Computer Architecture Project

Course code: *CC311*

Basic Processing Unit (RISC)

Engineers:

Nada Shalan	20104602
Ismail Mohamed	20107070
Haneen Talaat	20104712
Alaa Imam	20106028
Anton Ashraf	20105183
Ahmed Kamal	20106009
Mahmoud Hany	20104375
Ahmed Mohamed Eldalil	20104042

Introduction:

The execution of instructions in a RISC-style processor can be accomplished by following a sequential process consisting of five steps. Consequently, the hardware of the processor can be structured into five distinct stages, with each stage responsible for executing the actions required in a particular step. In this discussion, we will analyze the components depicted in Figure 5.1 and explore how they can be efficiently organized within this multi-stage framework.

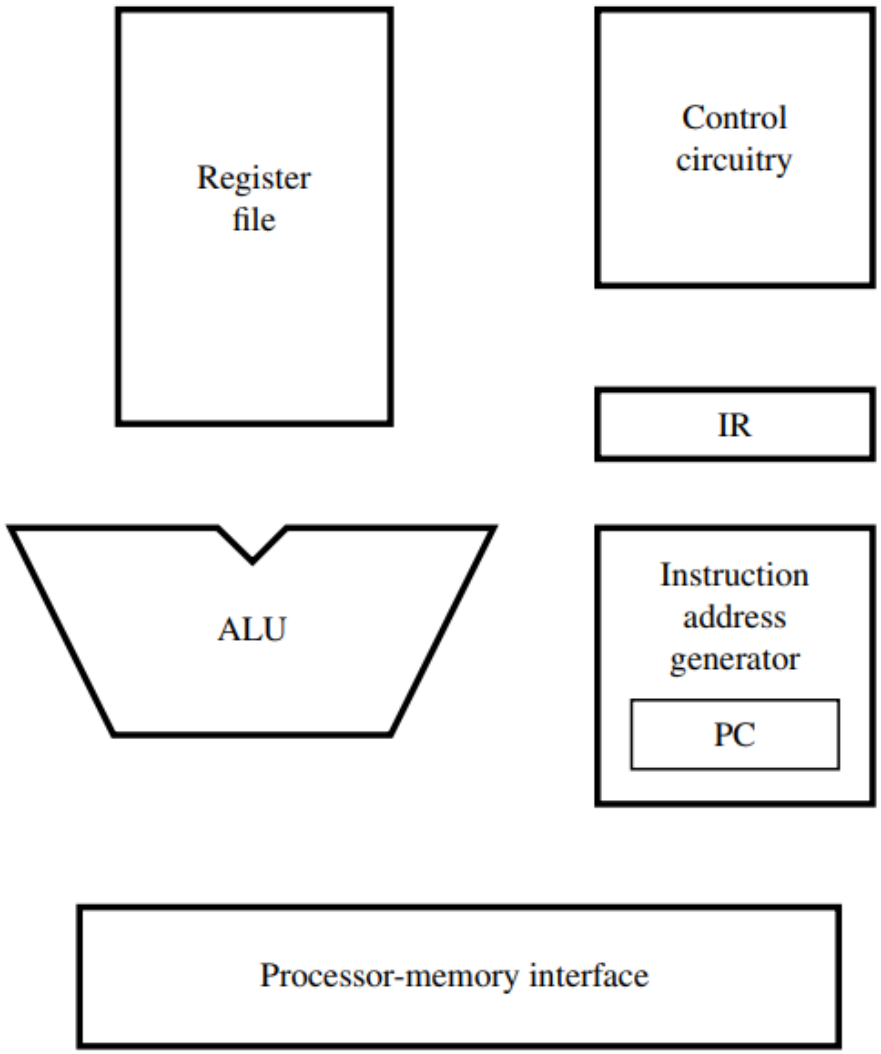
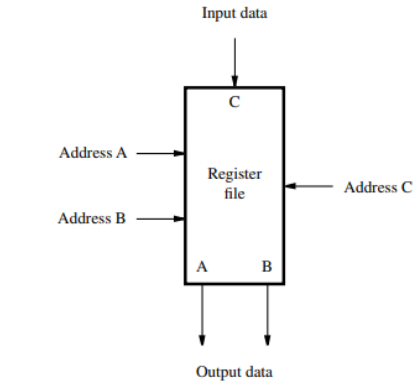


Figure 5.1 Main hardware components of a processor.

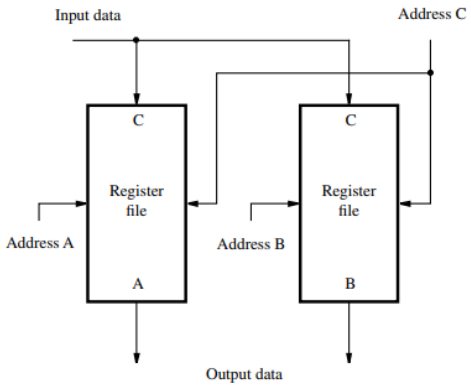
Register File:

In computer architecture, the implementation of general-purpose registers often involves utilizing a register file, which serves as a compact and high-speed memory block. The register file consists of an array of storage elements, accompanied by access circuitry that facilitates the reading and writing of data to and from any register within the file. The design of the access circuitry is specifically engineered to enable concurrent reading of two registers, thus providing separate outputs, namely A and B, for their respective contents.

The register file incorporates two address inputs that are responsible for selecting the two registers to be read. These inputs are connected to the fields within the Instruction Register (IR) that specify the source registers, allowing the desired registers to be accessed and their contents made available. Additionally, the register file encompasses a data input, referred to as C, along with a corresponding address input that determines the register into which data will be written. This address input is connected to the IR field that designates the destination register for the particular instruction.



(a) Single memory block



(b) Two memory blocks

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

By employing a register file, the processor can efficiently store and retrieve data from multiple registers, facilitating rapid access and manipulation of information during the execution of instructions. This organization plays a crucial role in enhancing the overall performance and efficiency of the processor's data processing capabilities.

Arithmetic and Logic Unit (ALU):

The arithmetic and logic unit (ALU) plays a vital role in data manipulation within a processor. It carries out various arithmetic operations, such as addition and subtraction, as well as logic operations, including AND, OR, and XOR. The connection between the register file and the ALU is illustrated in Figure 5.6, representing their conceptual integration.

During the execution of an instruction involving arithmetic or logic operations, the contents of the two registers specified in the instruction are retrieved from the register file. These values become accessible at outputs A and B. Output A is directly connected to the first input of the ALU, labeled InA. Output B, on the other hand, is connected to a multiplexer known as MuxB. The purpose of this multiplexer is to selectively choose between output B of the register file or the immediate value stored in the Instruction Register (IR). The chosen value is then connected to the second input of the ALU, referred to as InB.

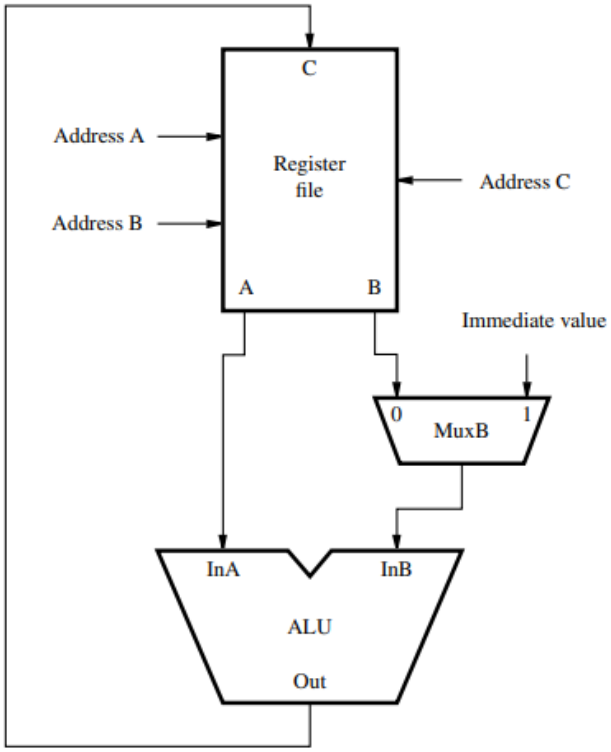


Figure 5.6 Conceptual view of the hardware needed for computation.

Subsequently, the output of the ALU, representing the result of the computation, is connected to the data input C of the register file. This connection allows the computed results to be loaded into the destination register specified in the instruction, enabling the storage of the computation outcome.

By establishing this link between the register file and the ALU, the processor can effectively perform arithmetic and logic operations on data, facilitating data manipulation and computation within the system. This integration significantly contributes to the processor's overall functionality and its ability to execute complex instructions efficiently.

Data Path:

Instruction processing in a processor involves two phases: fetching and execution. To organize the hardware effectively, it is divided into two sections: one for fetching instructions and the other for executing them.

In the fetching section, instructions are fetched, decoded, and control signals are generated for the execution section. The execution section reads data, performs computations, and stores results.

To structure the hardware, a multi-stage approach is used, similar to Figure 5.7. Each stage completes its actions in one clock cycle.

The first stage fetches an instruction and stores it in the Instruction Register (IR). In the second stage, the fetched instruction is decoded, and the source registers are read. The IR holds the instruction until its execution is completed.

Inter-stage registers are inserted between stages to ensure smooth data flow. These registers hold the results from one stage to be used as inputs in the next stage during the next clock cycle.

This multi-stage structure with inter-stage registers allows the processor to efficiently process instructions, maintain proper sequencing of actions, and ensure accurate data handling throughout program execution.

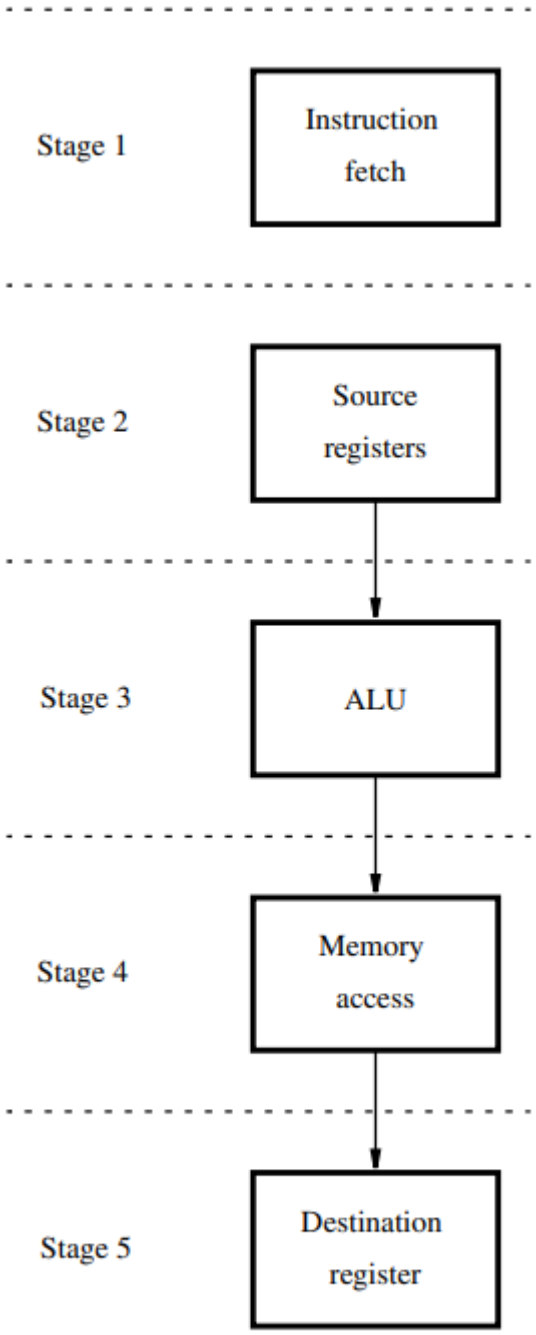


Figure 5.7 A five-stage organization.

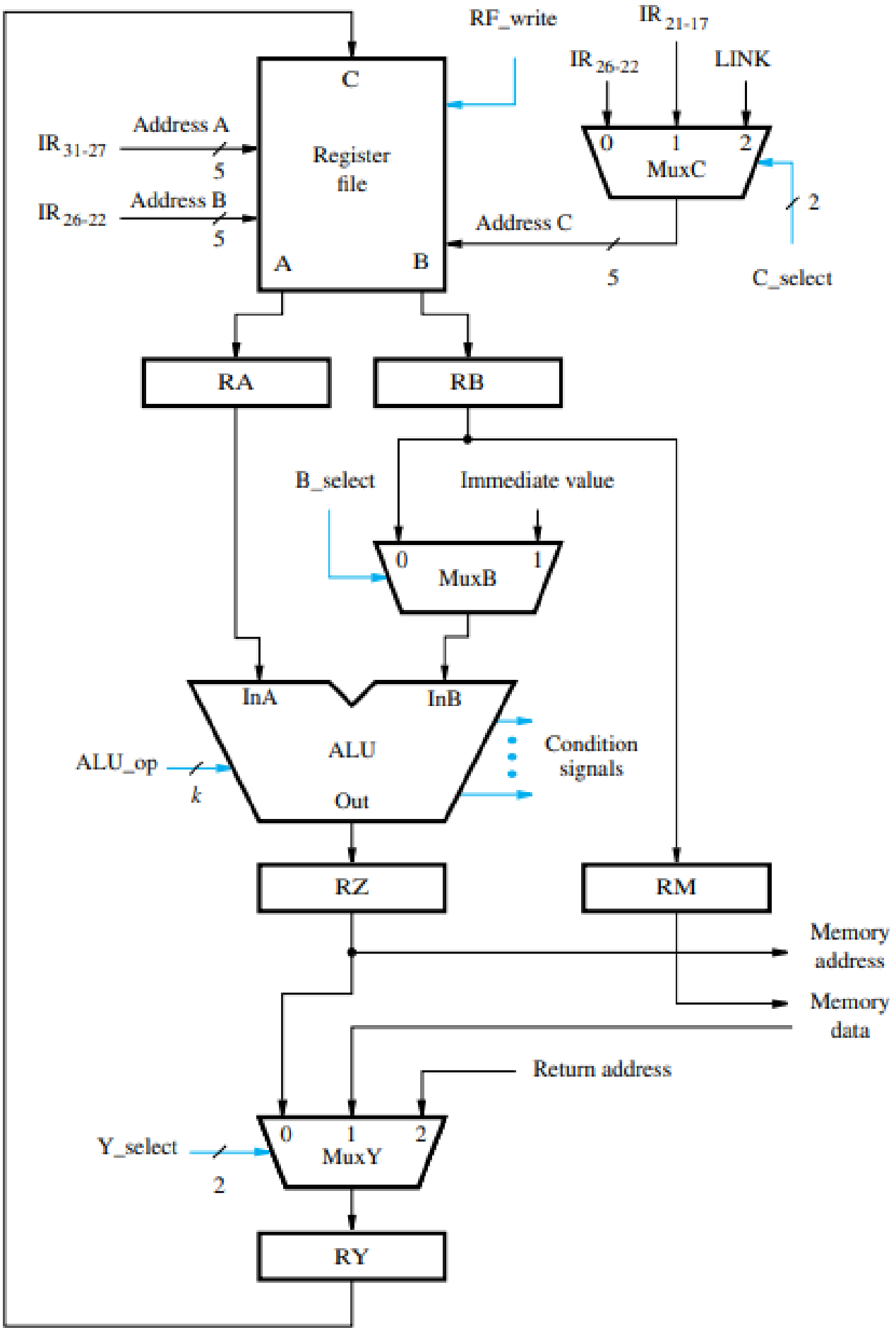


Figure 5.18 Control signals for the datapath.

Instruction Fetch Section:

Within the processor, there is an adder that performs addition operations. One input of the adder is connected to the Program Counter (PC). The second input is linked to a multiplexer called MuxINC. This multiplexer allows the selection of either a constant value of 4 or the branch offset, which is provided in the immediate field of the Instruction Register (IR). The branch offset is sign-extended to 32 bits by the Immediate block in Figure 5.9.

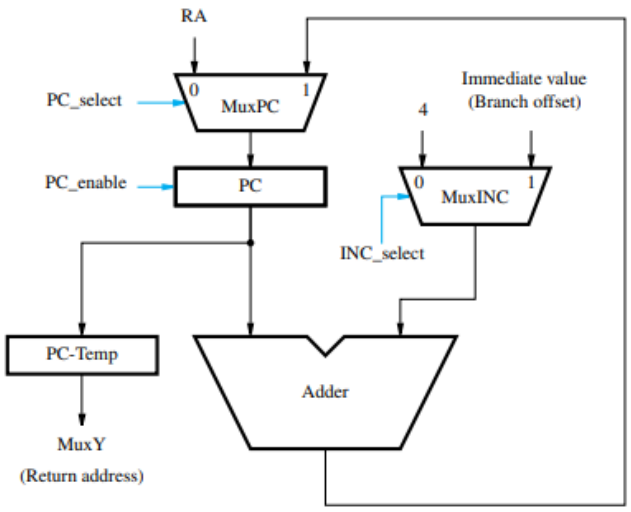


Figure 5.20 Control signals for the instruction address generator.

The output of the adder is then directed to the PC through another multiplexer called MuxPC. This multiplexer allows for the selection between the output of the adder and the contents of register RA. The connection to register RA becomes necessary during the execution of subroutine linkage instructions.

Additionally, a temporary register called PC-Temp is utilized to temporarily store the contents of the PC. This is particularly useful during the process of saving subroutine or interrupt return addresses.

Control Unit:

The generation of control signals in the circuitry can be organized according to Figure 5.21. The instruction decoder analyzes the OP-code and addressing mode information stored in the Instruction Register (IR). Based on this information, the corresponding outputs (INS1 to INSm) are set to 1.

To indicate the current step in the instruction fetching and execution process, outputs T1 to T5 of the step counter are utilized. Each clock cycle sets one of these outputs to 1, reflecting the specific step being carried out. As all instructions are completed within five steps, a modulo-5 counter is employed.

The control signal generator, which is a combinational circuit, generates the necessary control signals based on all its inputs. These inputs include the outputs from the

instruction decoder (INS1 to INSm) and the outputs from the step counter (T1 to T5). By analyzing these inputs, the control signal generator determines the required settings of the control signals.

To determine the appropriate settings of the control signals, the action sequences for each instruction represented by the INS1 to INSm signals are considered. Based on these action sequences, the control signal generator configures the control signals to facilitate the execution of the specific instruction at hand.

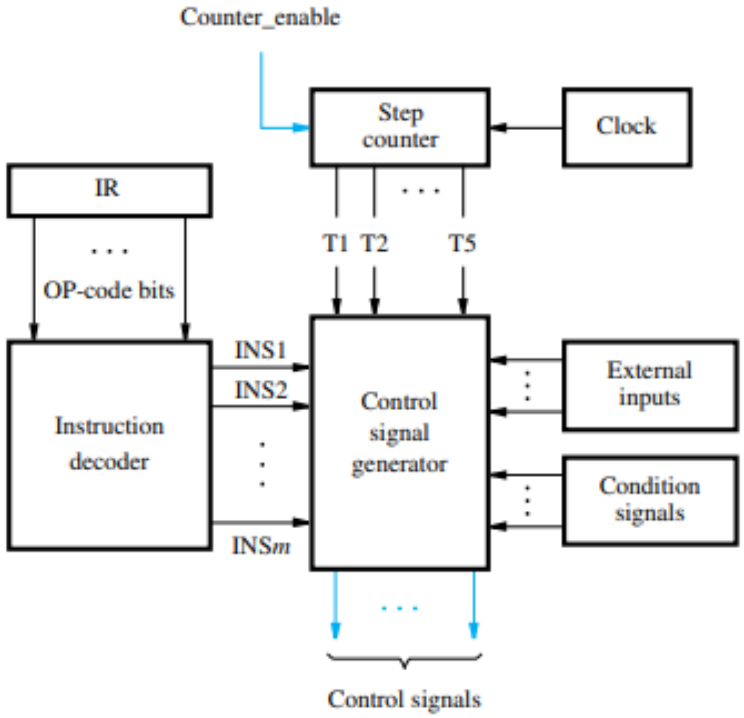


Figure 5.21 Generation of the control signals.

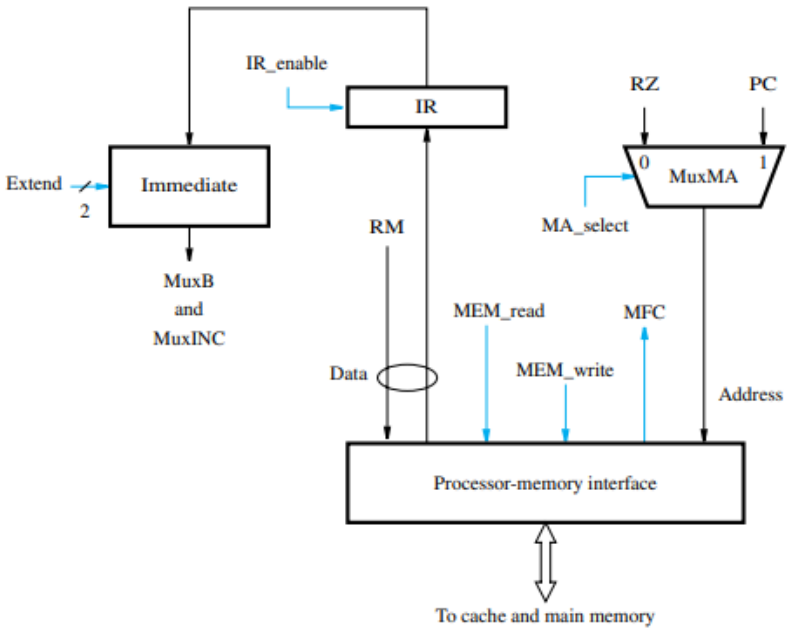


Figure 5.19 Processor-memory interface and IR control signals.

Operation's RTL

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).

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.

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).

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.