MODULE 4

Module IV (9 Hours)

The Processor - Introduction, Logic design conventions, Building a datapath, A simple implementation scheme, An overview of pipelining - Pipelined datapath and control - Structural hazards - Data hazards - Control hazards.
I/O organization - Accessing I/O devices, interrupts - handling multiple devices, Direct memory access

- J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.
- Hamacher, Vranesic & Zaky, “Computer Organization” (5th Ed), McGraw Hill.

RIT KOTTAYAM

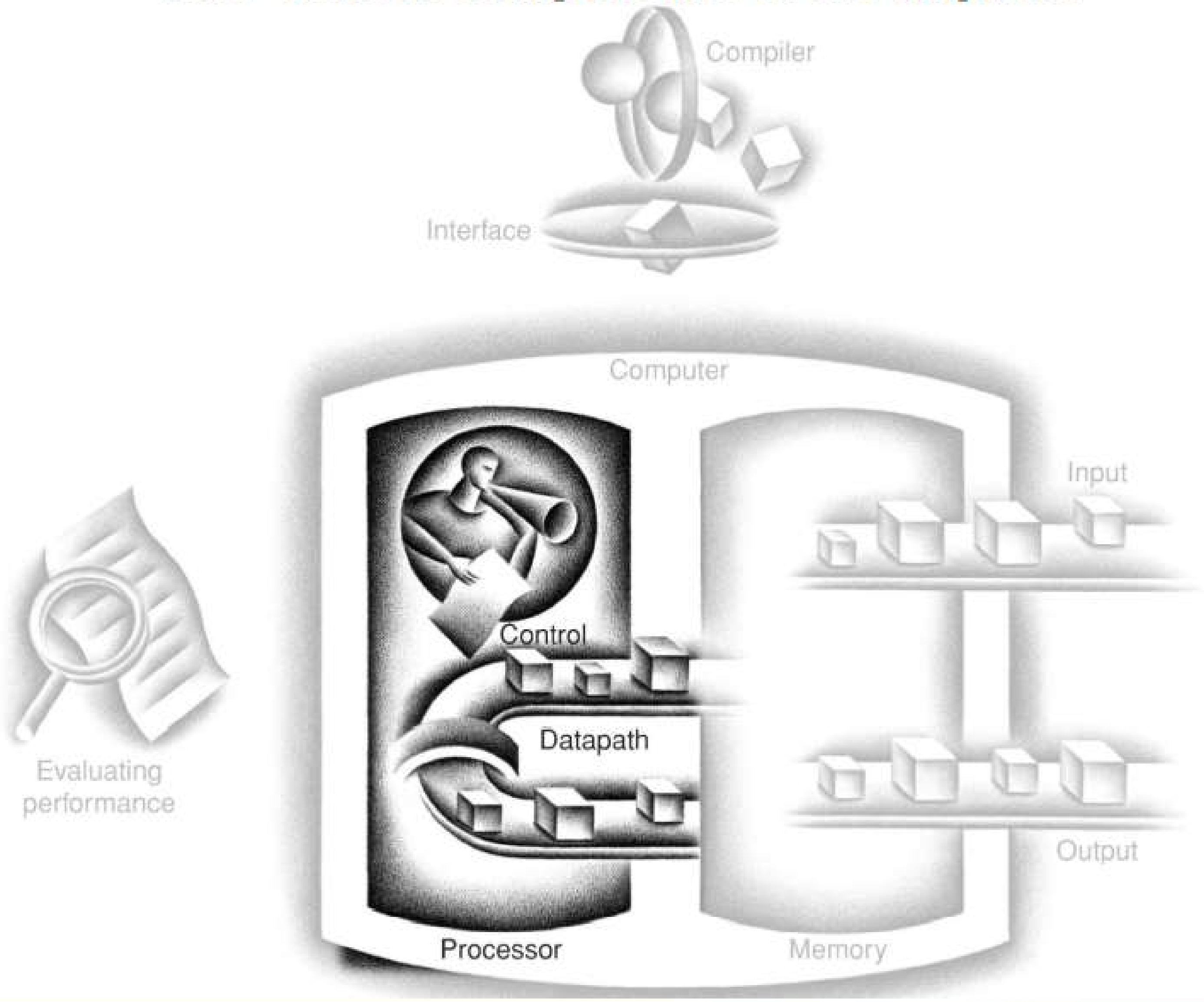
- MISSION
- To impart professional education to students from all sectors of the society to mould them to be competent, committed, research oriented and socially responsible citizens capable of deploying technology for the goodwill of society.

- VISION
- Be a centre of technological excellence for the betterment of society.

CO 4 : Understand Processor logic design conventions and data path, pipelining and hazards, I/O organization, Interrupts and direct memory access

Introduction

Five Classic Components of a Computer



Introduction

- The performance of a computer is determined by three key factors: instruction count, clock cycle time, and clock cycles per instruction (CPI).
- The compiler and the instruction set architecture determine the instruction count required for a given program.
- However, the implementation of the processor determines both the clock cycle time and the number of clock cycles per instruction.
- This module contains an explanation of the principles and techniques used in implementing a processor.

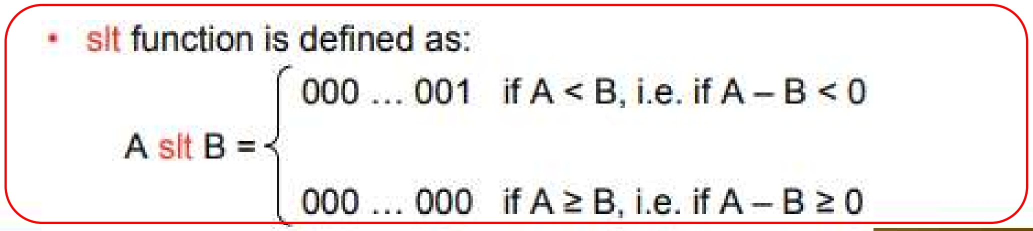
Introduction

MIPS (Microprocessor without Interlocked Pipeline Stages) is a type of computer processor architecture that uses a reduced instruction set computing (RISC) approach

A Basic MIPS Implementation

•We will be examining an implementation that includes a subset of the core MIPS instruction set:

- ■ The memory-reference instructions *load word (lw)* and *store word* (sw)
- ■ The arithmetic-logical instructions *add, sub, AND, OR,* and *slt* (set less than)
- ■ The instructions *branch equal (beq)* and *jump (j),* which we add last. This subset does not include all the integer instructions (for example, shift , multiply, and divide are missing), nor does it include any floating-point instructions.



• slt function is defined as:

$$A \text{ slt } B = \begin{cases} 000\ldots001 & \text{if } A < B \text{, i.e. if } A - B < 0 \\ 000\ldots000 & \text{if } A \geq B \text{, i.e. if } A - B \geq 0 \end{cases}$$

Introduction

A Basic MIPS Implementation

- In examining the implementation, we will have the opportunity to see how the instruction set architecture determines different aspects of the implementation, and how the choice of various implementation strategies affects the clock rate and CPI for the computer.

Introduction

A Basic MIPS Implementation

- For the implementation of every instructions (integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions), the first two steps are identical:

  1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.

  2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require reading two registers.

Introduction

A Basic MIPS Implementation

- After these two steps, the actions required to complete the instruction depend on the instruction class.
- For each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction.
- The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

Introduction

- For example, all instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers.
- The memory-reference instructions use the ALU for an address calculation, the arithmetic-logical instructions for the operation execution, and branches for comparison.
- After using the ALU, the actions required to complete various instruction classes differ.

Introduction

- A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
- An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register.
- For a branch instruction, we may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.

Introduction

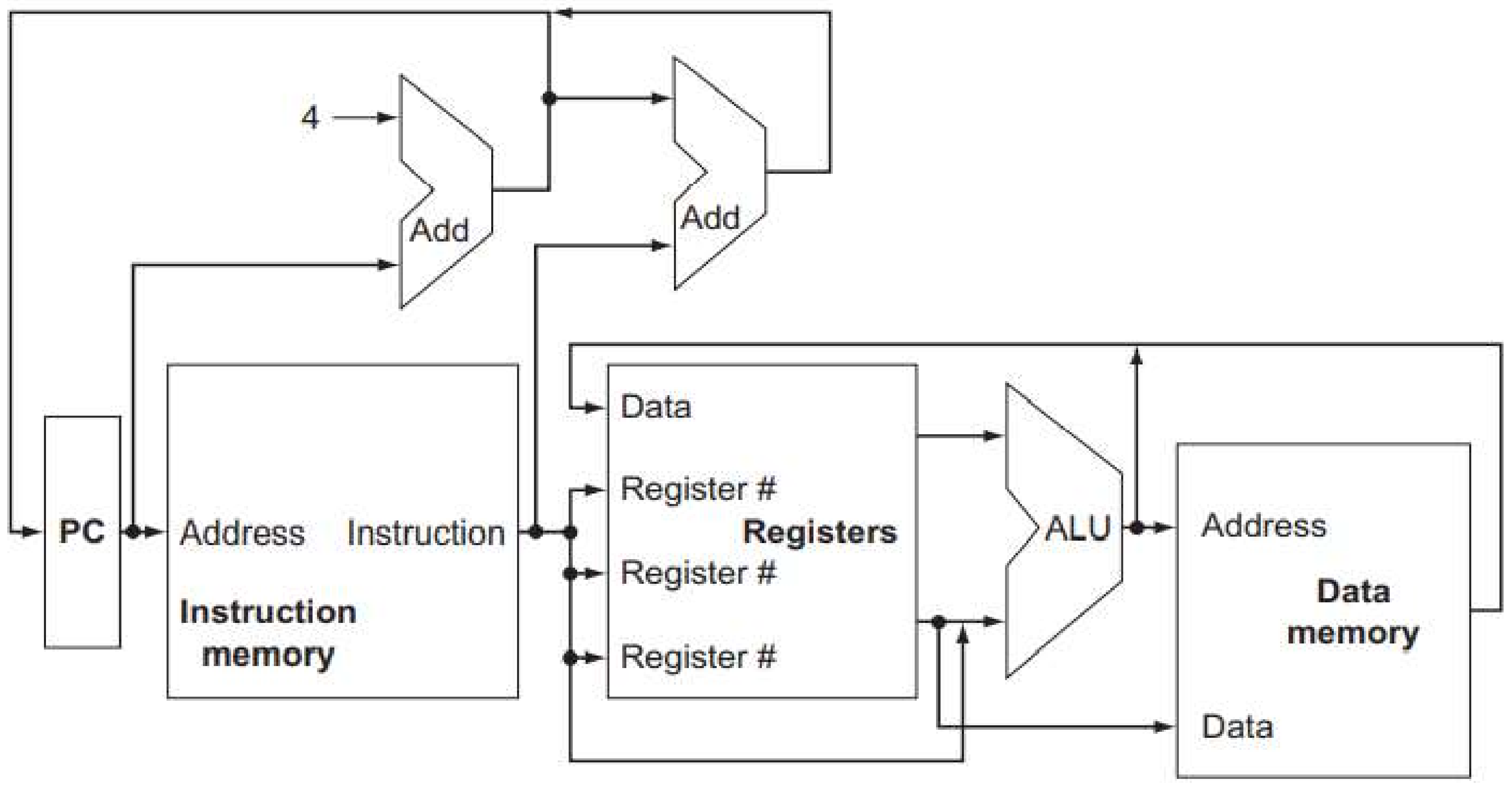


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

[J. Hennessy and D. Patterson, “Computer Organization and Design: The Hardware/Software Interface”, 5th Edition.]

Introduction

- All instructions start by using the program counter to supply the instruction address to the instruction memory.
- After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction.

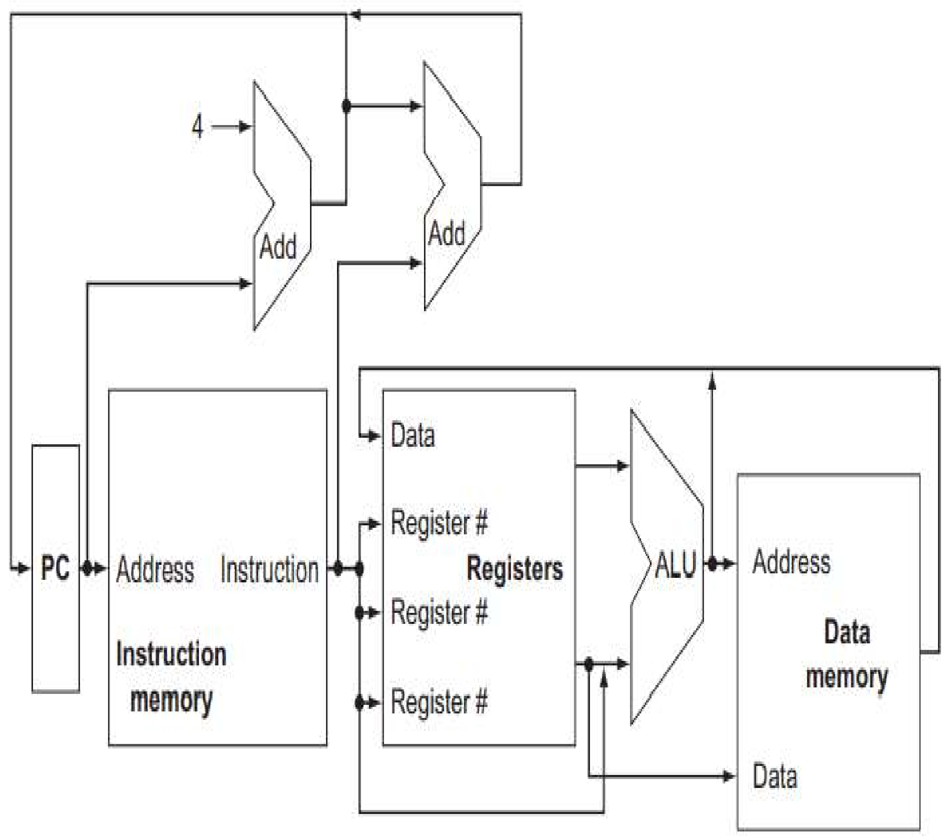


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Figure 1

Introduction

- Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch).
- If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register.

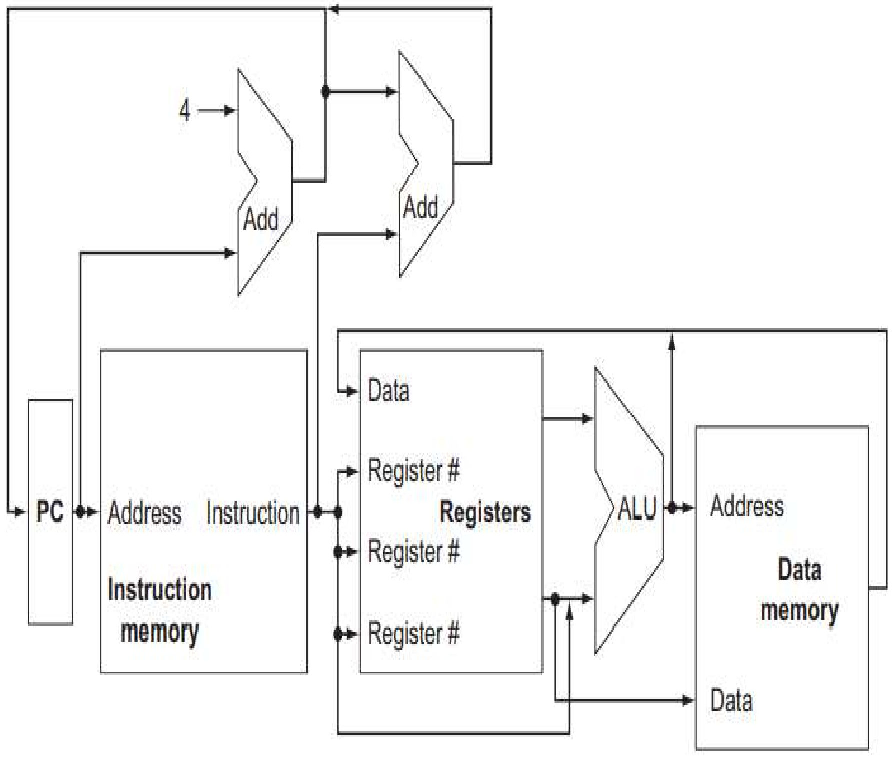


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Figure 1

Introduction

- If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers.
- The result from the ALU or memory is written back into the register file.
- Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch offset are summed) or from an adder that increments the current PC by 4.

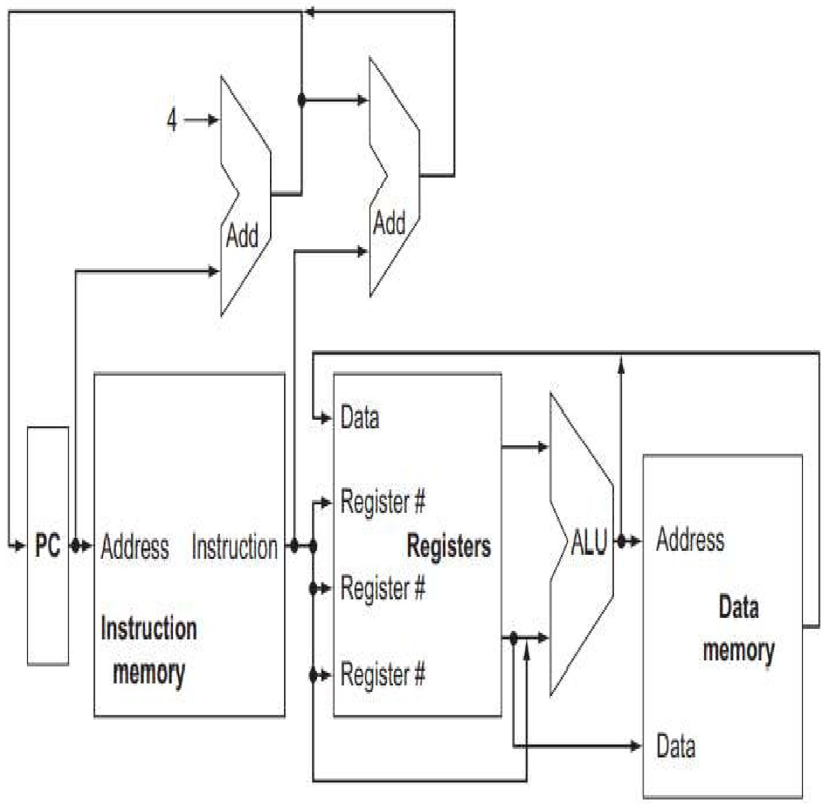


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Figure 1

Introduction

- The thick lines interconnecting the functional units represent buses, which consist of multiple signals.
- The arrows are used to guide the reader in knowing how information flows.
- Since signal lines may cross, figure explicitly shows that when crossing lines are connected by the presence of a dot where the lines cross.
- Although figure 1 shows most of the flow of data through the processor, it omits two important aspects of instruction execution.

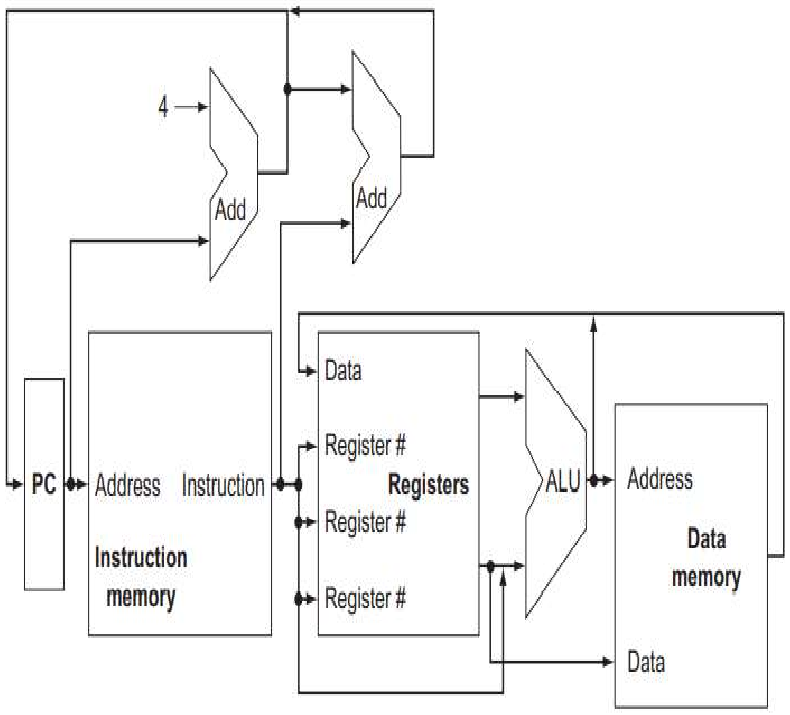


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Introduction

Figure 1

- First, in several places, Figure 1 shows data going to a particular unit as coming from two different sources.
- For example, the value written into the PC can come from one of two adders, the data written into the register file can come from either the ALU or the data memory, and the second input to the ALU can come from a register or the immediate field of the instruction.

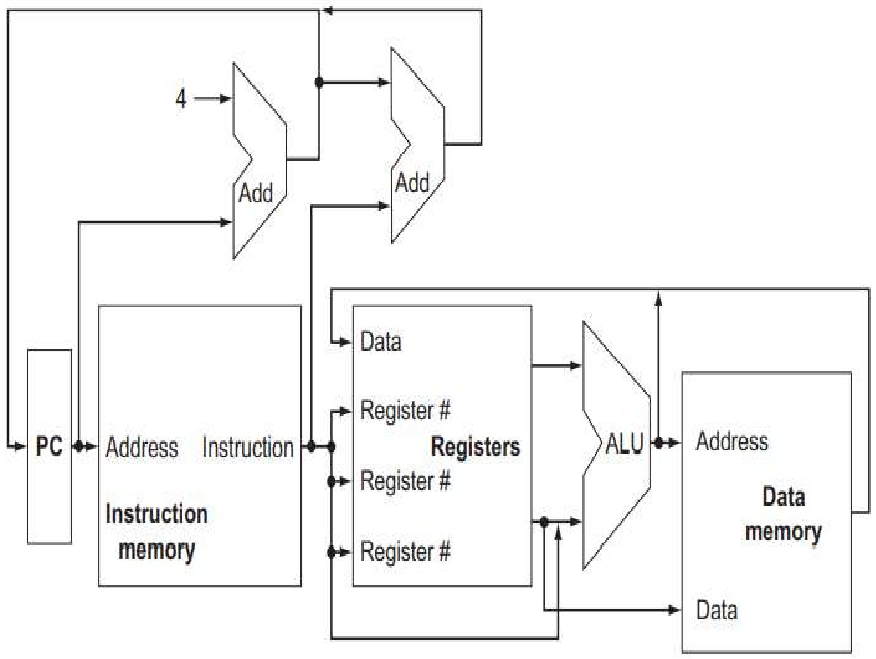


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Introduction

Figure 1

- In practice, these data lines cannot simply be wired together; we must add a logic element that chooses from among the multiple sources and steers one of those sources to its destination.
- This selection is commonly done with a device called a multiplexor, although this device might better be called a data selector.
- Multiplexor, which selects from among several inputs based on the setting of its control lines. The control lines are set based primarily on information taken from the instruction being executed.

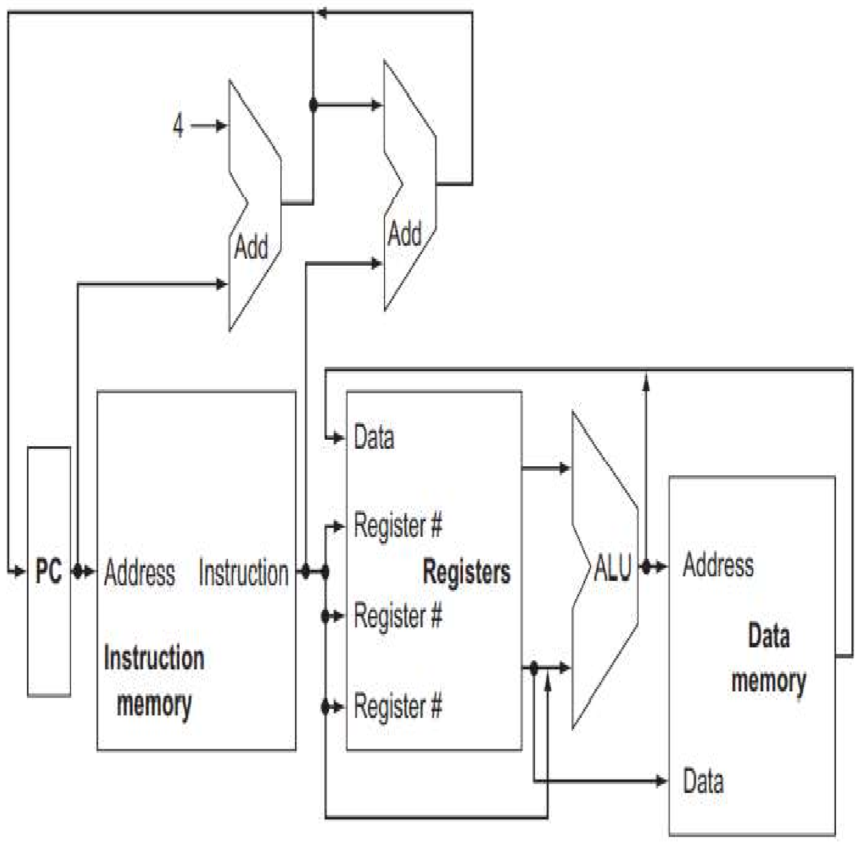


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Introduction

- The second omission in Figure 1 is that several of the units must be controlled depending on the type of instruction.
- For example, the data memory must read on a load and written on a store.
- The register file must be written only on a load or an arithmetic-logical instruction.
- The ALU must perform one of several operations.
- Like the multiplexors, control lines that are set on the basis of various fields in the instruction direct these operations.

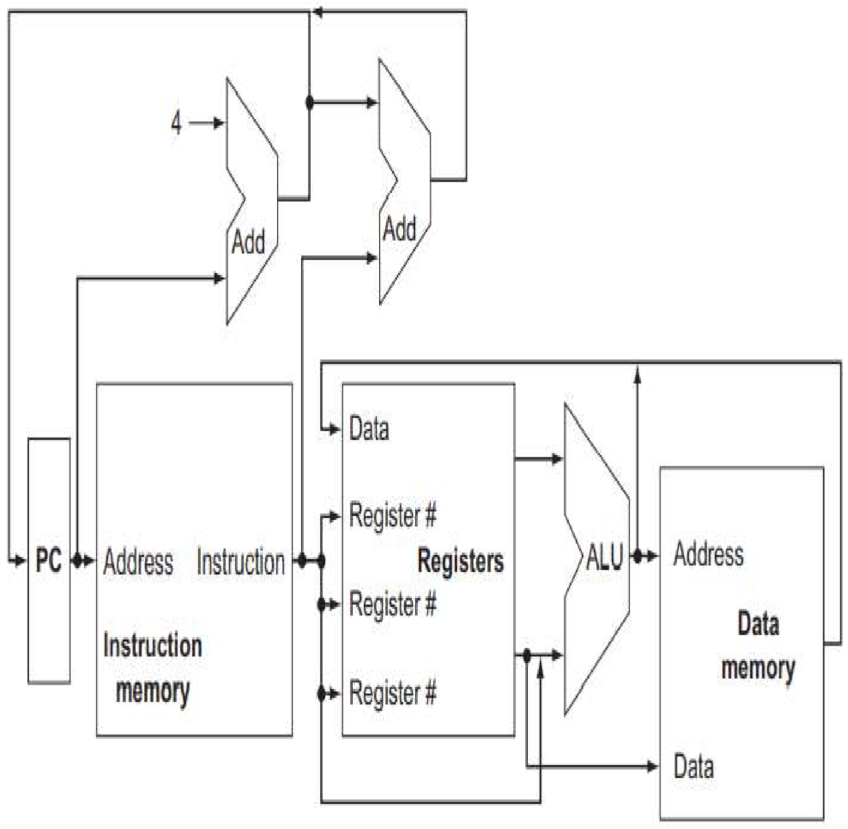
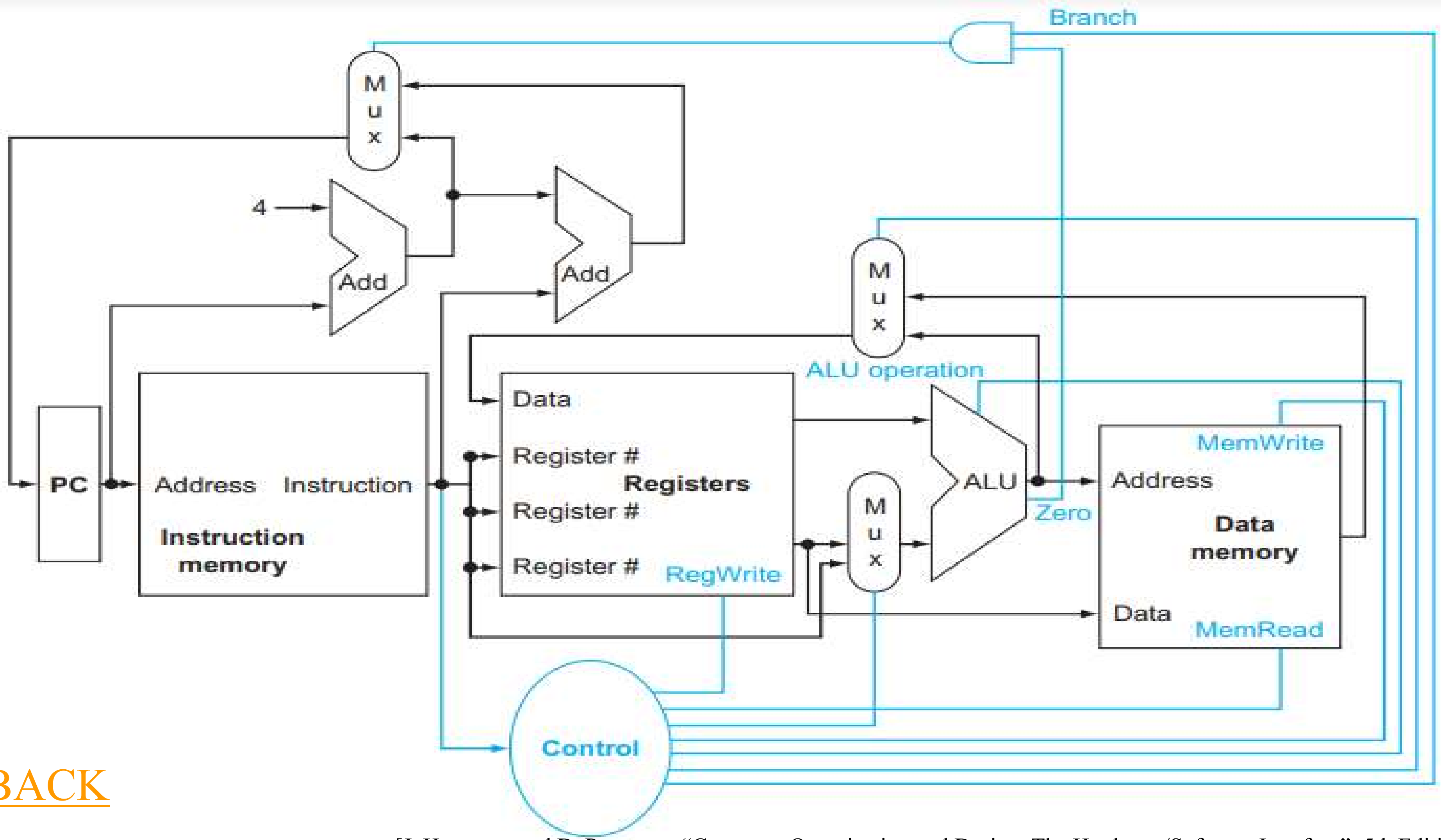


FIGURE 1: An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them.

Introduction

"branch if equal" which means "jump to given address if zero flag is set" such as when previous comparison is successful

Figure 2 shows the datapath of Figure 1 with the three required multiplexors added, as well as control lines for the major functional units.



BACK

[J. Hennessy and D. Patterson, "Computer Organization and Design: The Hardware/Software Interface", 5th Edition.]

FIGURE 2 The basic implementation of the MIPS subset, including the necessary multiplexors and control lines.

Introduction

- A control unit, which has the instruction as an input, is used to determine how to set the control lines for the functional units and two of the multiplexors.
- The third multiplexor, which determines whether PC + 4 or the branch destination address is written into the PC, is set based on the Zero output of the ALU, which is used to perform the comparison of a *beq* instruction.
- The regularity and simplicity of the MIPS instruction set means that a simple decoding process can be used to determine how to set the control lines.

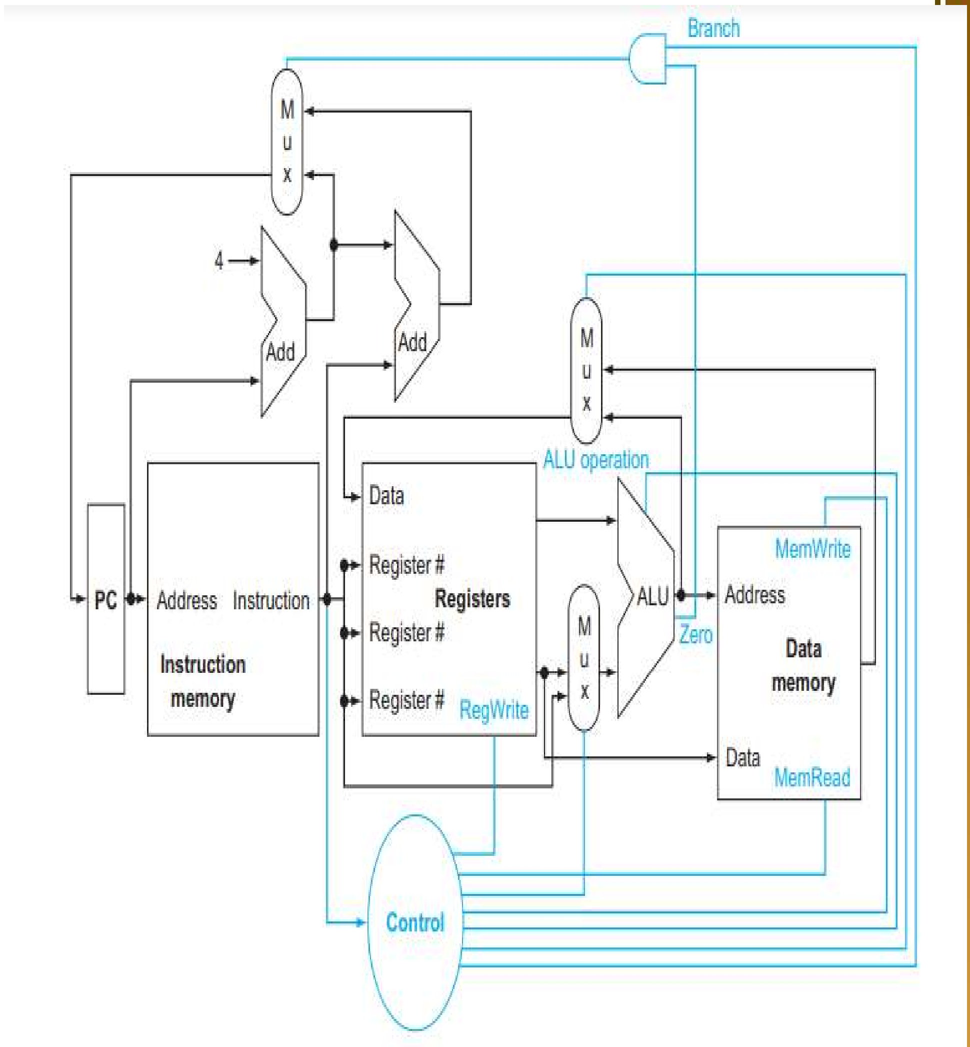


FIGURE 2 The basic implementation of the MIPS subset, including the necessary multiplexors and control lines.