# Instruction Execution Cycle

# Fetch & Execute Cycle

1. **Instruction Fetch Phase**
2. **Instruction Execute Phase**

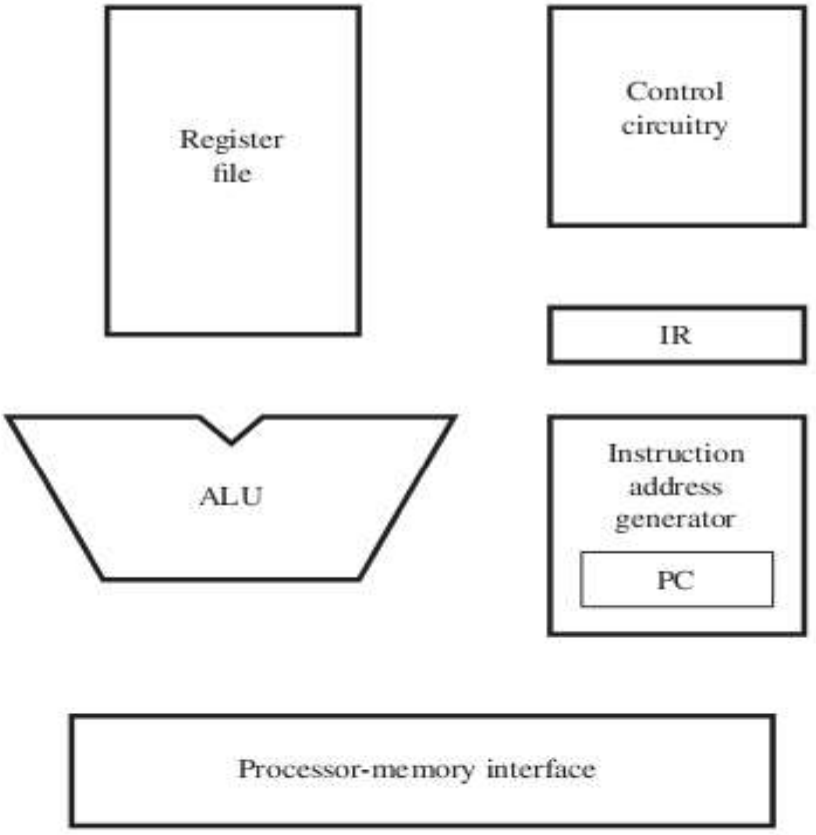


**Figure 5.1** Main hardware components of a processor.

Read page 152 and 153

# Instruction Fetching

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.

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. IR←[[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←[PC] + 4

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

Fetching an instruction and loading it into the IR is usually referred to as the **instruction fetch phase.**
Performing the operation specified in the instruction constitutes the **instruction execution phase.**

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 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.

# 5 Step suitable for all RISC Instructions

| 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.

# Load Instructions (1)

Load R5, X(R7)

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.

# Load Instructions (2)

In the discussion that follows, we will assume that the processor has five hardware stages, which is a commonly used arrangement in RISC-style processors. Execution of each instruction is divided into five steps, such that each step is carried out by one hardware stage. 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 & Logic Instructions (1)

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. No action.
5. Load the result into the destination register, R3.

# Arithmetic & Logic Instructions (2)

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 :

1. Fetch the instruction and increment the program counter.
2. Decode the instruction and read register R4.
3. Compute the sum [R4] + 1000.
4. No action.
5. Load the result into the destination register, R3.

# Store Instructions (1)

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.

### 5.3.2 ALU

The arithmetic and logic unit is used to manipulate data. It performs arithmetic operations such as addition and subtraction, and logic operations such as AND, OR, and XOR. Conceptually, the register file and the ALU may be connected as shown in Figure 5.6. When an instruction that performs an arithmetic or logic operation is being executed, the contents of the two registers specified in the instruction are read from the register file and become available at outputs A and B. Output A is connected directly to the first input of the ALU, InA, and output B is connected to a multiplexer, MuxB. The multiplexer selects either output B of the register file or the immediate value in the IR to be connected to the second ALU input, InB. The output of the ALU is connected to the data input, C, of the register file so that the results of a computation can be loaded into the destination register.

# 5 Stage Organization

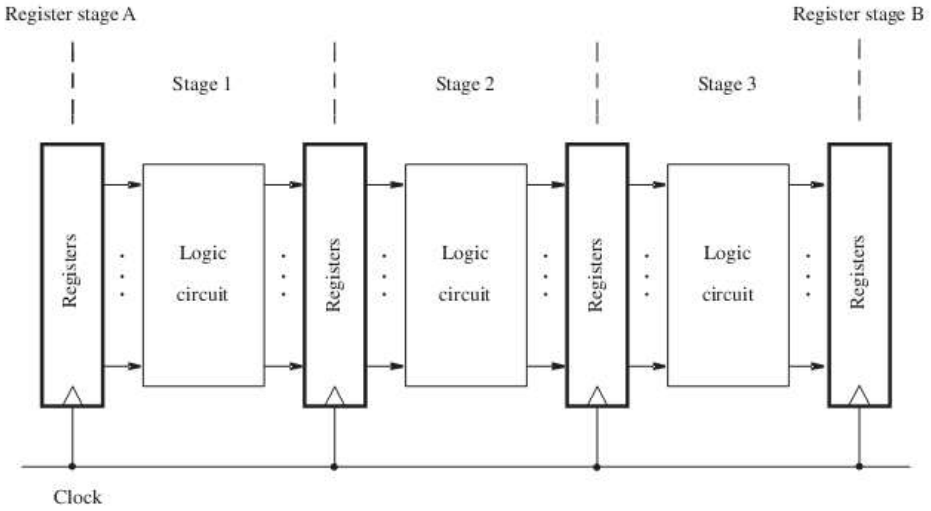


**Figure 5.3** A hardware structure with multiple stages.

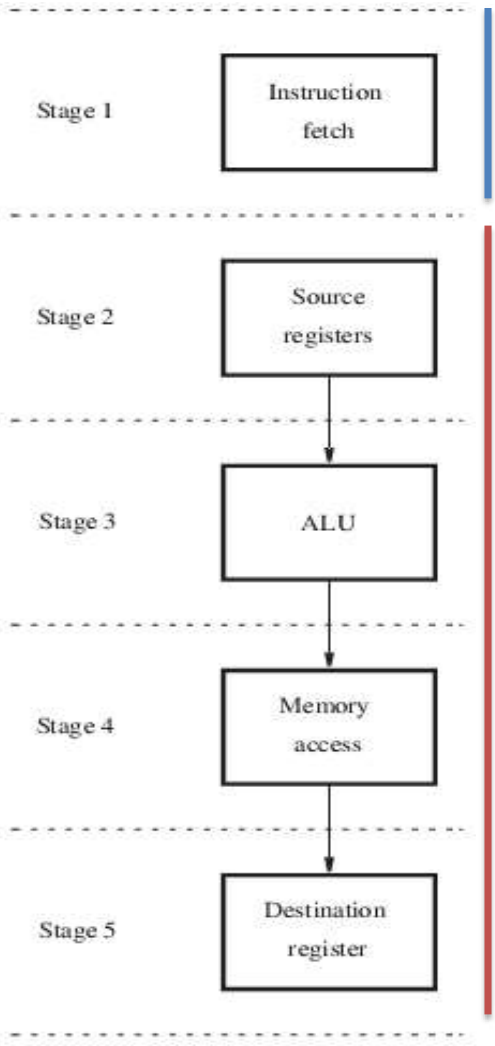


**Figure 5.7** A five-stage organization.

# Instruction Fetch & Execution Steps

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] |

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

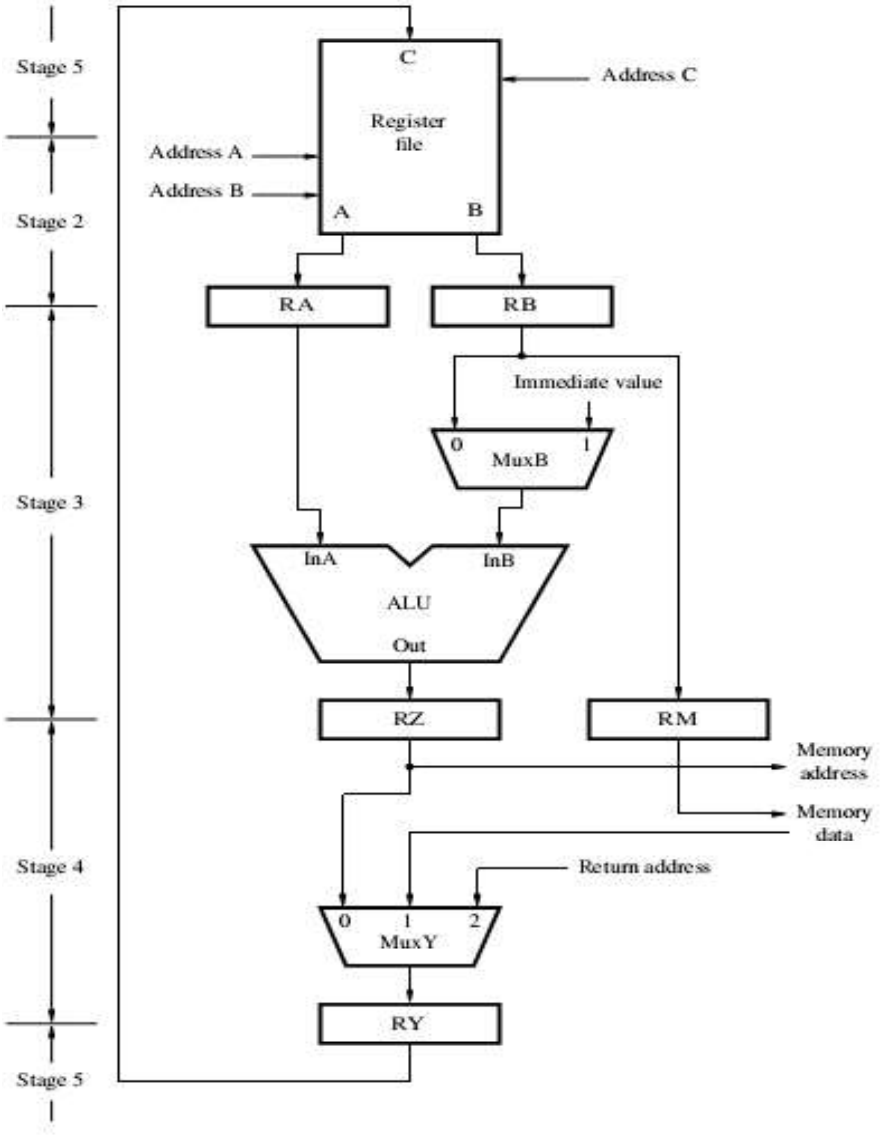


**Figure 5.8** Datapath in a processor.

# Instruction Fetch & Execution Steps

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] |

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

# Instruction Fetch & Execution Steps

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 |

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

# Instruction Fetch & Execution Steps

Branch DONE

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

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

# Instruction Fetch & Execution Steps

Branch_if_[R5]=[R6] LOOP

| 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 |

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