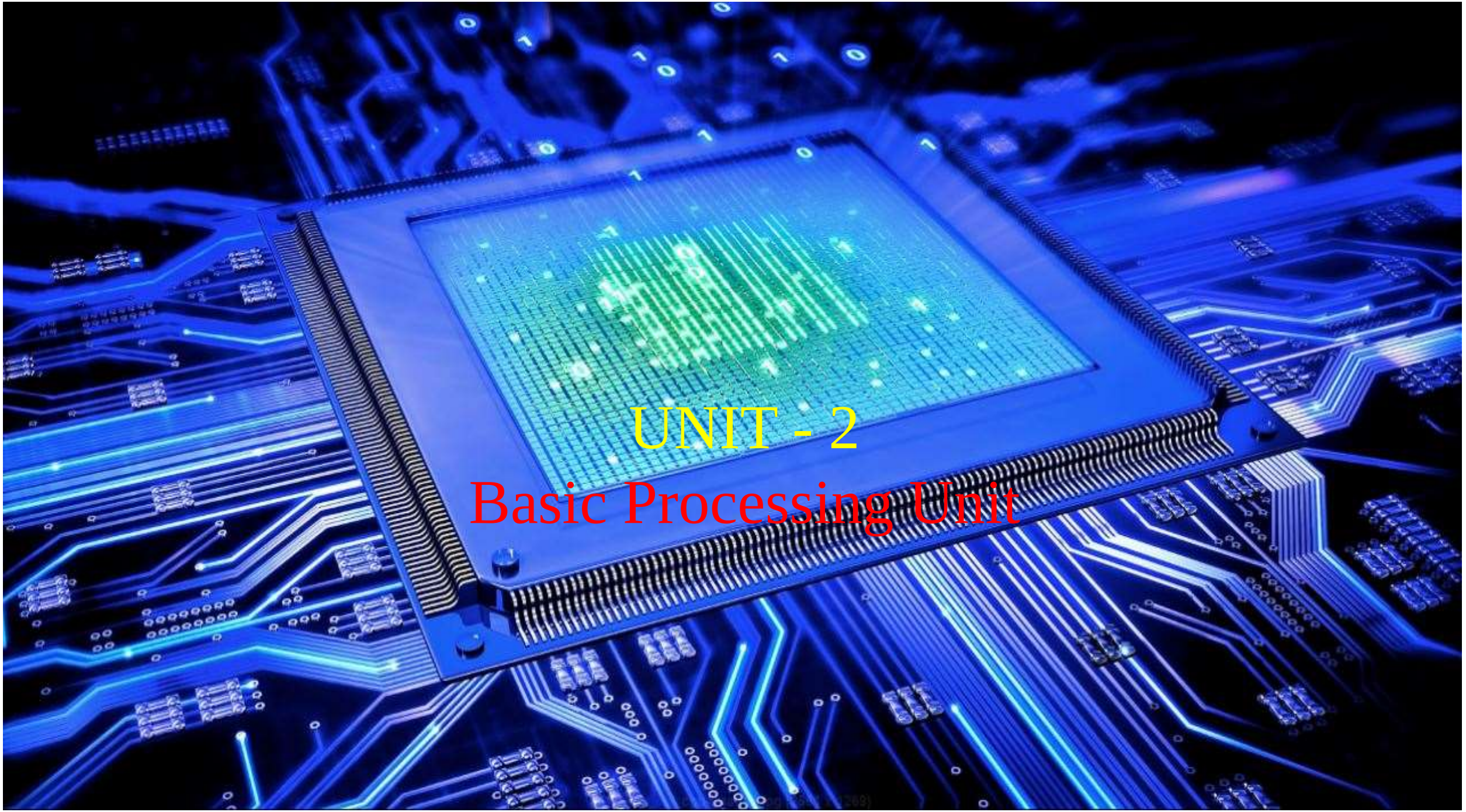

UNIT - 2
Basic Processing Unit

Basic Processing Unit

2.1 Some Fundamental Concepts

2.2 Instruction Execution

- 2.2.1 Load Instructions
- 2.2.2 Arithmetic and Logic Instructions
- 2.2.3 Store Instructions

2.3 Hardware Components

- 2.3.1 Register File
- 2.3.2 ALU
- 2.3.3 Data path
- 2.3.4 Instruction Fetch Section

2.4 Instruction Fetch and Execution Steps

- 2.4.1 Branching
- 2.4.2 Waiting for Memory

2.5 Control Signals

2.6 Hardwired Control

- 2.6.1 Data path Control Signals
- 2.6.2 Dealing with Memory Delay

# Some Fundamental Concepts

- A typical computing task consists of a series of operations specified by a sequence of machine-language instructions that constitute a program. The processor fetches one instruction at a time and performs the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- The processor uses the program counter, PC, to keep track of the address of the next instruction to be fetched and executed. After fetching an instruction, the contents of the PC are updated to point to the next instruction in sequence.
- A branch instruction may cause a different value to be loaded into the PC.
- When an instruction is fetched, it is placed in the instruction register, IR, from where it is interpreted, or decoded, by the processor's control circuitry.
- The IR holds the instruction until its execution is completed.
- Consider a 32-bit computer in which each instruction is contained in one word in the memory, as in RISC-style instruction set architecture.
- 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 the instruction to be executed; hence they are loaded into the IR. In register transfer notation, the required action is

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

2. Increment the PC to point to the next instruction. Assuming that the memory is byte addressable, the PC is incremented by 4; that is

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

3. Carry out the operation specified by the instruction in the IR.

In cases where an instruction occupies more than one word, steps 1 and 2 must be repeated as many times as necessary to fetch the complete instruction. These two steps are usually referred to as the fetch phase; step 3 constitutes the execution phase.

With few exceptions, the operation specified by an instruction can be carried out by performing one or more of the following actions:

- Read the contents of a given memory location and load them into a processor register.
- Read data from one or more processor registers.
- Perform an arithmetic or logic operation and place the result into a processor register.
- Store data from a processor register into a given memory location.

The hardware components needed to perform these actions are shown in Figure

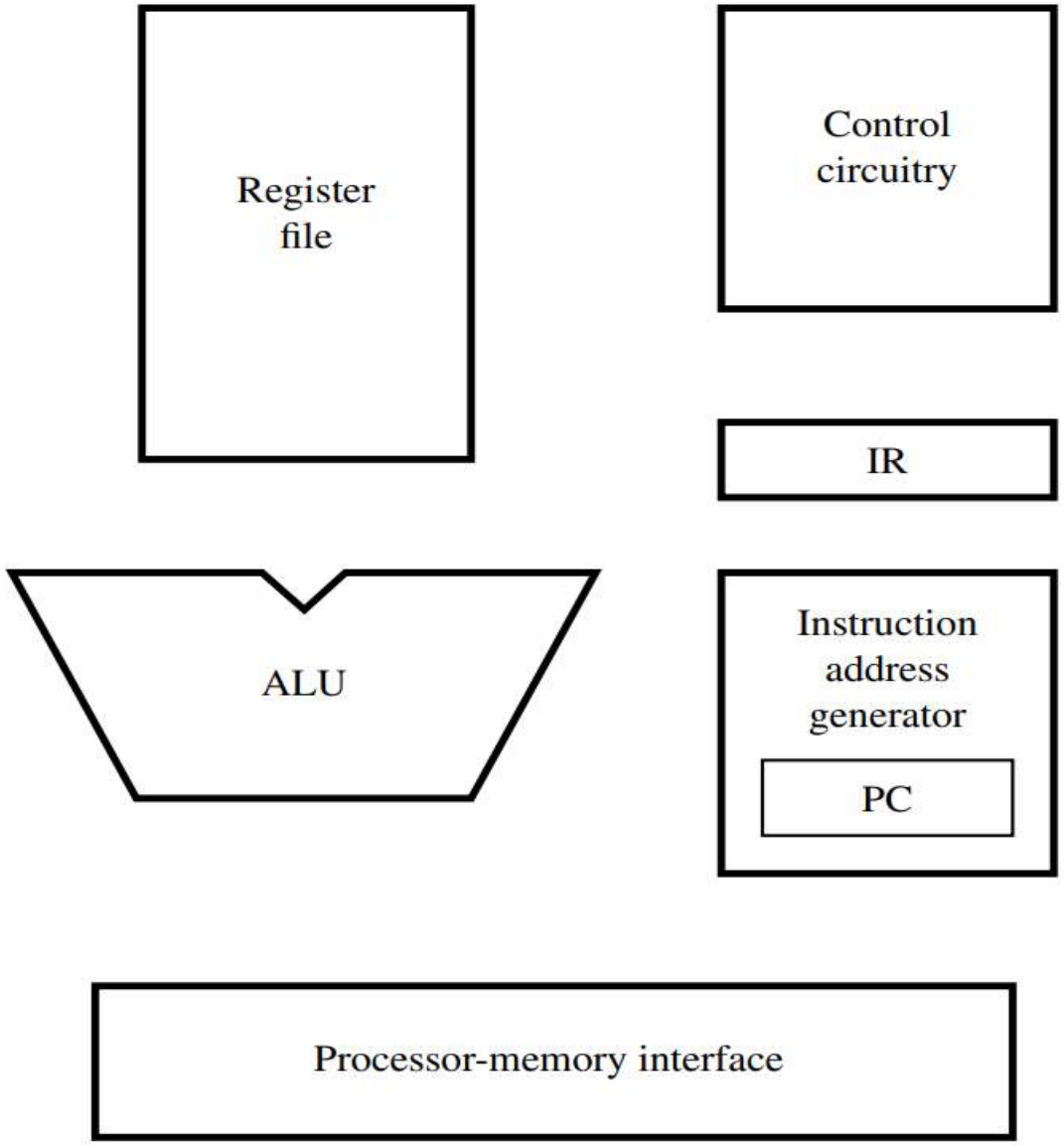



**Figure 5.1** Main hardware components of a processor.

- The processor communicates with the memory through the processor-memory interface, which transfers data from and to the memory during Read and Write operations.
- The instruction address generator updates the contents of the PC after every instruction is fetched.
- The register file is a memory unit whose storage locations are organized to form the processor's general-purpose registers.
- During execution, the contents of the registers named in an instruction that performs an arithmetic or logic operation are sent to the arithmetic and logic unit (ALU), which performs the required computation.
- The results of the computation are stored in a register in the register file.

## Data Processing Hardware

A typical computation operates on data stored in registers. These data are processed by combinational circuits, such as adders, and the results are placed into a register. Figure illustrates this structure.

A clock signal is used to control the timing of data transfers. The registers comprise edge-triggered flip-flops into which new data are loaded at the active edge of the clock

we assume that the rising edge of the clock is the active edge. The clock period, which is the time between two successive rising edges, must be long enough to allow the combinational circuit to produce the correct result.

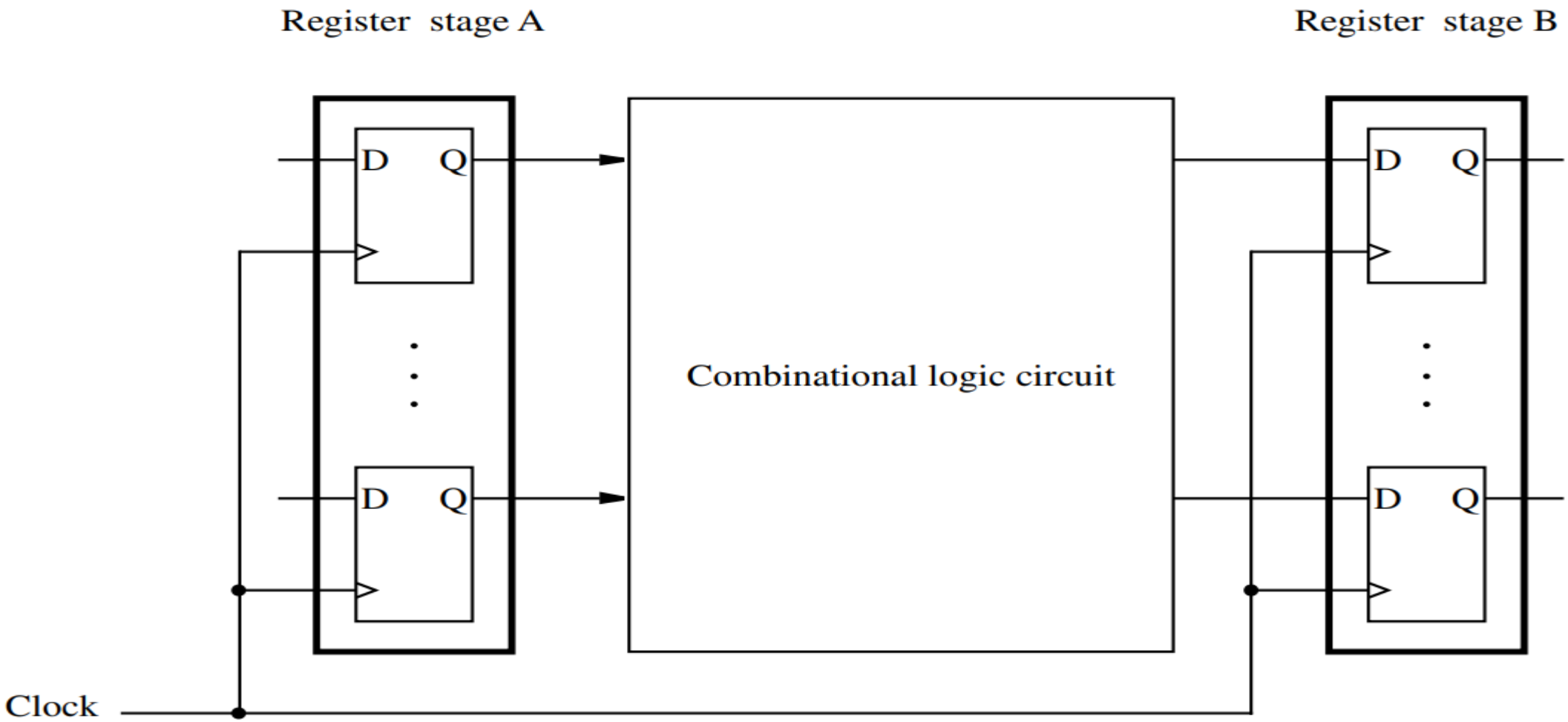
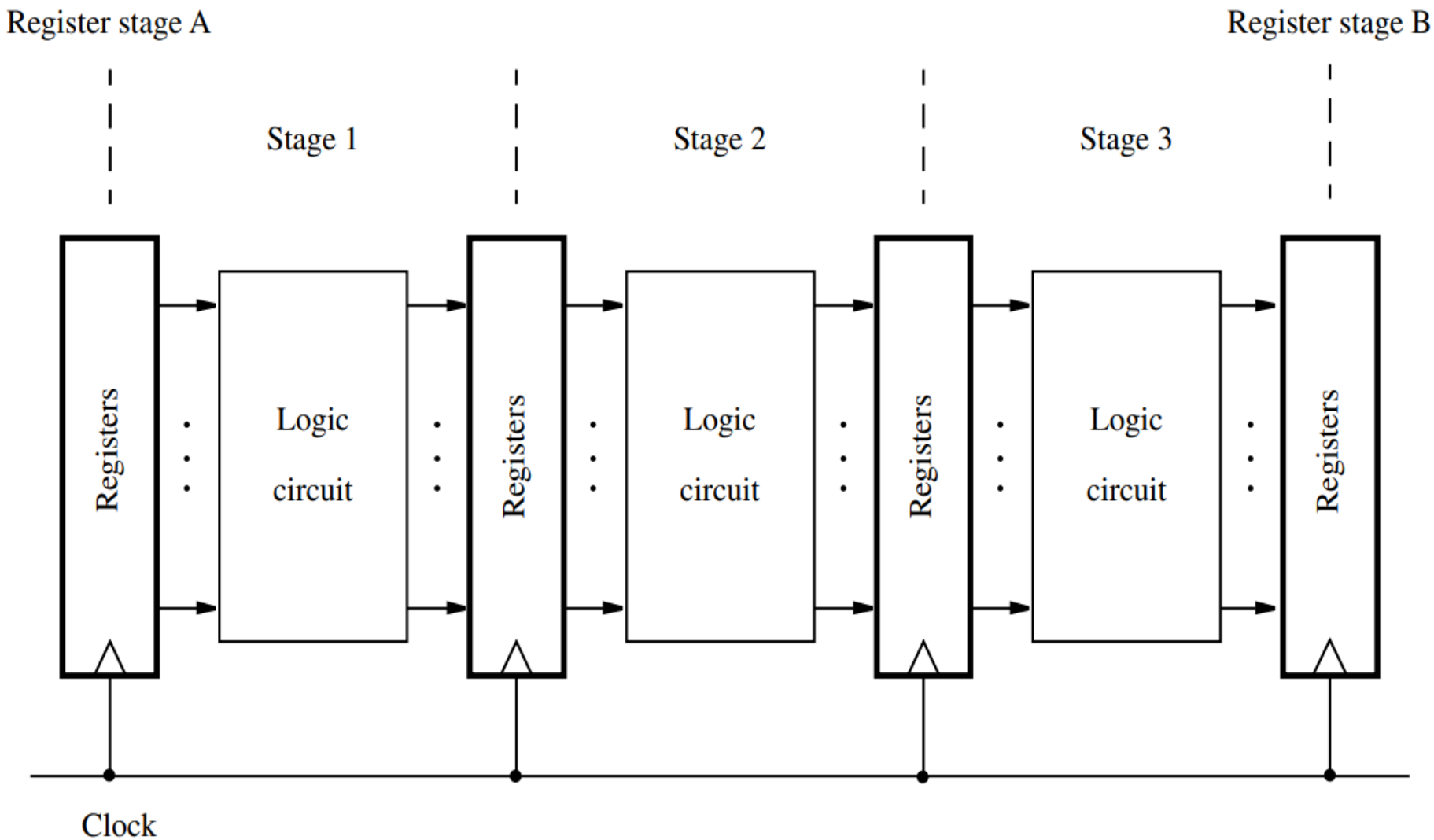



**Figure 5.2** Basic structure for data processing.


Register stage A
Register stage B
Stage 1
Stage 2
Stage 3
Registers
Logic
circuit
Registers
Logic
circuit
Registers
Logic
circuit
Registers
Clock

# Instruction Execution

## Load Instructions

Consider the instruction

Load R5, X(R7)

which uses the Index addressing mode to load a word of data from memory location X + [R7] into register R5.

Execution of this instruction involves the following actions:

- 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

In this case, fetching and executing the Load instruction above can be completed as follows:

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

Instructions that involve an arithmetic or logic operation can be executed using similar steps. They differ from the Load instruction in two ways:

- There are either two source registers, or a source register and an immediate source operand.
- No access to memory operands is required.

A typical instruction of this type is

Add R3, R4, R5

It requires the following steps:

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. Load the result into the destination register, R3

The Add instruction would then be performed as follows:

1. Fetch the instruction and increment the program counter.

2. Decode the instruction and read registers R4 and R5.

3. Compute the sum [R4] + [R5]

4. No action.

5. Load the result into the destination register, R3.

If the instruction uses an immediate operand, as in Add R3, R4, #1000

the immediate value is given in the instruction word.

Once the instruction is loaded into the IR, the immediate value is available for use in the addition operation.

The same five-step sequence can be used, with steps 2 and 3 modified as:

2. Decode the instruction and read register R4.

3. Compute the sum [R4] + 1000.

## Store Instructions

The five-step sequence used for the Load and Add instructions is also suitable for Store instructions, except that the final step of loading the result into a destination register is not required. The hardware stage responsible for this step takes no action.

For example, the instruction Store R6, X(R8) stores the contents of register R6 into memory location X + [R8].

It can be implemented as follows:

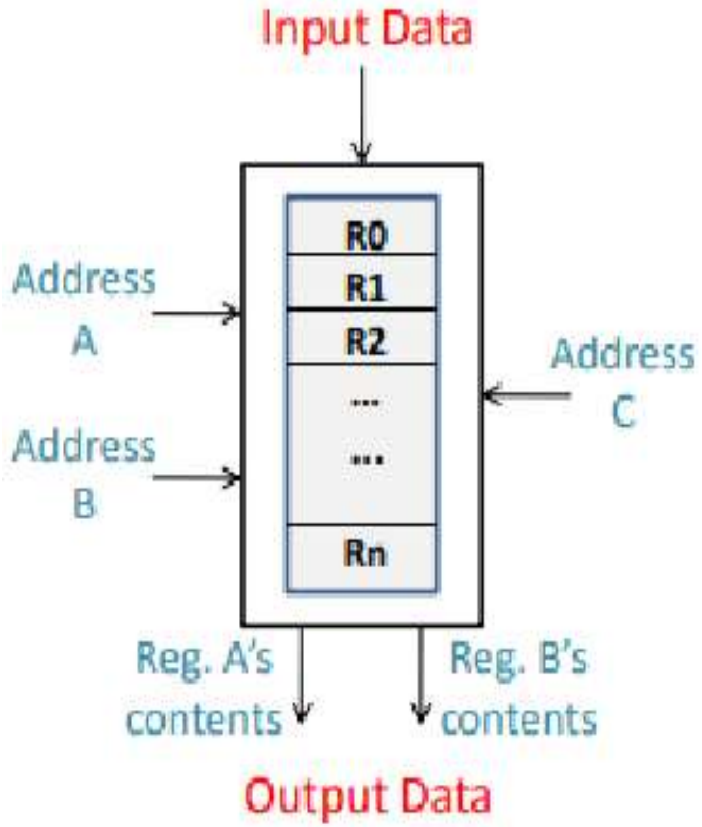
1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read registers R6 and R8.
3. Compute the effective address X + [R8].
4. Store the contents of register R6 into memory location X + [R8].
5. No action

| 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. |

# Hardware Components

## 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 and 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

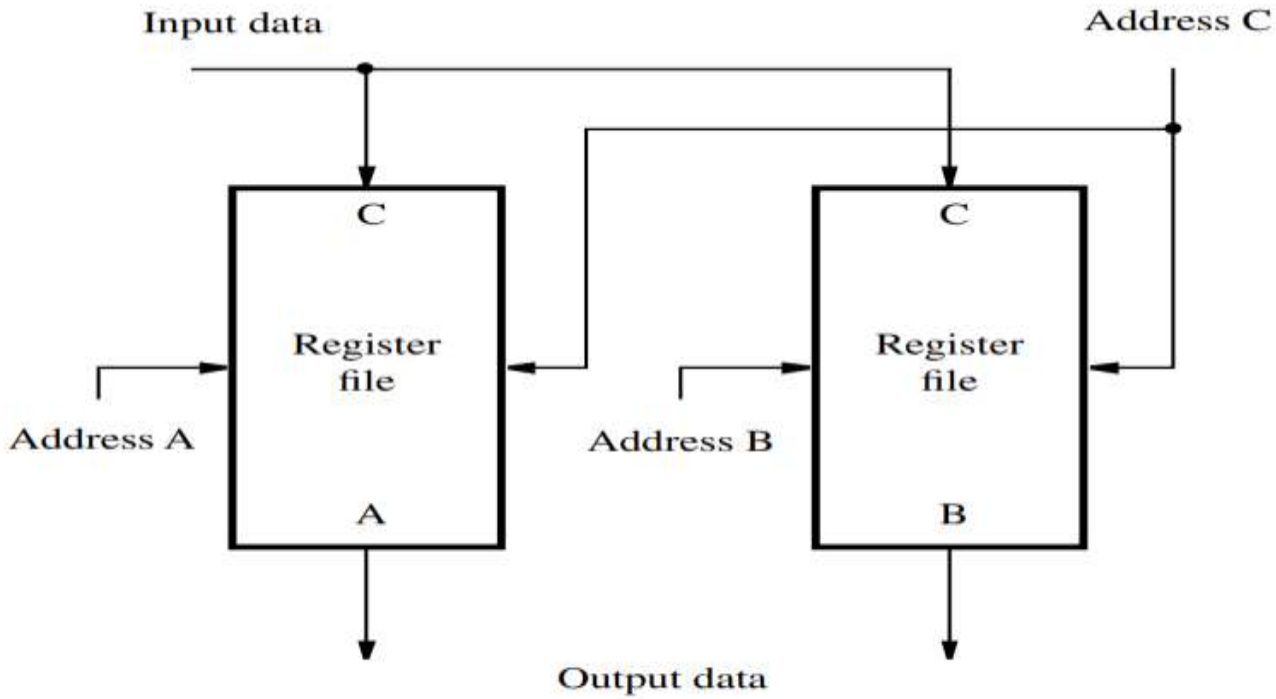
A memory unit with two output ports is said to be dual ported

• 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




(b) Two memory blocks

# ALU

Arithmetic and Logic Unit (ALU) performs:

– Arithmetic operations (addition, 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

• 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

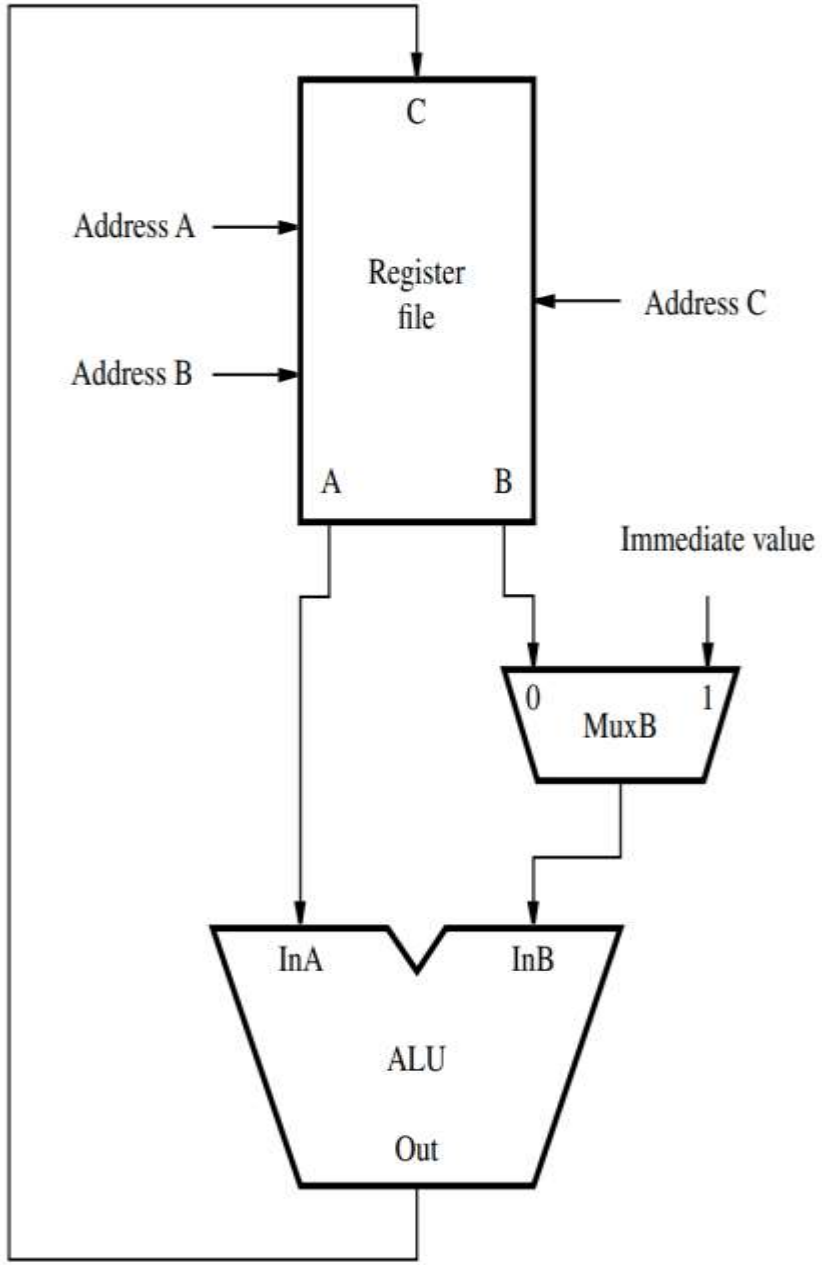



**Figure 5.6** Conceptual view of the hardware needed for computation.

# Datapath

Instruction processing consists of two phases: the fetch phase and the execution phase.

It is convenient to divide the processor hardware into two corresponding sections. One section fetches instructions and the other executes them.

– Front end (fetch section): Includes the fetch and decode stages

The section that fetches instructions is also responsible for decoding them and for generating the control signals that cause appropriate actions to take place in the execution section.

– Back end (execute section): Includes the ALU, Memory Access and Writeback stages

The execution section reads the data operands specified in an instruction, performs the required computations, and stores the results.

Datapath is a combination of register read, back-end stages and inter-stage registers

We need to organize the hardware into a multi-stage structure with stages corresponding to the five steps. A possible structure is shown in Figure.

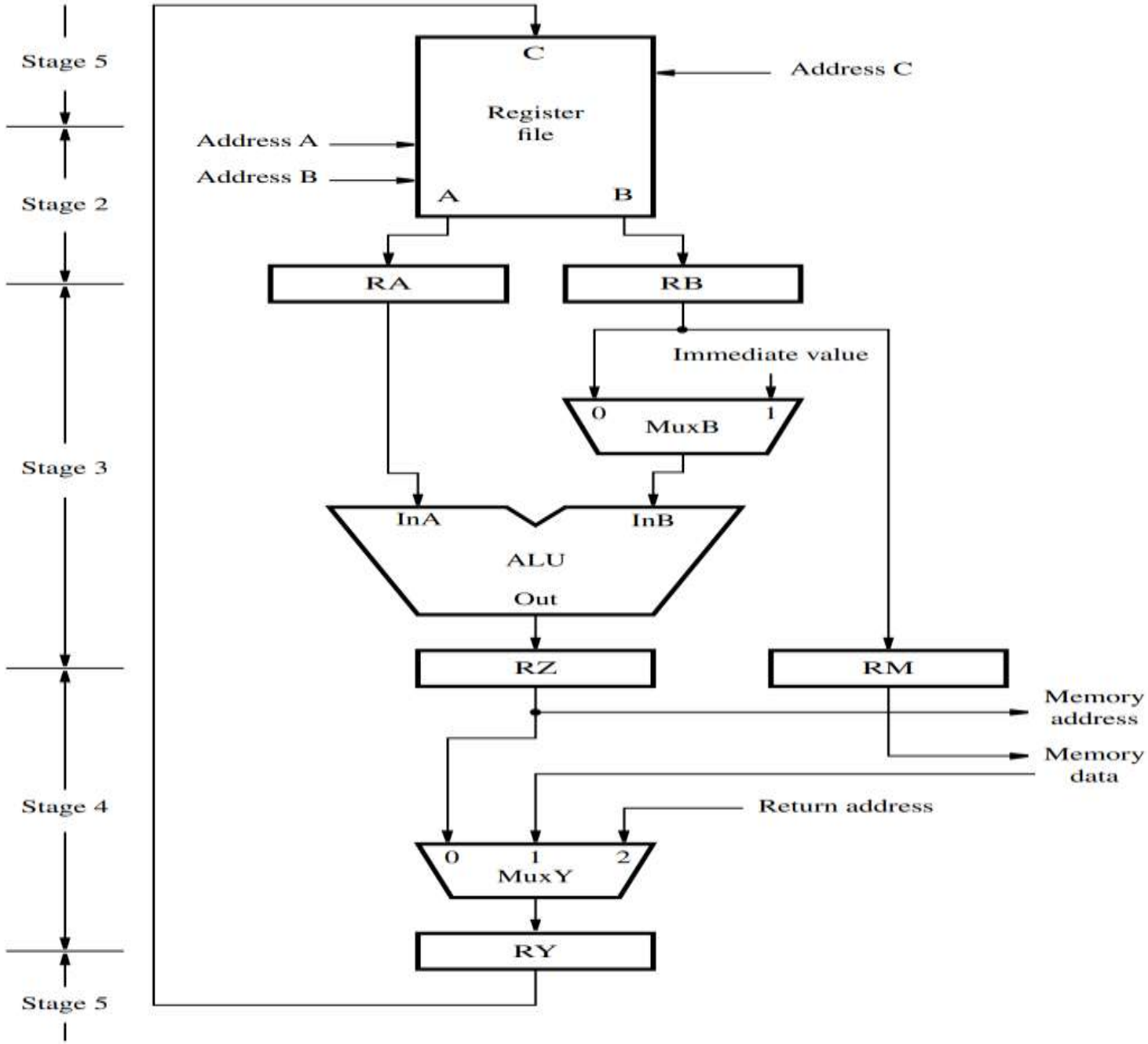
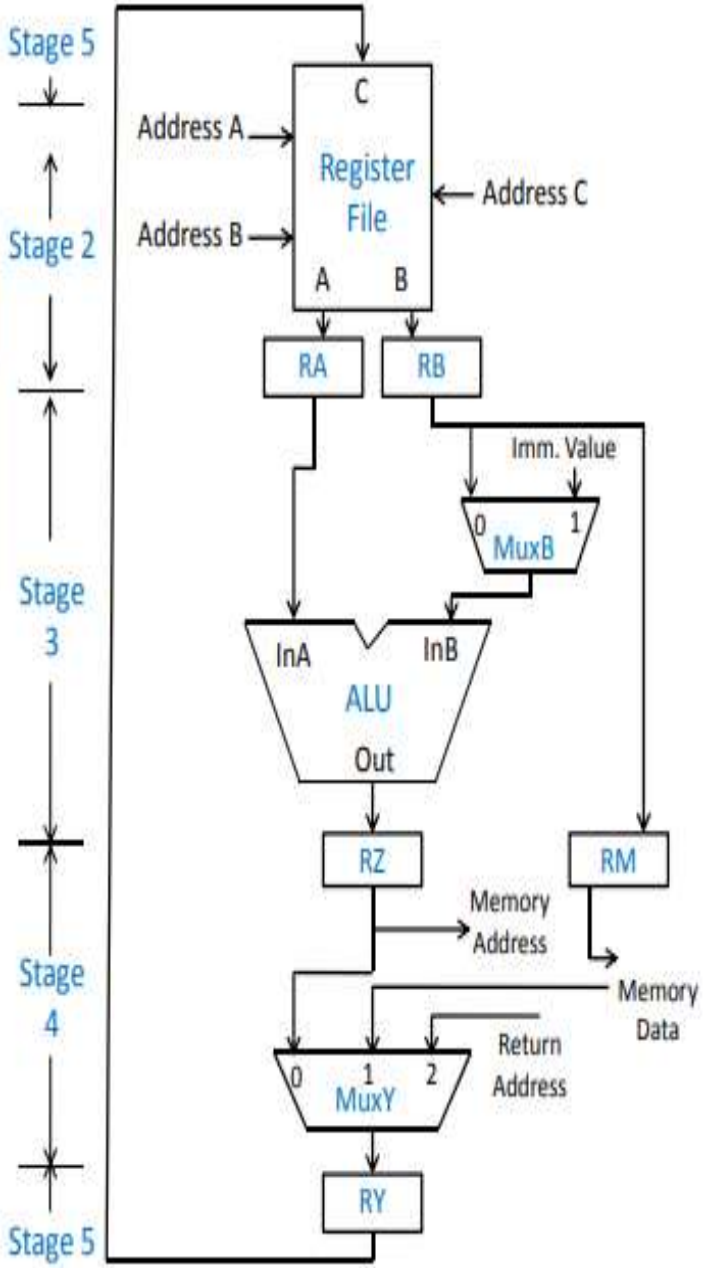



**Figure 5.8** Datapath in a processor.

## Arithmetic/Logic Instructions:

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

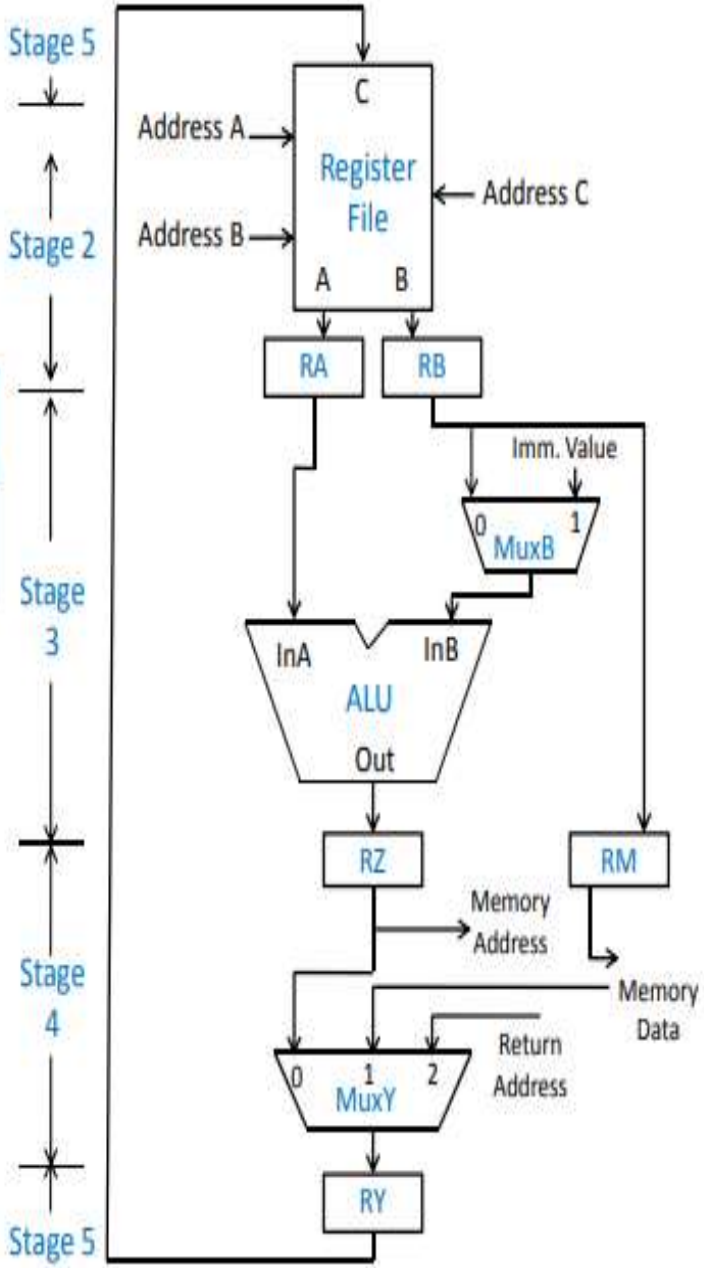
- Stage 3: ALU carries out the desired arithmetic/logic operation on the source operands , result is loaded into RZ
- Stage 4: Result computed in stage-3 moves from RZ to RY
- Stage 5: Data transferred from RY into destination




## Load Instructions:

| Instruction Type | MuxB selection | MuxY selection |
|---|---|---|
| Load | 1 | 1 |

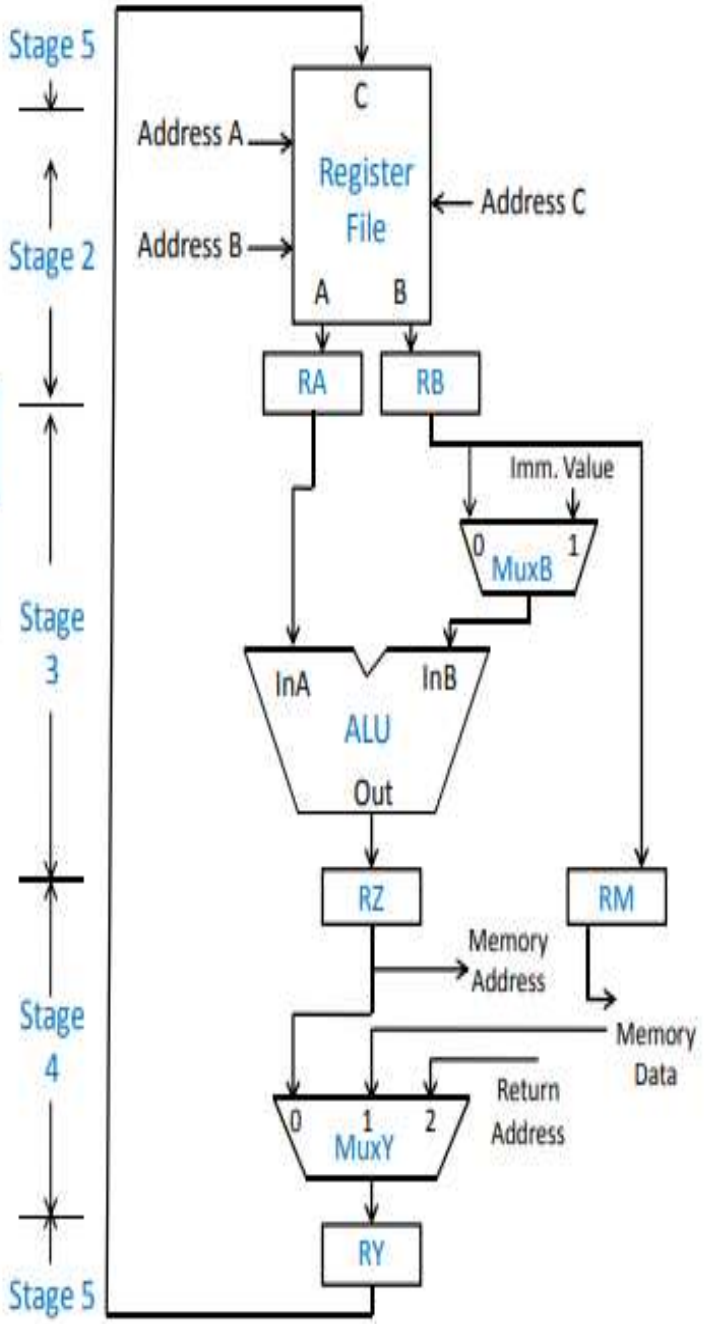
- Stage 3: Effective address computed and loaded into RZ
- Stage 4: Address sent from RZ to memory, data returned by memory loaded into RY
- Stage 5: Data transferred from RY into destination



## 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

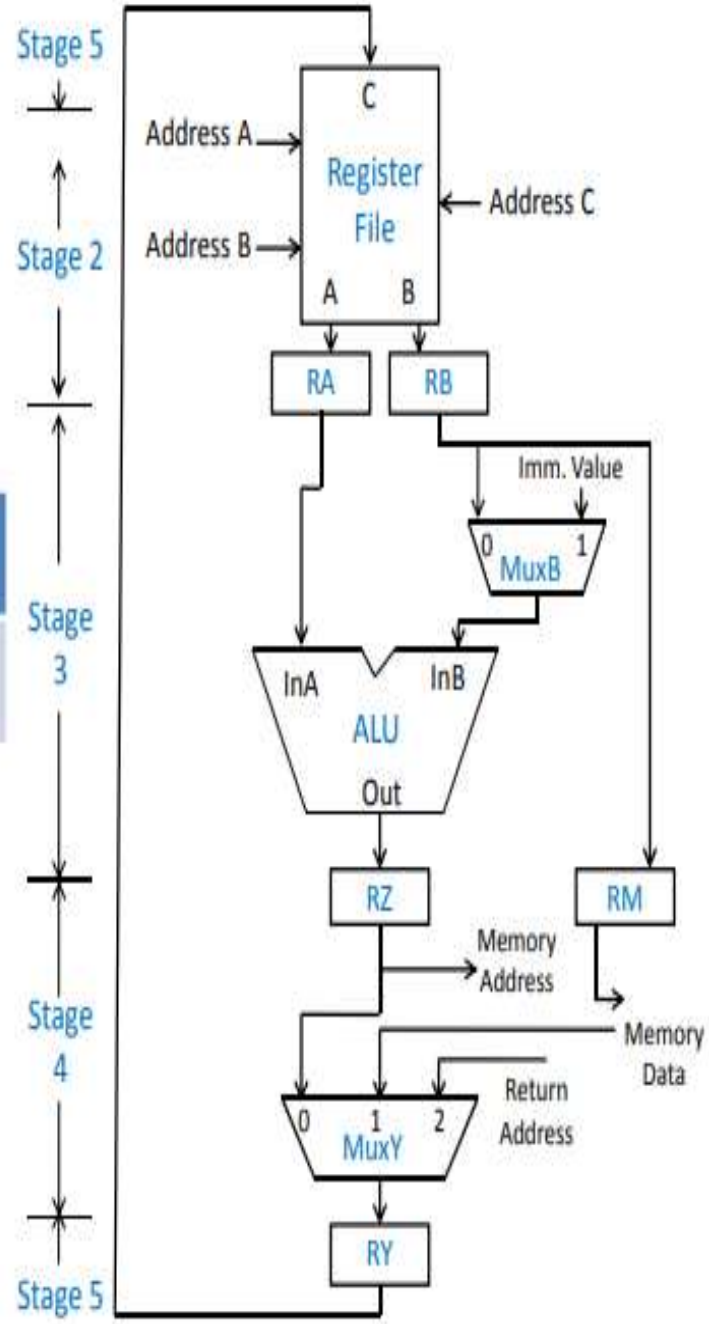



## Subroutine Call Instruction:

saves the subroutine return address (current contents of PC) in a destination register

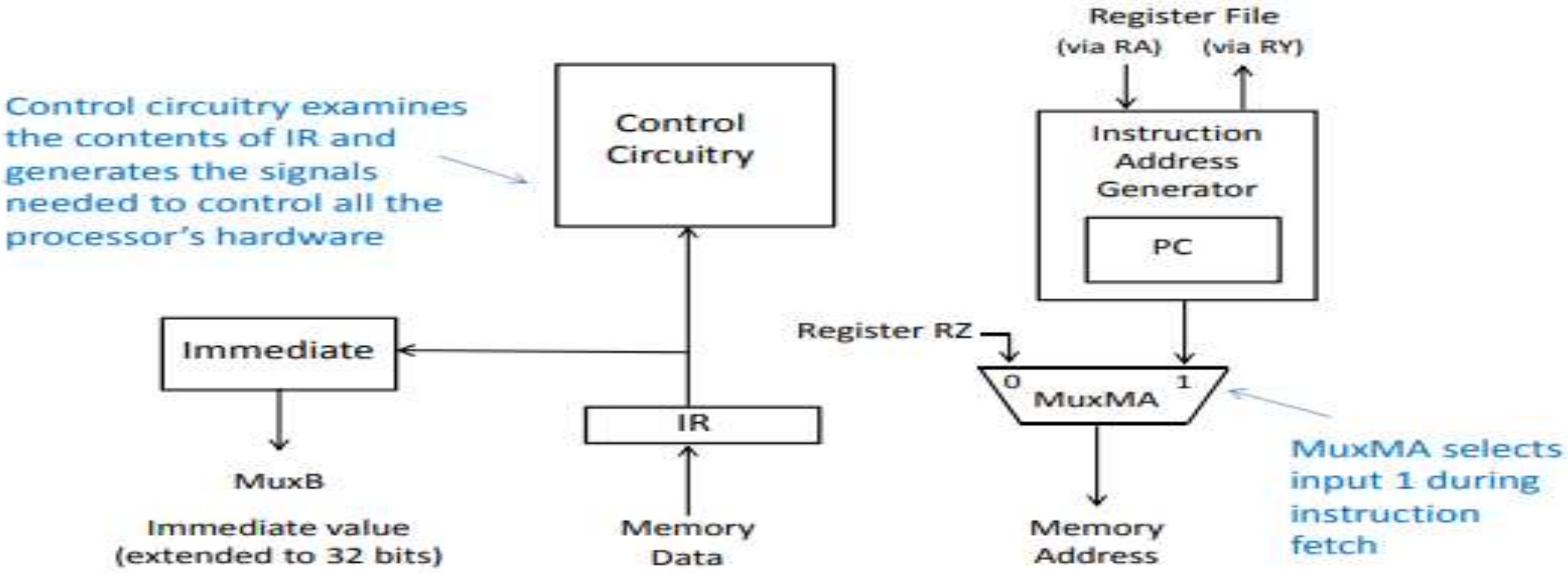
| Instruction Type | MuxB selection | MuxY selection |
|---|---|---|
| Subroutine Call | x | 2 |

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



# Instruction Fetch Section

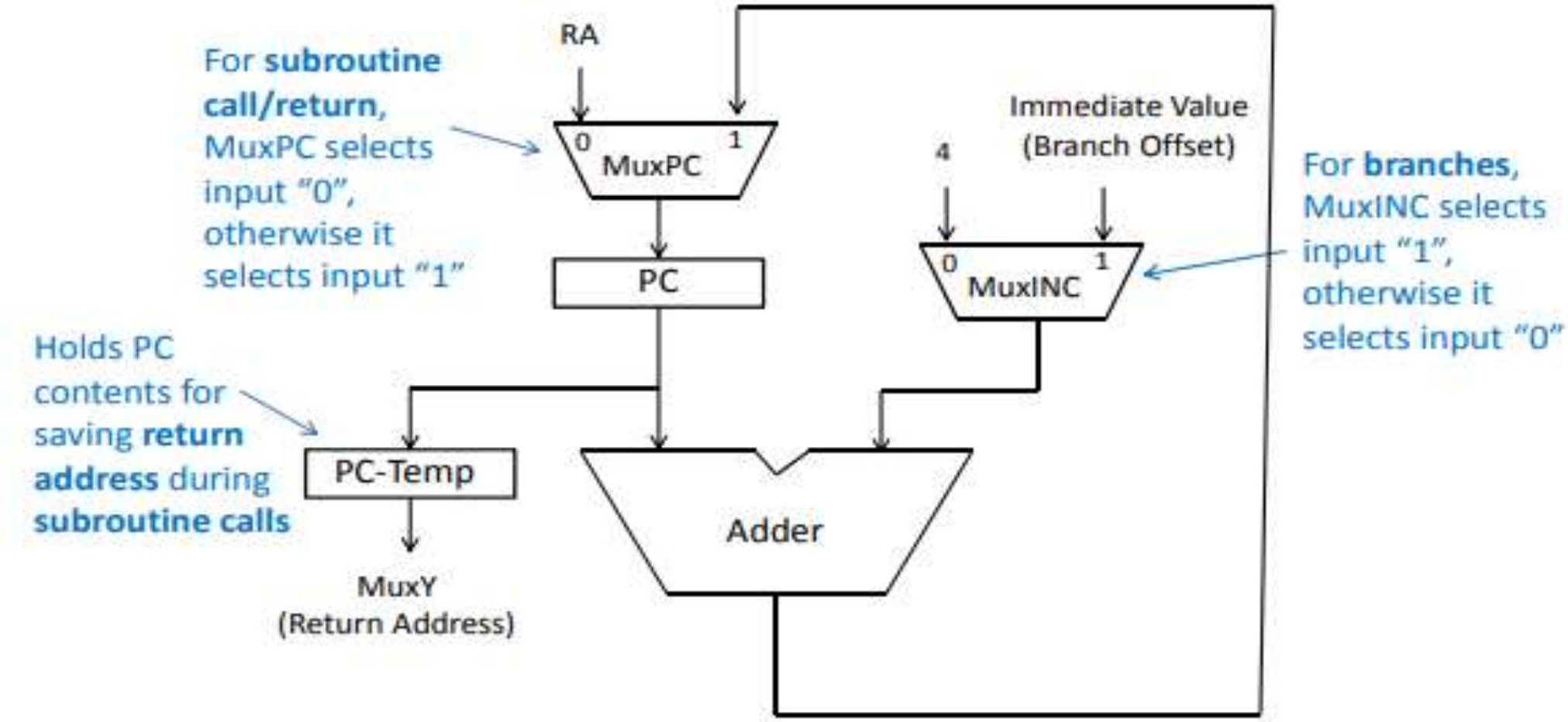
The organization of the instruction fetch section of the processor is




- The addresses used to access the memory come from the PC when fetching instructions and from register RZ in the datapath when accessing instruction operands.
- Multiplexer MuxMA selects one of these two sources to be sent to the processor-memory interface.

- The PC is included in a larger block, the instruction address generator, which updates the contents of the PC after each instruction is fetched.
- The instruction read from the memory is loaded into the IR, where it stays until its execution is completed and the next instruction is fetched.
- The contents of the IR are examined by the control circuitry to generate the signals needed to control all the processor's hardware.
- They are also used by the block labeled Immediate. An immediate value may be included in some instructions.
- A 16-bit immediate value is extended to 32 bits. The extended value is then used either directly as an operand or to compute the effective address of an operand.
- For some instructions, such as those that perform arithmetic operations, the immediate value is sign-extended; for others, such as logic instructions, it is padded with zeros.
- The Immediate block generates the extended value and forwards it to MuxB to be used in an ALU computation.
- It also generates the extended value to be used in computing the target address of branch instructions.

The address generator circuit is shown in

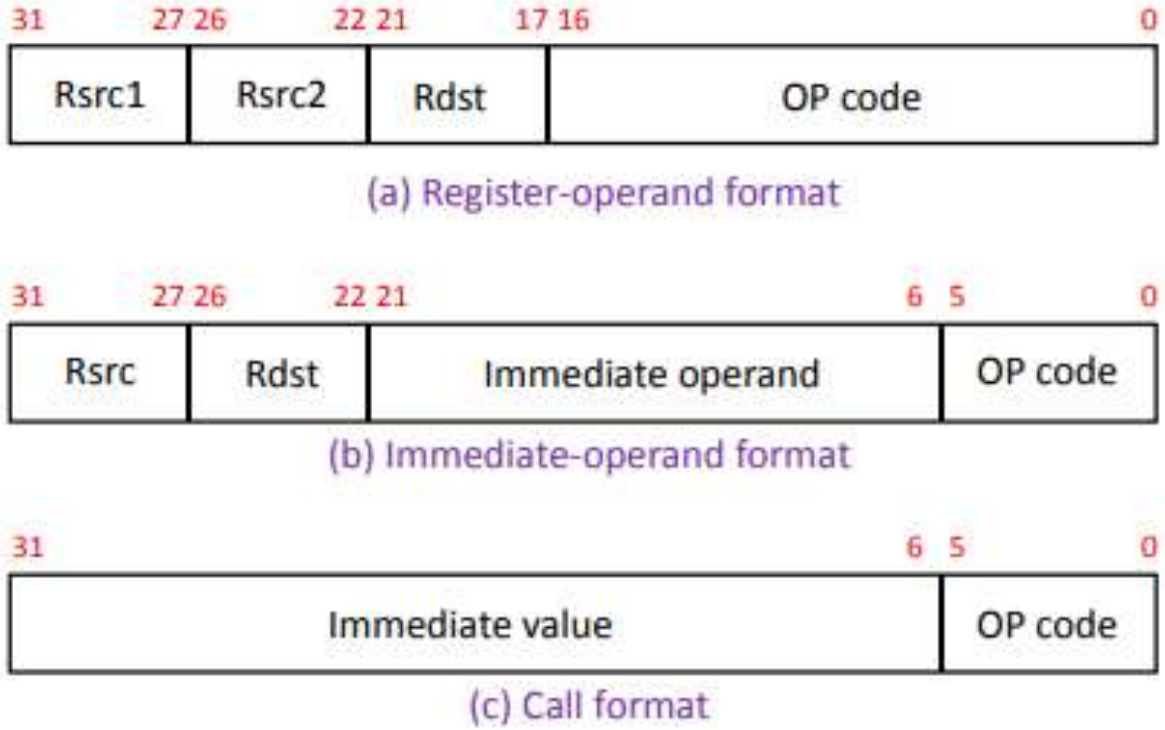



- An adder is used to increment the PC by 4 during straight-line execution. It is also used to compute a new value to be loaded into the PC when executing branch and subroutine call instructions. One adder input is connected to the PC.
- The second input is connected to a multiplexer, MuxINC, which selects either the constant 4 or the branch offset to be added to the PC.

- The branch offset is given in the immediate field of the IR and is sign-extended to 32 bits by the Immediate block.
- The output of the adder is routed to the PC via a second multiplexer, MuxPC, which selects between the adder and the output of register RA.
- The latter connection is needed when executing subroutine linkage instructions.
- Register PC-Temp is needed to hold the contents of the PC temporarily during the process of saving the subroutine or interrupt return address.

# Instruction Fetch and Execution Steps

Using a few standard instruction formats simplifies the instruction decoding process




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

Consider the instruction Add R3, R4, R5

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

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

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

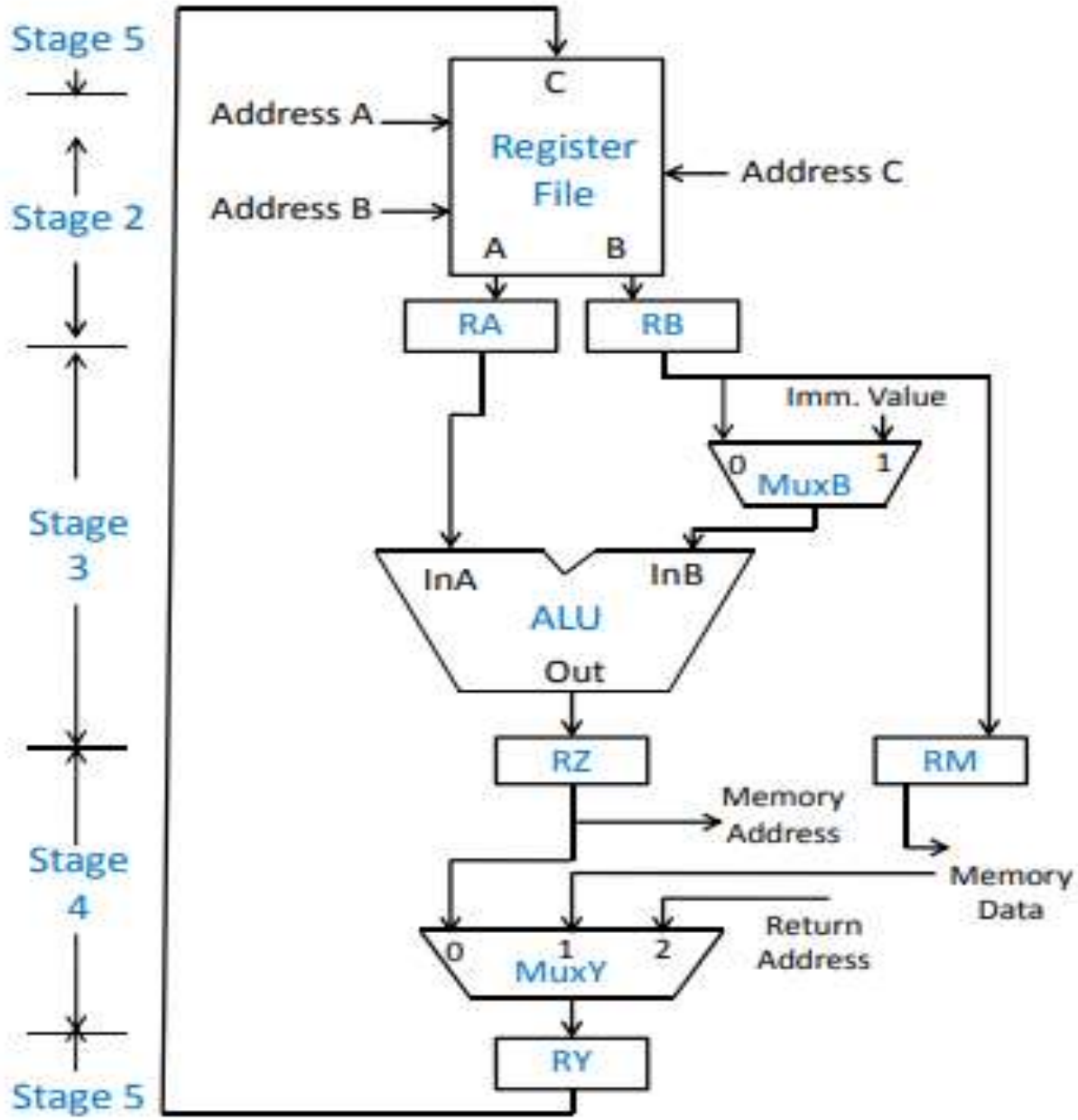
**Stage 1:**
Memory address ← [PC],
Read memory,
IR ← Memory data,
PC ← [PC] + 4

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

**Stage 3:**
RZ ← [RA] + [RB]

**Stage 4:**
RY ← [RZ]

**Stage 5:**
R3 ← [RY]



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

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

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

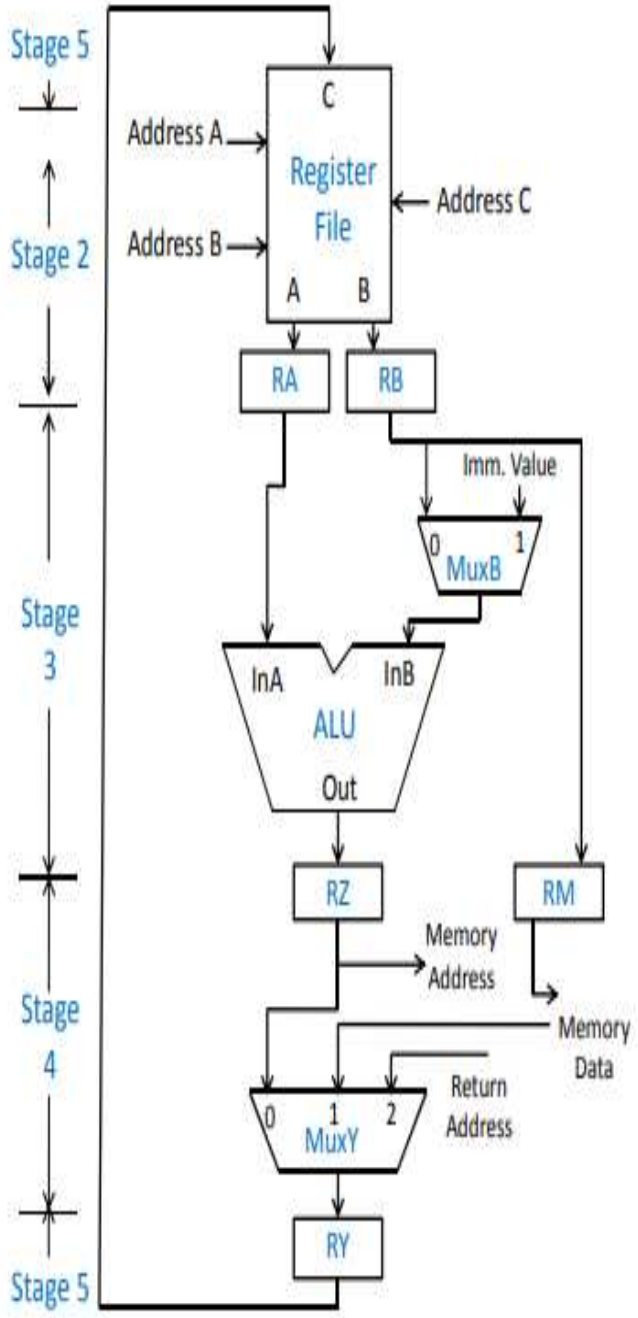
**Stage 1:**
Memory address ← [PC],
Read memory,
IR ← Memory data,
PC ← [PC] + 4

**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]



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

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

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

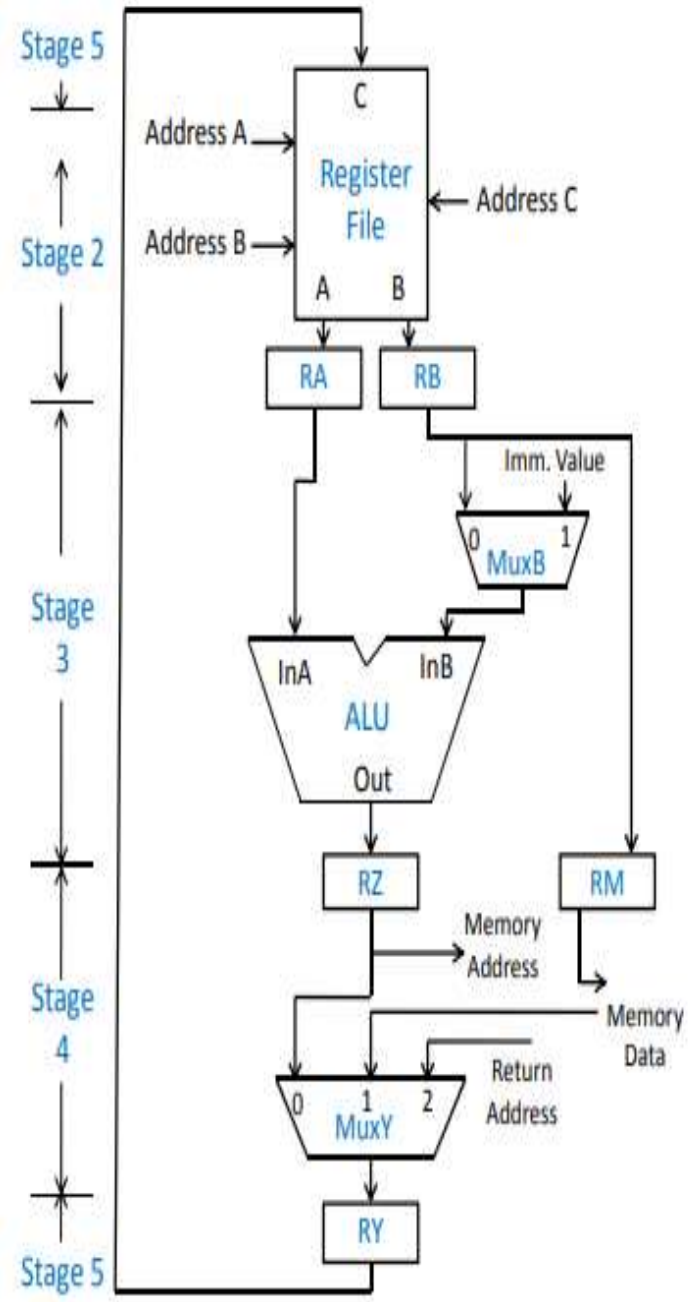
**Stage 1:**
Memory address ← [PC],
Read memory,
IR ← Memory data,
PC ← [PC] + 4

**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



# Branching

- Instructions are fetched from sequential word locations in the memory during straight-line program execution. Whenever an instruction is fetched, the processor increments the PC by 4 to point to the next word.
- This execution pattern continues until a branch or subroutine call instruction loads a new address into the PC.
- Branches and subroutine calls change the control flow of program by loading a new address in the PC
- Branch instruction specifies the target address relative to the PC

– Branch offset given as a 16-bit immediate field in IR

– Branch offset added to the current contents of PC

– Branches are of two types, conditional and unconditional

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

| Step | Action |
|---|---|
| 1 | Memory address ←[PC], Read memory, IR ← Memory data, PC ← [PC] + 4 |
| 2 | Decode instruction |
| 3 | PC ←[PC] + Branch offset (MuxINC selects its “1" input) |

4 No action

5 No action

PC is first incremented by 4 during stage 1 and then by the branch offset during stage 3 – [PC]+4 points to the location following the branch instruction.

Therefore Branch offset is the distance between the branch target and the memory location following the branch instruction

## Conditional Branch Execution Steps

The branch instruction specifies a compare-and-test operation that determines the branch condition. For example, the instruction Branch_if_[R5]=[R6] LOOP results in a branch if the contents of registers R5 and R6 are identical.

When this instruction is executed, the register contents are compared, and if they are equal, a branch is made to location LOOP

Example: Branch_if_[R5]=[R6] Offset

| Step | Action |
|---|---|
| 1 | Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4 |
| 2 | Decode instruction, RA ← [R5], RB ←[R6] |
| 3 | Compare [RA] to [RB], If [RA] = [RB], then PC ← [PC] + Branch offset |

| | |
|---|---|
| 4 | No action |
| 5 | No action |

## Subroutine Call Instructions

- Subroutine calls and returns are implemented in a similar manner to branch instructions.
- The address of the subroutine may either be computed using an immediate value given in the instruction or it may be given in full in one of the general-purpose registers.

Example: Call_Register R9 (Call a subroutine whose starting address is in register R9)

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

Return address of the subroutine is saved in a general-purpose register called LINK in the register file

## Return-from-Subroutine Execution Steps

• Sub-routine return instruction transfers the value saved in register LINK back to the PC

• This Instruction is encoded such that the address of LINK register appears in IR[31-27]

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

Stage 2: Decode instruction, RA ←[Register LINK]

Stage 3: PC ← [RA] (MuxPC selects its “0” input)

Stage 4: No Action

Stage 5: No Action

## Waiting for Memory

The role of the processor-memory interface circuit is to control data transfers between the processor and the memory.

We pointed out earlier that modern processors use fast, on-chip cache memories.

Most of the time, the instruction or data referenced in memory Read and Write operations are found in the cache, in which case the operation is completed in one clock cycle.

- When the requested information is not in the cache and has to be fetched from the main memory, several clock cycles may be needed.
- The interface circuit must inform the processor’s control circuitry about such situations, to delay subsequent execution steps until the memory operation is completed.
- Assume that the processor-memory interface circuit generates a signal called Memory Function Completed (MFC).
- It asserts this signal when a requested memory Read or Write operation has been completed.
- The processor’s control circuitry checks this signal during any processing step in which it issues a memory Read or Write request, to determine when it can proceed to the next step.
- When the requested data are found in the cache, the interface circuit asserts the MFC signal before the end of the same clock cycle in which the memory request is issued. Hence, instruction execution continues uninterrupted.
- If access to the main memory is required, the interface circuit delays asserting MFC until the operation is completed.
- In this case, the processor’s control circuitry must extend the duration of the execution step for as many clock cycles as needed, until MFC is asserted.

- We will use the command Wait for MFC to indicate that a given execution step must be extended, if necessary, until a memory operation is completed.
- When MFC is received, the actions specified in the step are completed, and the processor proceeds to the next step in the execution sequence.
- Step 1 of the execution sequence of any instruction involves fetching the instruction from the memory.
- Therefore, it must include a Wait for MFC command, as follows:

Memory address ← [PC],

Read memory,

Wait for MFC,

IR ← Memory data,

PC ← [PC] + 4

- Most of the time, the requested information is found in the cache, so the MFC signal is generated quickly, and the step is completed in one clock cycle.
- When an access involves the main memory, the MFC response is delayed, and the step is extended to several clock cycles

# Control Signals

The operation of the processor's hardware components is governed by control signals. These signals determine which multiplexer input is selected, what operation is performed by the ALU, and so on

In each clock cycle, the results of the actions that take place in one stage are stored in inter-stage registers, to be available for use by the next stage in the next clock cycle. Since data are transferred from one stage to the next in every clock cycle, inter-stage registers are always enabled. This is the case for registers RA, RB, RZ, RY, RM, and PC-Temp.

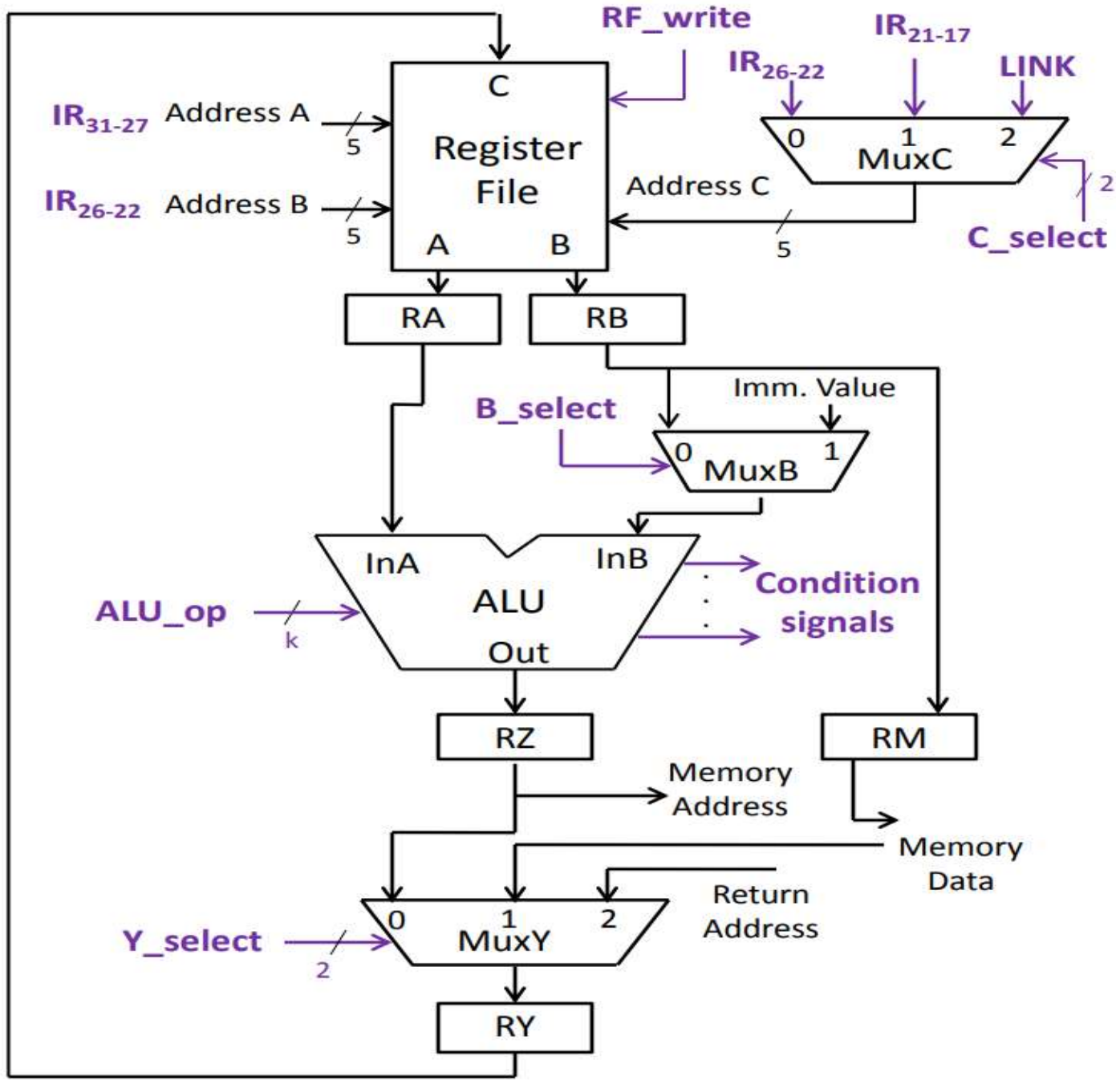
The contents of the other registers, namely, the PC, the IR, and the register file, must not be changed in every clock cycle. New data are loaded into these registers only when called for in a particular processing step. They must be enabled only at those times.

Control circuitry examines the instruction in IR and generates the control signals needed to execute the instruction . Examples of decisions made by control signals:

– Which registers (if any) are enabled for writing? – Which input is selected by a multiplexer?

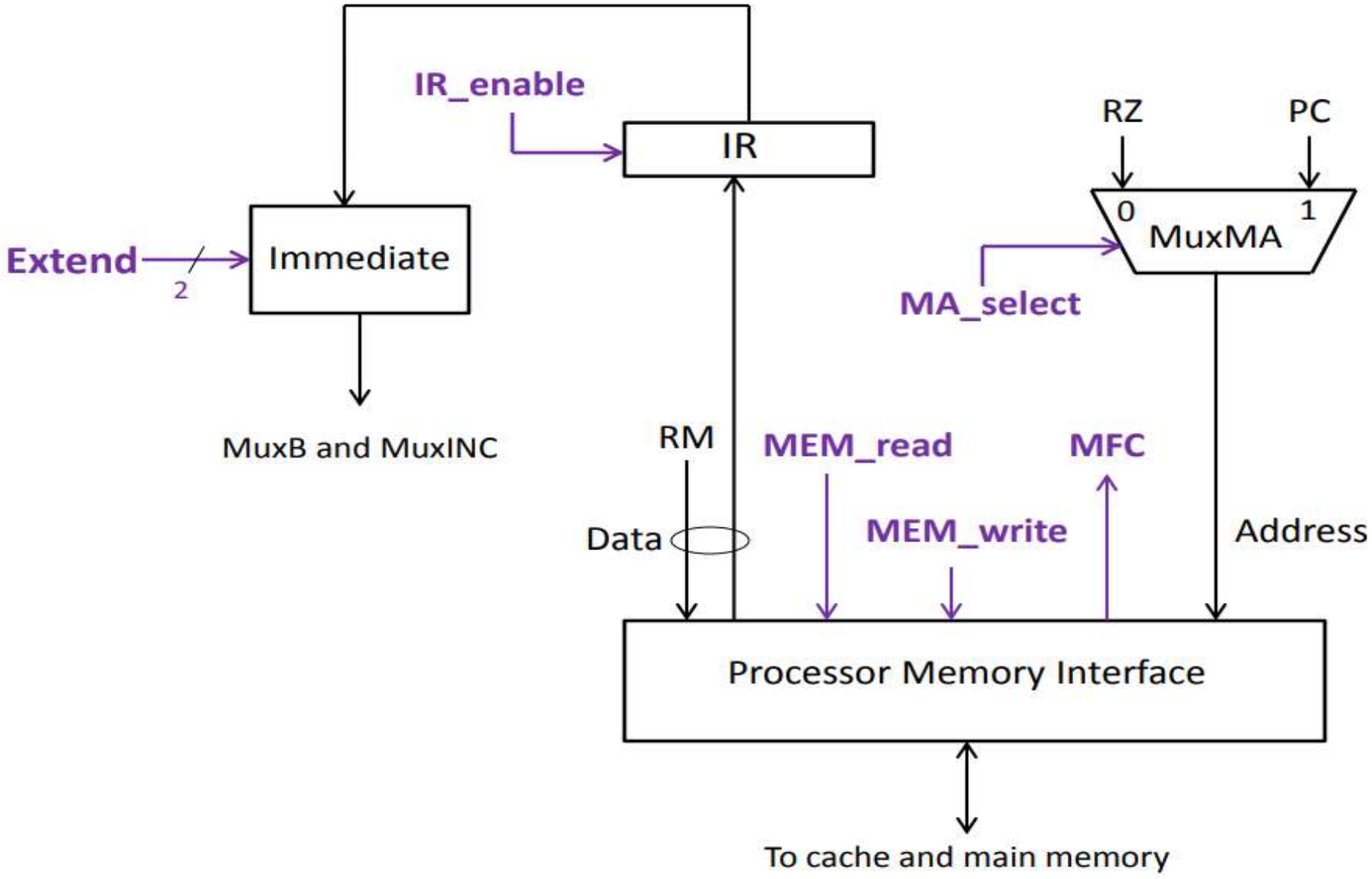
– What operation is performed by the ALU?

• Some control signals depend only on instruction type, while others depend on both the instruction type and current processing step


RF_write
IR21-17
IR26-22
LINK
C
IR31-27
Address A
5
Register
File
IR26-22
Address B
5
A
B
0
1
2
MuxC
Address C
5
2
C_select
RA
RB
Imm. Value
B_select
0
MuxB
1
InA
InB
ALU
Out
ALU_op
k
Condition
signals
RZ
RM
Memory
Address
Memory
Data
Return
Address
0
1
2
MuxY
Y_select
2
RY

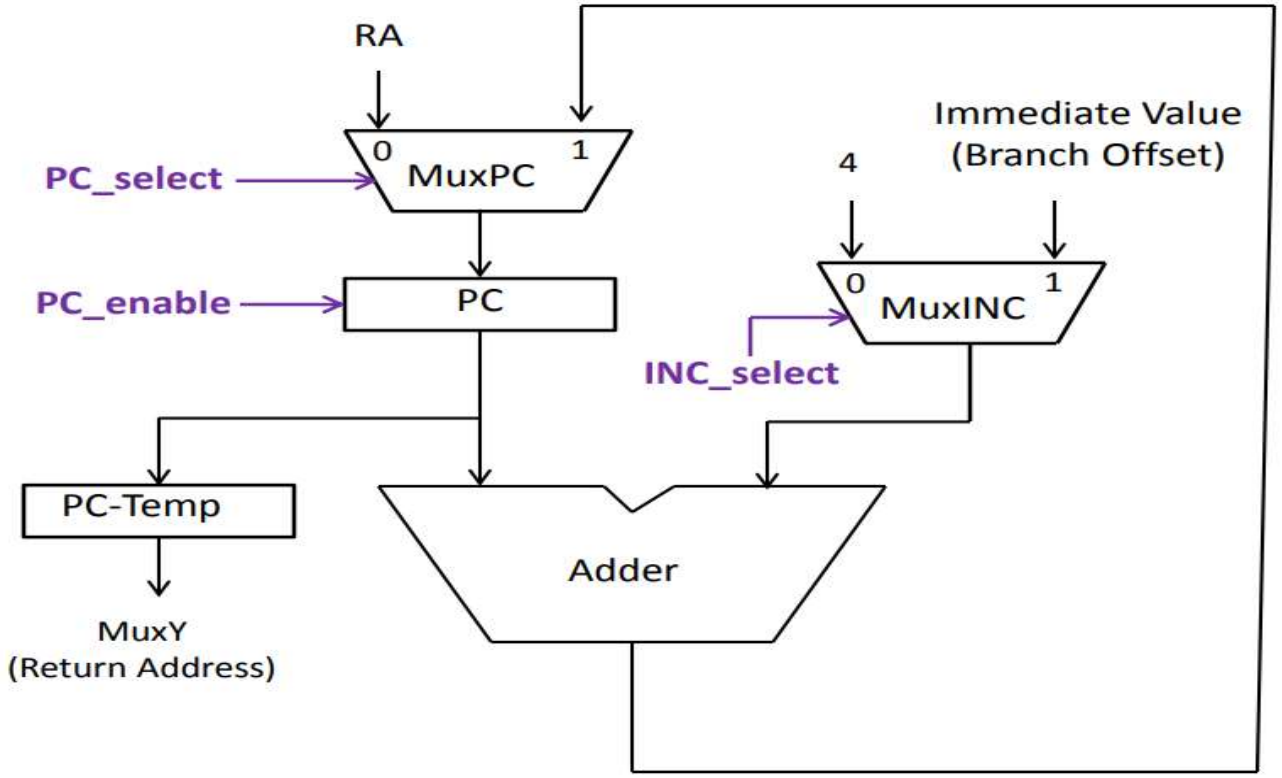
- The register file has three 5-bit address inputs, allowing access to 32 general-purpose registers.
- Two of these inputs, Address A and Address B, determine which registers are to be read. They are connected to fields IR31−27 and IR26−22 in the instruction register.
- The third address input, Address C, selects the destination register, into which the input data at port C are to be written. Multiplexer MuxC selects the source of that address.
- We have assumed that three-register instructions use bits IR21−17 and other instructions use IR26−22 to specify the destination register.
- The third input of the multiplexer is the address of the link register used in subroutine linkage instructions. New data are loaded into the selected register only when the control signal RF_write is asserted
- Multiplexers are controlled by signals that select which input data appear at the multiplexer's output.
- For example, when B_select is equal to 0, MuxB selects the contents of register RB to be available at input InB of the ALU.
- Note that two bits are needed to control MuxC and MuxY, because each multiplexer selects one of three inputs.
- The operation performed by the ALU is determined by a k-bit control code, ALU_op, which can specify up to 2k distinct operations, such as Add, Subtract, AND, OR, and XOR

The interface between the processor and the memory and the control signals associated with the instruction register are presented in Figure



- Two signals, MEM_read and MEM_write are used to initiate a memory Read or a memory Write operation. When the requested operation has been completed, the interface asserts the MFC signal.
- The instruction register has a control signal, IR_enable, which enables a new instruction to be loaded into the register.
- During a fetch step, it must be activated only after the MFC signal is asserted

The signals that control the operation of the instruction address generator are shown in Figure



- The INC_select signal selects the value to be added to the PC, either the constant 4 or the branch offset specified in the instruction.
- The PC_select signal selects either the updated address or the contents of register RA to be loaded into the PC when the PC_enable control signal is activated.

## Hardwired Control

We examine how the processor generates the control signals that cause these actions to take place in the correct sequence and at the right time.

There are two basic approaches: hardwired control and microprogrammed control.

### Hardwired control

- An instruction is executed in a sequence of steps, where each step requires one clock cycle. Hence, a step counter may be used to keep track of the progress of execution.
- Several actions are performed in each step, depending on the instruction being executed. In some cases, such as for branch instructions, the actions taken depend on tests applied to the result of a computation or a comparison operation.
- External signals, such as interrupt requests, may also influence the actions to be performed.

Thus, the setting of the control signals depends on:

- Contents of the step counter
- Contents of the instruction register
- The result of a computation or a comparison operation
- External input signals, such as interrupt requests

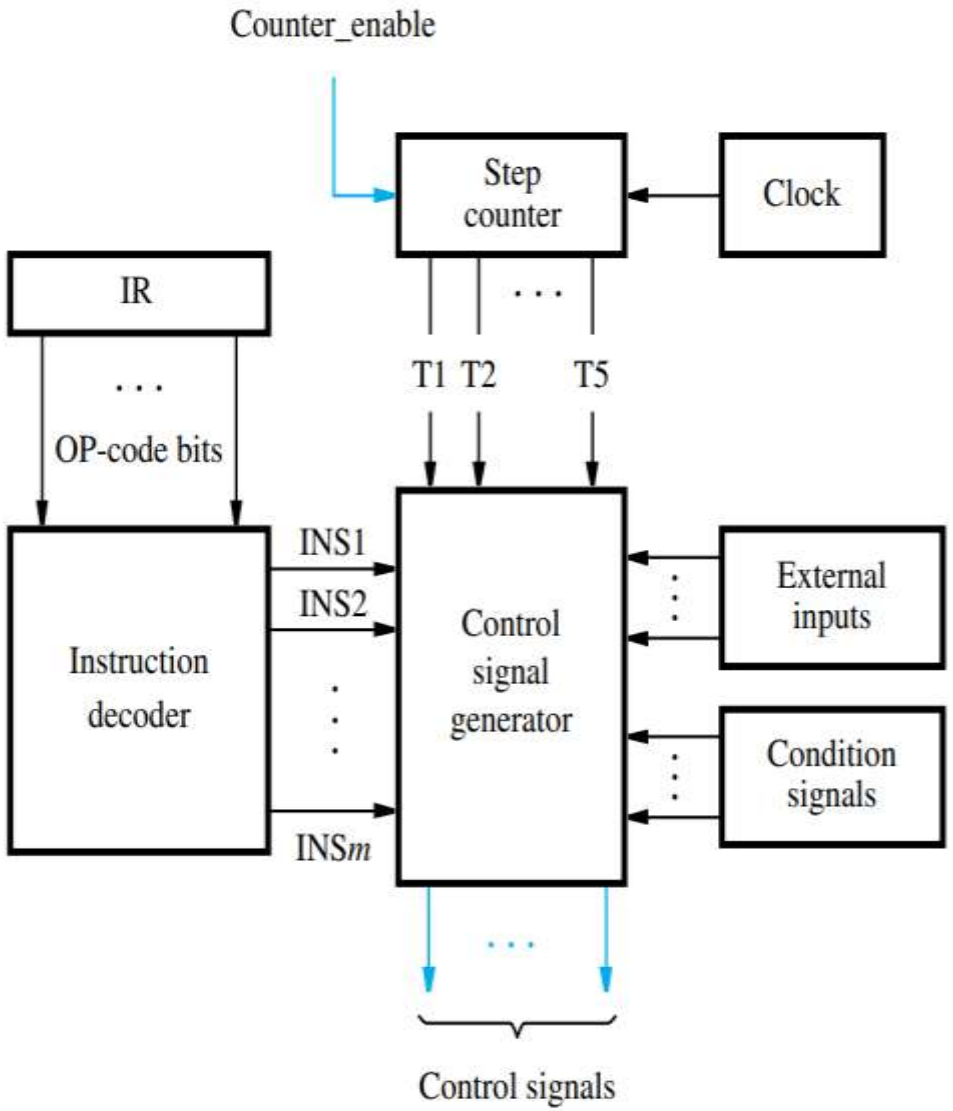



**Figure 5.21** Generation of the control signals.

The instruction decoder interprets the OP-code and addressing mode information in the IR and sets to 1 the corresponding INSi output.

During each clock cycle, one of the outputs T1 to T5 of the step counter is set to 1 to indicate which of the five steps involved in fetching and executing instructions is being carried out.

Since all instructions are completed in five steps, a modulo-5 counter may be used.

The control signal generator is a combinational circuit that produces the necessary control signals based on all its inputs.

The required settings of the control signals can be determined from the action sequences that implement each of the instructions represented by the signals INS1 to INSm.

Example: Consider the fetch stage (stage 1) of the five-stage hardware

Step counter asserts the signal T1

• Control circuitry:

– sets MA_select signal to 1 to select PC contents as memory address

– activates Mem_Read to initiate a memory read operation

– activates IR_enable to load the data returned from memory into IR, when MFC is asserted

– sets Inc_Select to 0, PC_select to 1 and asserts PC_enable to increment PC by 4 at the end of step T1

# Datapath Control Signals

• Setting of control signals can be determined by examining the actions taken in each execution step of every instruction

•Example 1: RF_write signal is set to 1 in step T5 during an instruction that writes data into the register file:

$$RF\_write = T5.(ALU + Load + Call)$$

where ALU, Load and Call stand for arithmetic/logic instructions, load instructions and subroutine call instructions respectively

– RF_write is a function of both the timing and instruction signals

• Example 2: The multiplexer B_select is a function of only the instruction and does not need to change from one timing step to the other

$$B\_select = Immediate$$

where Immediate stands for all instructions that use an immediate operand

## Dealing with Memory Delay

- The timing signals T1 to T5 are asserted in sequence as the step counter is advanced. Most of the time, the step counter is incremented at the end of every clock cycle.
- However, a step in which a MEM_read or a MEM_write command is issued does not end until the MFC signal is asserted, indicating that the requested memory operation has been completed
- Assume that the counter is incremented when enabled by a control signal called Counter_enable. Let the need to wait for a memory operation to be completed be indicated by a control signal called WMFC, which is activated during any execution step in which the Wait for MFC command is issued.
- Counter_enable should be set to 1 in any step in which WMFC is not asserted. Otherwise, it should be set to 1 when MFC is asserted.

This means that $\text{Counter\_enable} = \text{NOT(WMFC)} + \text{MFC}$

- We must ensure that the PC is incremented only once when an execution step is extended for more than one clock cycle.
- Hence, when fetching an instruction, the PC should be enabled only when MFC is received. It is also enabled in step 3 of instructions that cause branching.

Let BR denote all instructions in this group. Then, PC_enable may be realized as

$$\text{PC\_enable} = T1 \cdot \text{MFC} + T3 \cdot \text{BR}$$