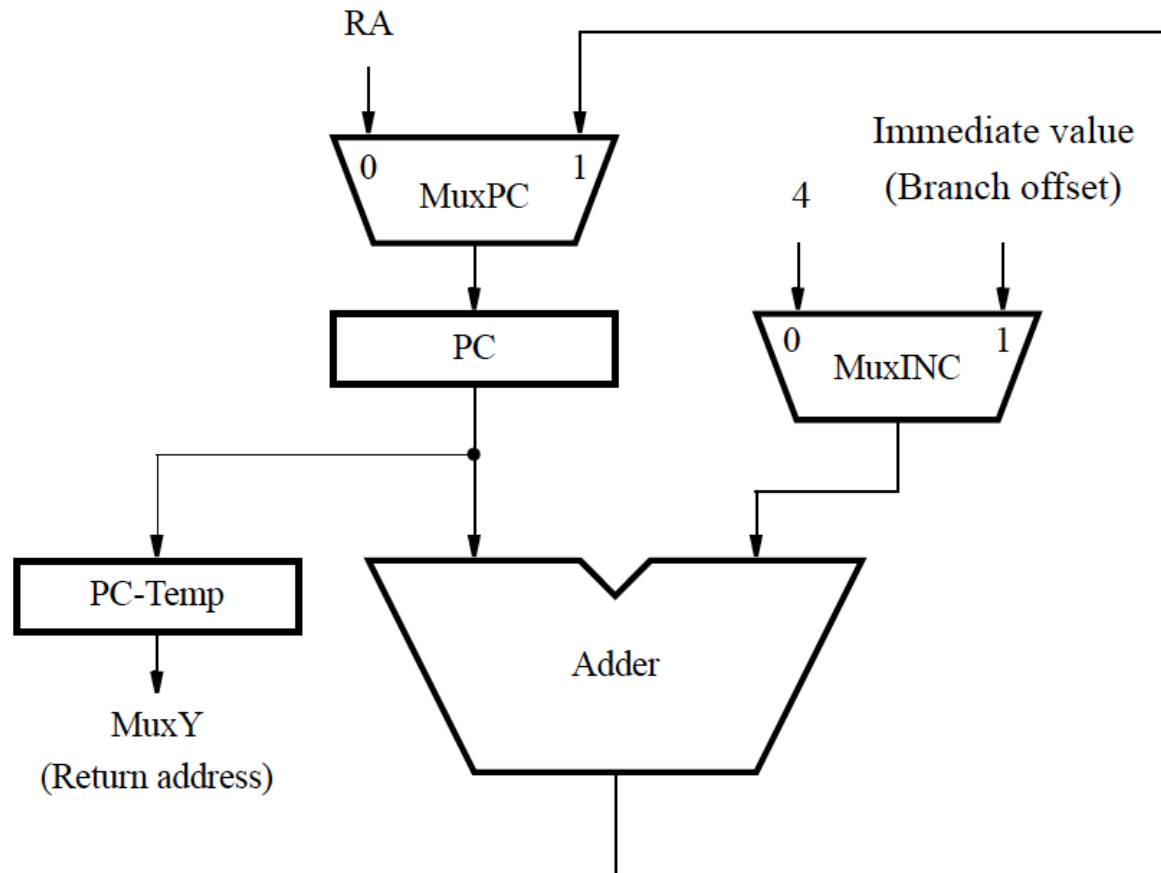
- MuxMA selects the PC when fetching instructions.
- The Instruction address generator increments the PC after fetching an instruction.
- It also generates branch and subroutine addresses.
- MuxMA selects RZ when reading or writing data operands.



Instruction Fetch Section (4)

■ Instruction Address Generator

- Connections to RY and RA are used for subroutine call and return.
- The branch offset is given in the immediate field of the IR





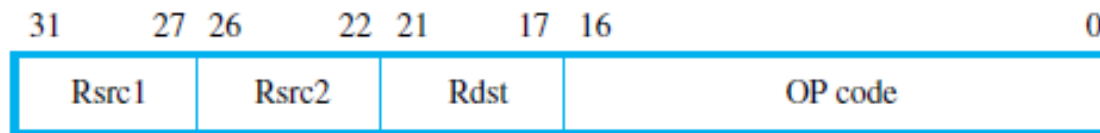
Content of this lecture

- 5.4 Instruction Fetch and Execution Steps
 - Add Instructions
 - Load Instructions
 - Store Instructions
 - Branch Instructions
- Summary

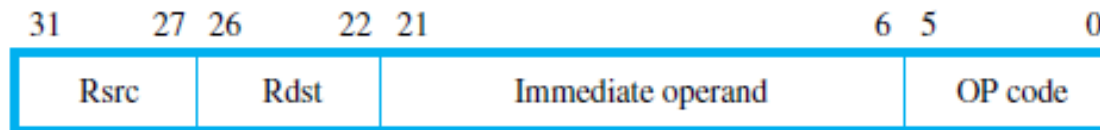
Add Instructions (1)

■ Example: Add R3, R4, R5

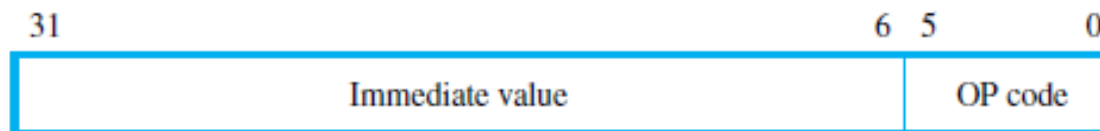
□ Instruction Encoding



(a) Register-operand format



(b) Immediate-operand format



(c) Call format

Figure 5.12 Instruction encoding.

Add Instructions (2)

■ Example: Add R3, R4, R5 (ctd.)

□ Sequence of Actions

Step	Action
1	Memory address \leftarrow [PC], Read memory, IR \leftarrow Memory data, PC \leftarrow [PC] + 4
2	Decode instruction, RA \leftarrow [R4], RB \leftarrow [R5]
3	RZ \leftarrow [RA] + [RB]
4	RY \leftarrow [RZ]
5	R3 \leftarrow [RY]

Figure 5.11 Sequence of actions needed to fetch and execute the instruction: Add R3, R4, R5.

Add Instructions (3)

- Example: Add R3, R4, R5 (ctd.)

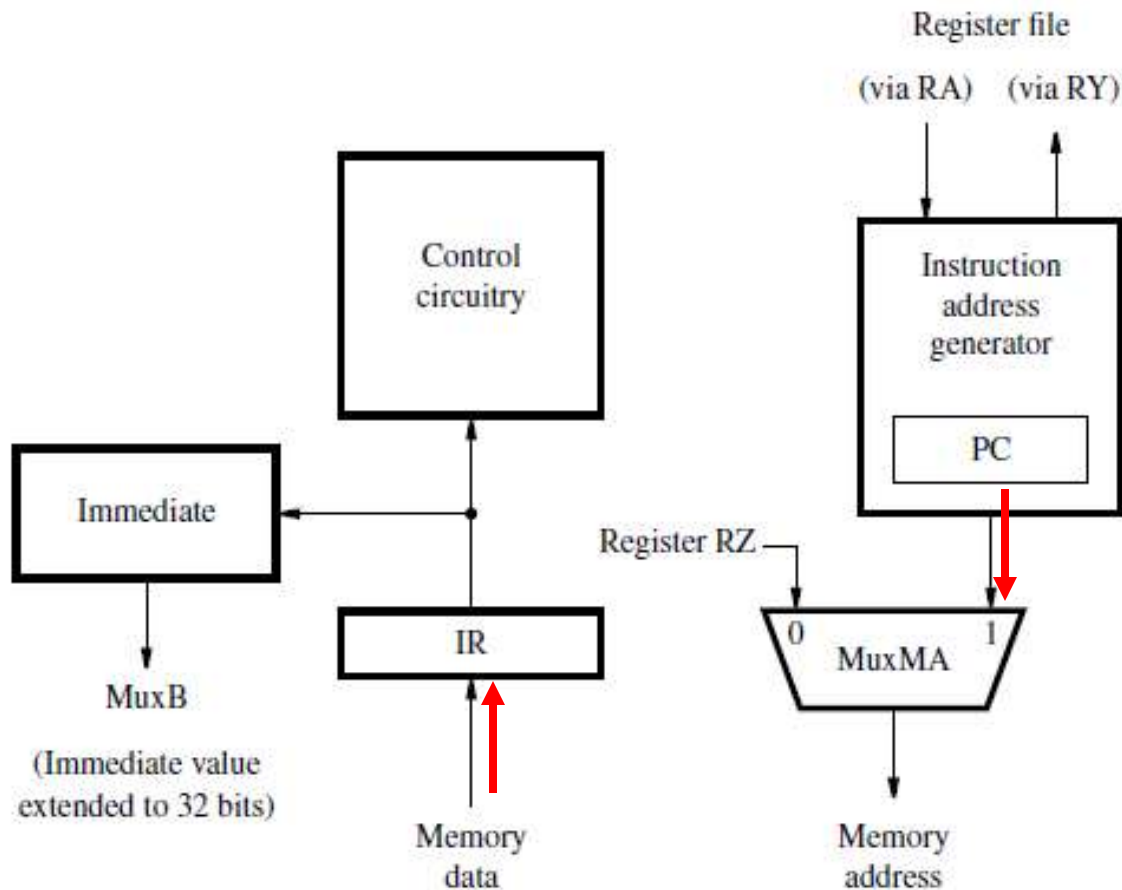
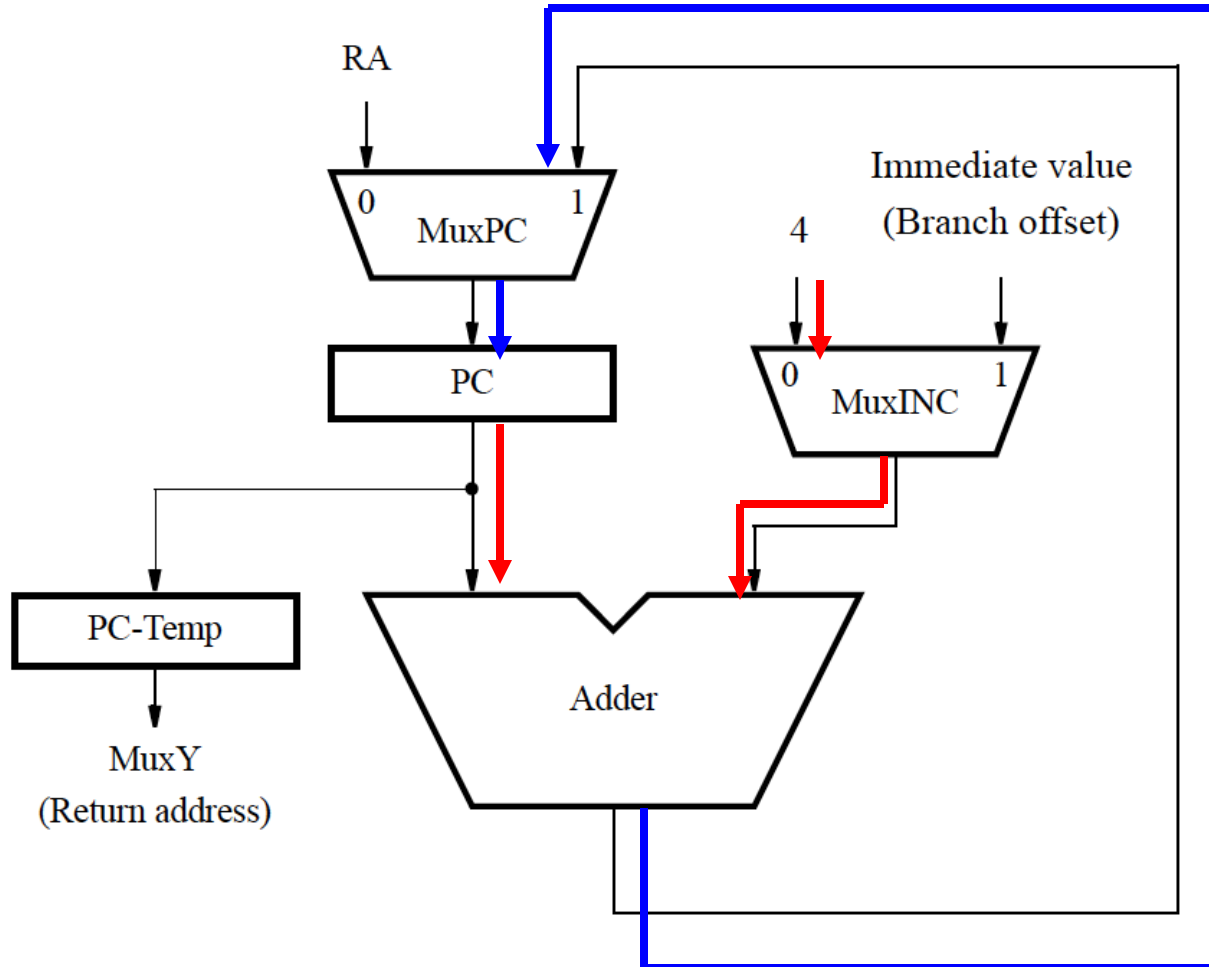


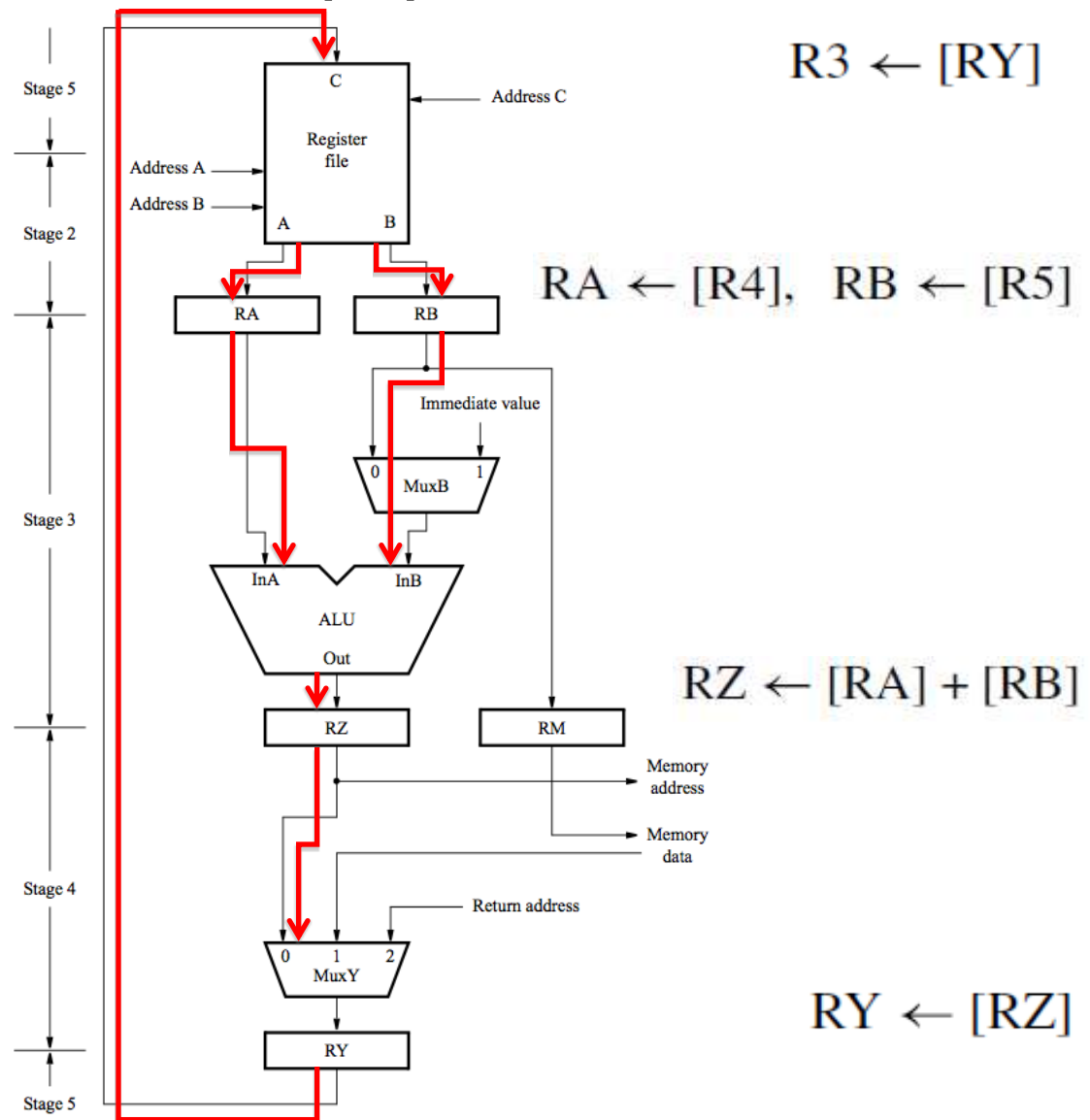
Figure 5.9 Instruction fetch section of Figure 5.7.

Add Instructions (4)

- Example: Add R3, R4, R5 (ctd.)



Add Instructions (5)



Load Instructions (1)

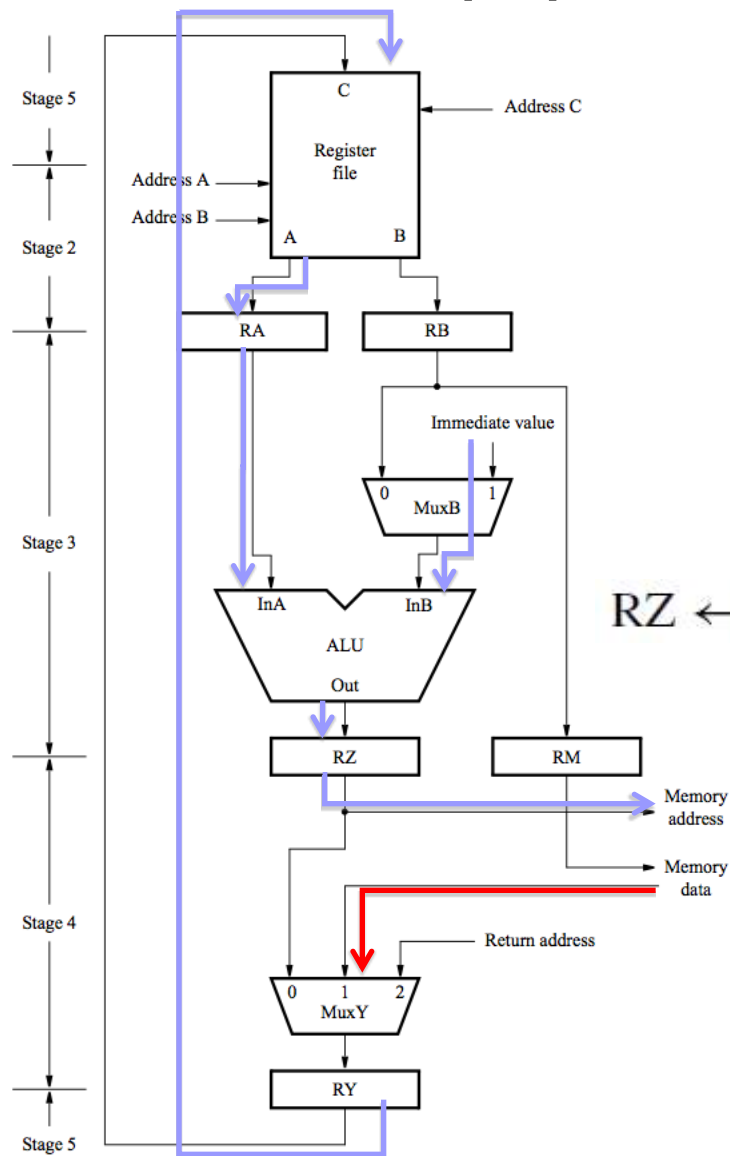
■ Example: Load R5, X(R7)

□ Sequence of Actions

Step	Action
1	Memory address \leftarrow [PC], Read memory, IR \leftarrow Memory data, PC \leftarrow [PC] + 4
2	Decode instruction, RA \leftarrow [R7]
3	RZ \leftarrow [RA] + Immediate value X
4	Memory address \leftarrow [RZ], Read memory, RY \leftarrow Memory data
5	R5 \leftarrow [RY]

Figure 5.13 Sequence of actions needed to fetch and execute the instruction: Load R5, X(R7).

Load Instructions (2)



$R5 \leftarrow [RY]$

$RA \leftarrow [R7]$

$RZ \leftarrow [RA] + \text{Immediate value } X$

Memory address $\leftarrow [RZ]$,

$RY \leftarrow \text{Memory data}$

Store Instructions (1)

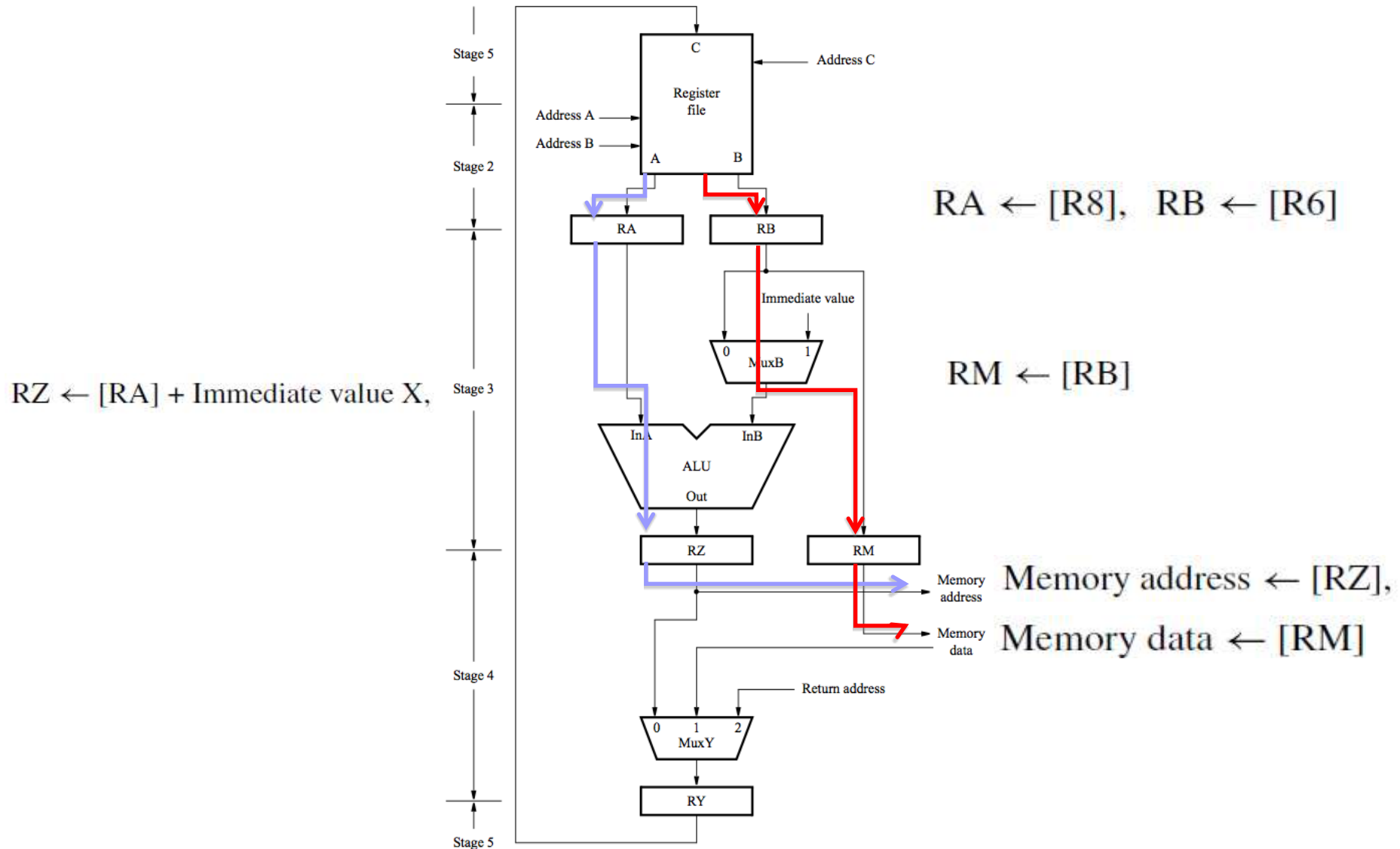
■ Example: Store R6, X(R8)

□ Sequence of Actions

Step	Action
1	Memory address \leftarrow [PC], Read memory, IR \leftarrow Memory data, PC \leftarrow [PC] + 4
2	Decode instruction, RA \leftarrow [R8], RB \leftarrow [R6]
3	RZ \leftarrow [RA] + Immediate value X, RM \leftarrow [RB]
4	Memory address \leftarrow [RZ], Memory data \leftarrow [RM], Write memory
5	No action

Figure 5.14 Sequence of actions needed to fetch and execute the instruction: Store R6, X(R8).

Store Instructions (2)

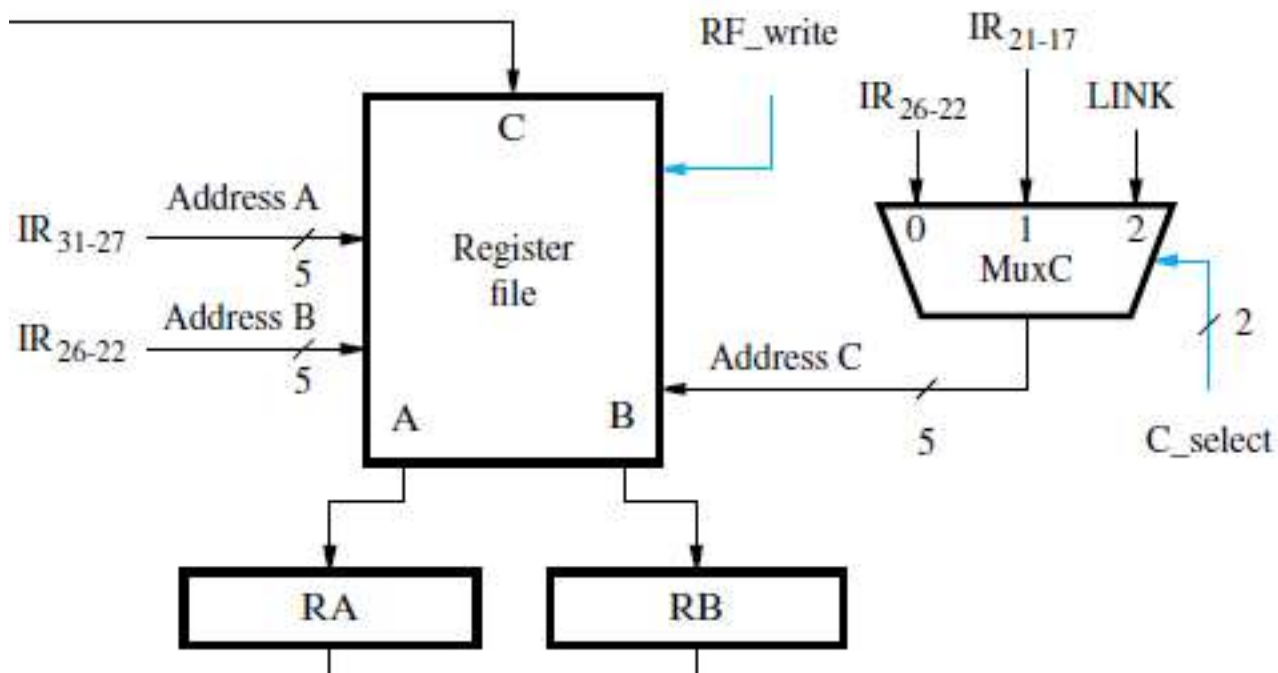


Some Observations (1)

- Memory Read and Write operations can be completed in one clock cycle?
 - A memory Read or Write operation can be completed in one clock cycle when the data involved are available in the cache
- We assume that the processor reads the source registers of the instruction in step 2, while it is still decoding the OP code of the instruction that has just been loaded into the IR. Can these two tasks be completed in the same step? How can the control hardware know which registers to read before it completes decoding the instruction?
 - Source register addresses are specified using the same bit positions in all instructions.

Some Observations (2)

- Part of Figure 5.18



Branching

■ Straight-line Program Execution

- $PC \leftarrow [PC] + 4$

■ Branch Instructions

- A branch offset given as an immediate value in the instruction is added to the current contents of the PC.
- The range of addresses that can be reached by a branch instruction is limited.

■ Subroutine Call Instructions

- Subroutine call instructions can reach a larger range of addresses.
- Most RISC-style computers have Jump and Call instructions that use a general-purpose register to specify a full 32-bit address.

Branch Instructions (1)

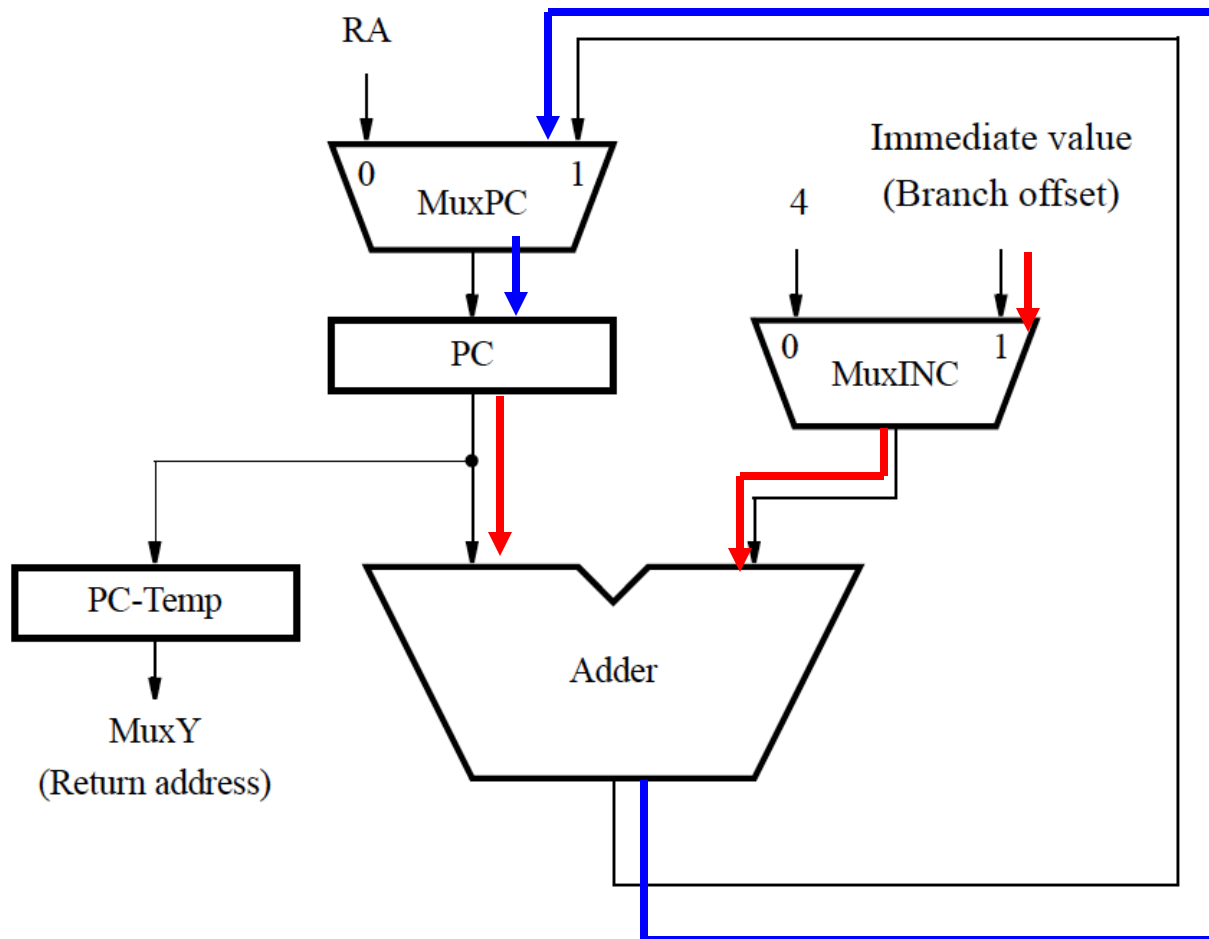
■ Unconditional Branch Instructions

Step	Action
1	Memory address \leftarrow [PC], Read memory, IR \leftarrow Memory data, PC \leftarrow [PC] + 4
2	Decode instruction
3	PC \leftarrow [PC] + Branch offset
4	No action
5	No action

Figure 5.15 Sequence of actions needed to fetch and execute an unconditional branch instruction.

Branch Instructions (2)

- Unconditional Branch Instructions (ctd.)



Branch Instructions (3)

■ Conditional Branch Instructions

□ Example: Branch_if_[R5]=[R6] LOOP

Step	Action
1	Memory address \leftarrow [PC], Read memory, IR \leftarrow Memory data, PC \leftarrow [PC] + 4
2	Decode instruction, RA \leftarrow [R5], RB \leftarrow [R6]
3	Compare [RA] to [RB], If [RA] = [RB], then PC \leftarrow [PC] + Branch offset
4	No action
5	No action

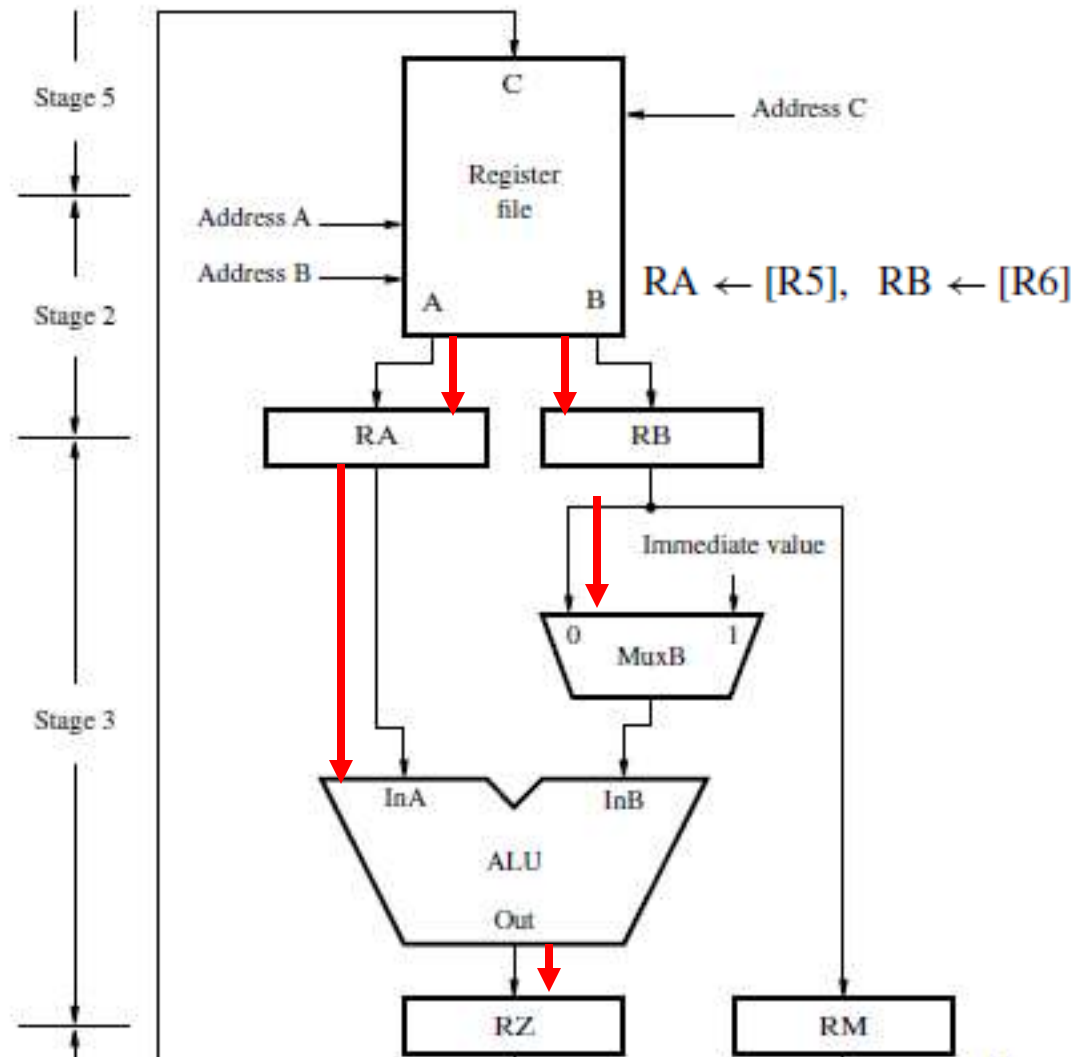
Figure 5.16 Sequence of actions needed to fetch and execute the instruction: Branch_if_[R5]=[R6] LOOP.

Branch Instructions (4)

■ Conditional Branch Instructions (ctd.)

□ Example (ctd.)

- Comparison can be done by $[R5] - [R6]$ in the ALU



Branch Instructions (5)

■ Conditional Branch Instructions (ctd.)

□ Example (ctd.)

- A subtraction operation in the ALU is time consuming, and is not needed in this case.
- A simpler and faster comparator circuit (a part of ALU) can examine the contents of registers RA and RB and produce the required condition signals.

Branch Instructions (6)

- Conditional Branch Instructions (ctd.)

- Example 5.3 (P186) : Derive the logic expressions for a circuit that compares two unsigned numbers:

$X = x_2x_1x_0$ and $Y = y_2y_1y_0$ and generates three outputs: XGY , XEY , and $XL Y$. One of these outputs is set to 1 to indicate that X is greater than, equal to, or less than Y , respectively.

Branch Instructions (7)

■ Conditional Branch Instructions (ctd.)

- Solution: To compare two unsigned numbers, we need to compare individual bit locations, starting with the most significant bit. If $x_2 = 1$ and $y_2 = 0$, then X is greater than Y . If $x_2 = y_2$, then we need to compare the next lower bit location, and so on. Thus, the logic expressions for the three outputs may be written as follows:

$$XGY = x_2\bar{y}_2 + \overline{(x_2 \oplus y_2)} \cdot (x_1\bar{y}_1 + \overline{(x_1 \oplus y_1)} x_0\bar{y}_0)$$

$$XEY = \overline{(x_2 \oplus y_2)} \cdot \overline{(x_1 \oplus y_1)} \cdot \overline{(x_0 \oplus y_0)}$$

$$XLY = \overline{XGY + XEY}$$

Recall: 2.7 Subroutine

■ Subroutine Linkage

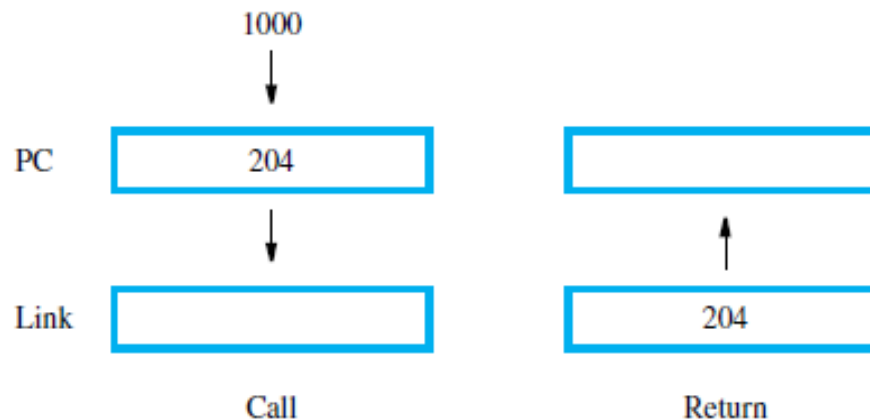
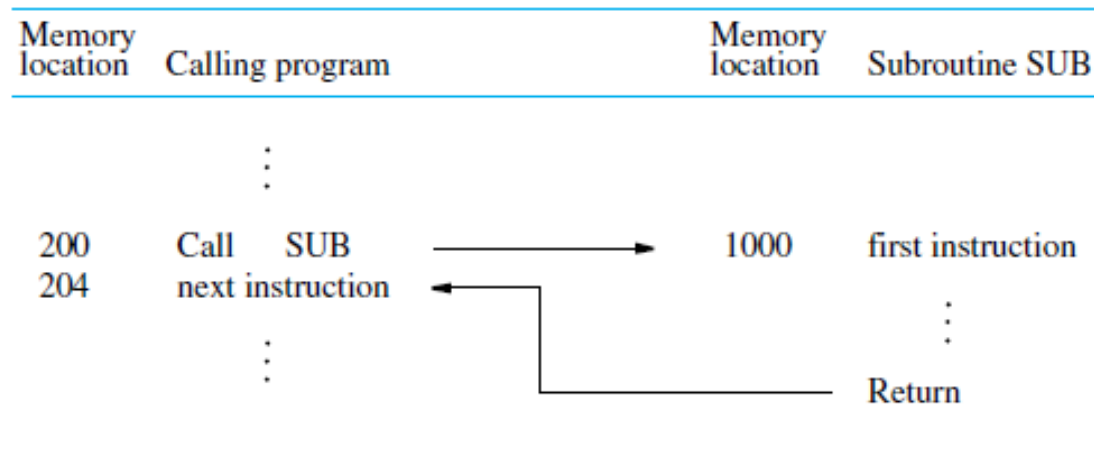


Figure 2.16 Subroutine linkage using a link register.

Subroutine Call Instructions (1)

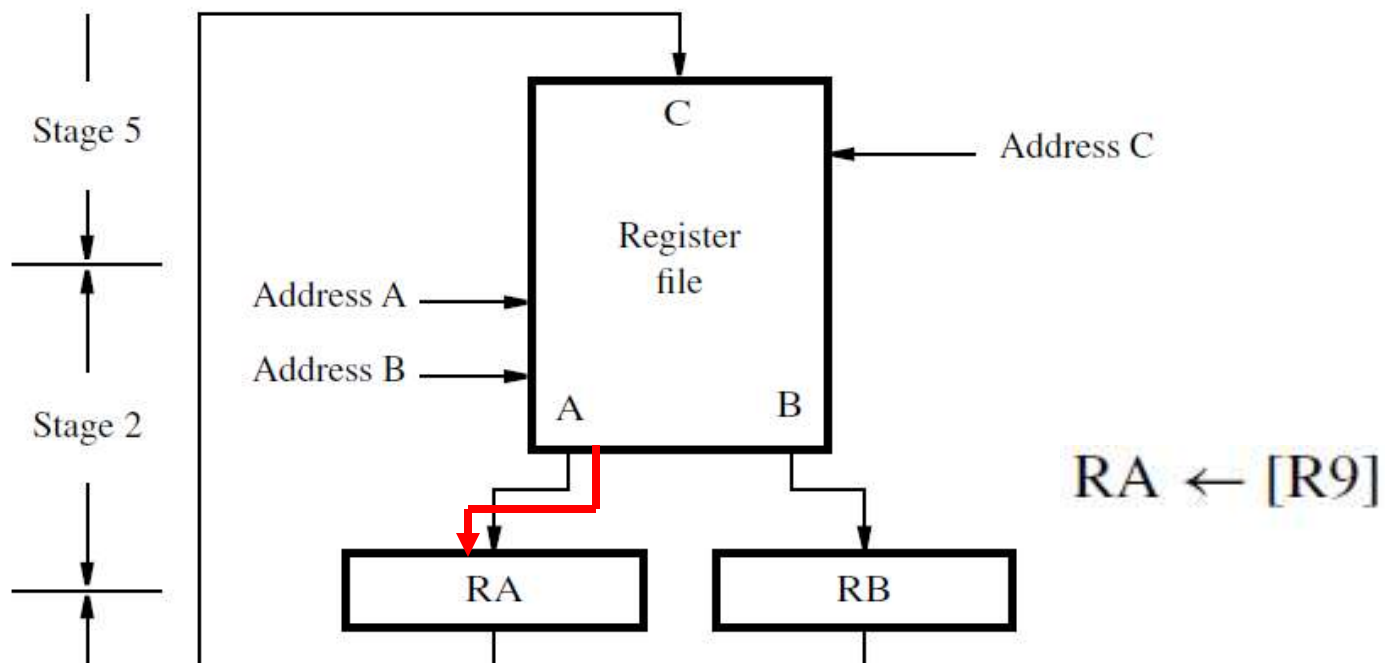
■ Example: Call_Register R9

Step	Action
1	Memory address \leftarrow [PC], Read memory, IR \leftarrow Memory data, PC \leftarrow [PC] + 4
2	Decode instruction, RA \leftarrow [R9]
3	PC-Temp \leftarrow [PC], PC \leftarrow [RA]
4	RY \leftarrow [PC-Temp]
5	Register LINK \leftarrow [RY]

Figure 5.17 Sequence of actions needed to fetch and execute the instruction: Call_Register R9.

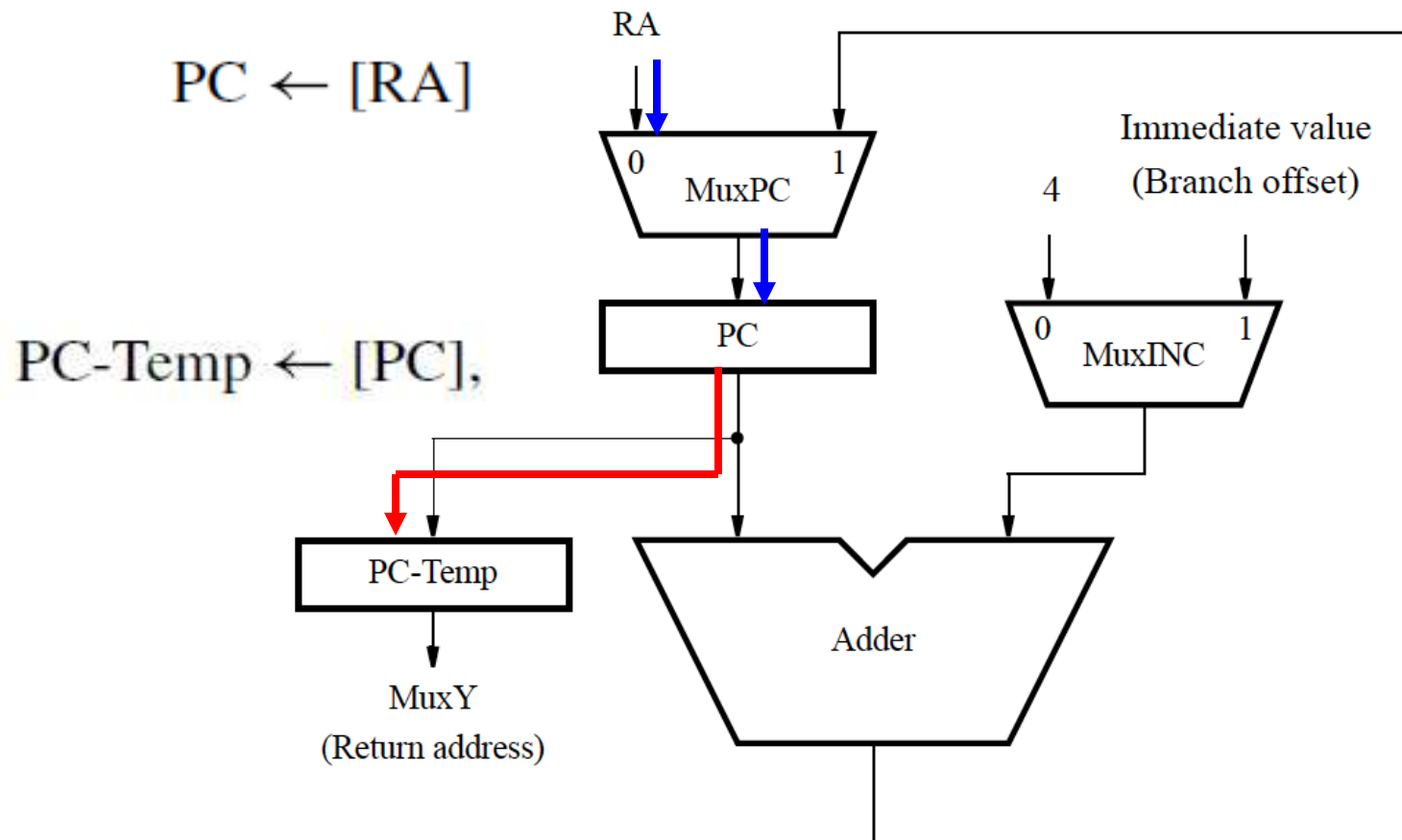
Subroutine Call Instructions (3)

- Example: Call_Register R9 (ctd.)



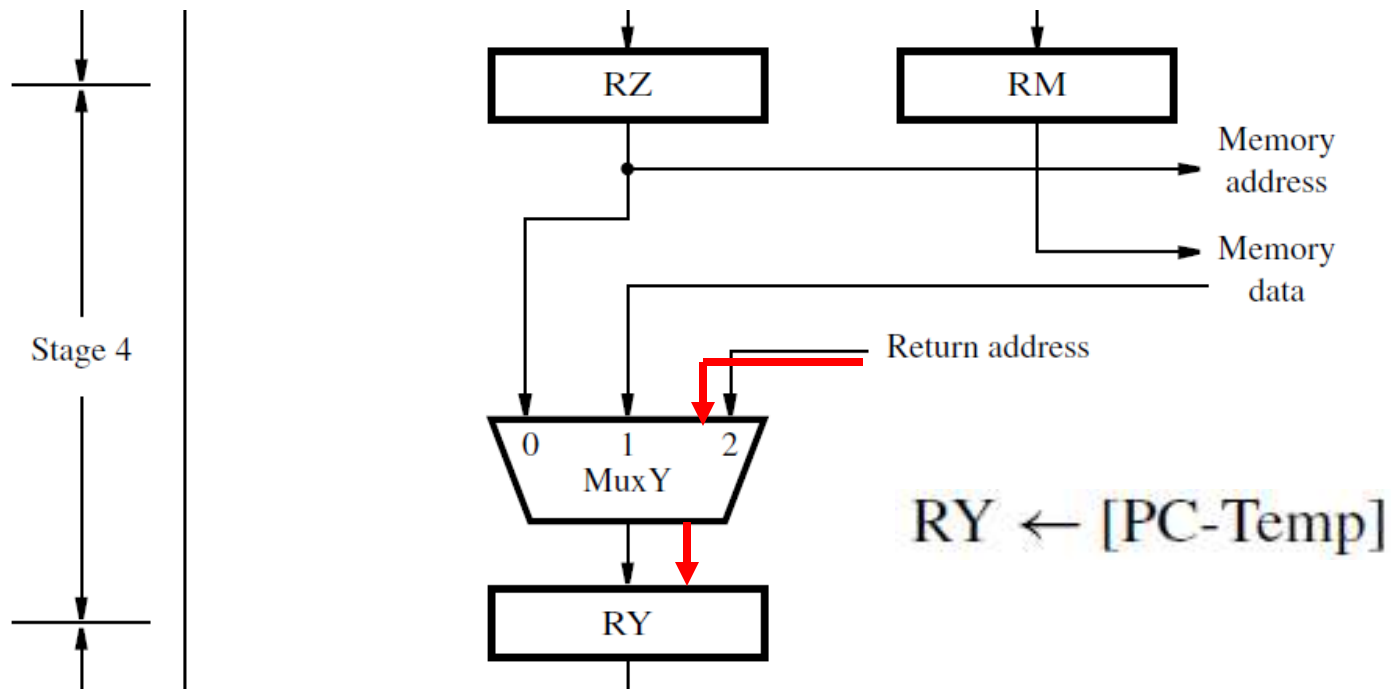
Subroutine Call Instructions (4)

- Example: Call_Register R9 (ctd.)



Subroutine Call Instructions (4)

- Example: Call_Register R9 (ctd.)



Waiting For Memory (1)

- Cache memory described earlier as faster and smaller storage that is an adjunct to the larger and slower main memory.
- When data are found in the cache, access to memory can be completed in one clock cycle.
- Otherwise, read and write operations may require several clock cycles to load data from main memory into the cache.
- A control signal is needed to indicate that memory function has been completed (MFC).

Waiting For Memory (2)

- E.g., for step 1 :

1. Memory address \leftarrow [PC], Read memory, Wait for MFC, IR \leftarrow Memory data, PC \leftarrow [PC] + 4

- E.g., for step 4 of Load instruction

4. Memory address \leftarrow [RZ], Read memory, Wait for MFC, RY \leftarrow Memory data

- E.g., for step 4 of Store instruction

4. Memory address \leftarrow [RZ], Memory data \leftarrow [RM], Write memory, Wait for MFC

Summary

- 知识点: Execution steps of an instruction
- 掌握程度
 - 给出数据通路图, 能够写出Add、Load、Store、Branch、Subroutine Call指令的执行步骤。

Homework

- 自学P186 Example 5.4
- P188 5.4