**CPE 300L**

**Digital System Architecture And Design Laboratory**

**Laboratory 8**

**MIPS: Multi Cycle Implementation**

Department of Electrical and Computer Engineering

University of Nevada, Las Vegas

Goals

Understand Multi cycle MIPS processor.

Introduction

The single-cycle processor has three primary weaknesses. First, it requires a clock cycle long enough to support the slowest instruction (lw), even though most instructions are faster. Second, it requires three adders (one in the ALU and two for the PC logic); adders are relatively expensive circuits, especially if they must be fast. And third, it has separate instruction and data memories, which may not be realistic. Most computers have a single large memory that holds both instructions and data that can be read and written.

The multicycle processor addresses these weaknesses by breaking an instruction into multiple shorter steps. In each short step, the processor can read or write the memory or register file or use the ALU. Different instructions use different numbers of steps, so simpler instructions can be completed faster than more complex ones. The processor needs only one adder; this adder is reused for different purposes on various steps. And the processor uses a combined memory for instructions and data. The instruction is fetched from memory on the first step and data may be read or written on later steps.

We design a multicycle processor following the same procedure we used for the single-cycle processor. First, we construct a Datapath by connecting the architectural state elements and memories with combinational logic. But, this time, we also add nonarchitecture state elements to hold intermediate results between the steps. Then, we design the controller. During the execution of a single instruction, the controller produces different signals on each step, so now the controller uses a finite state machine rather than combinational logic. Finally, we analyze the performance.

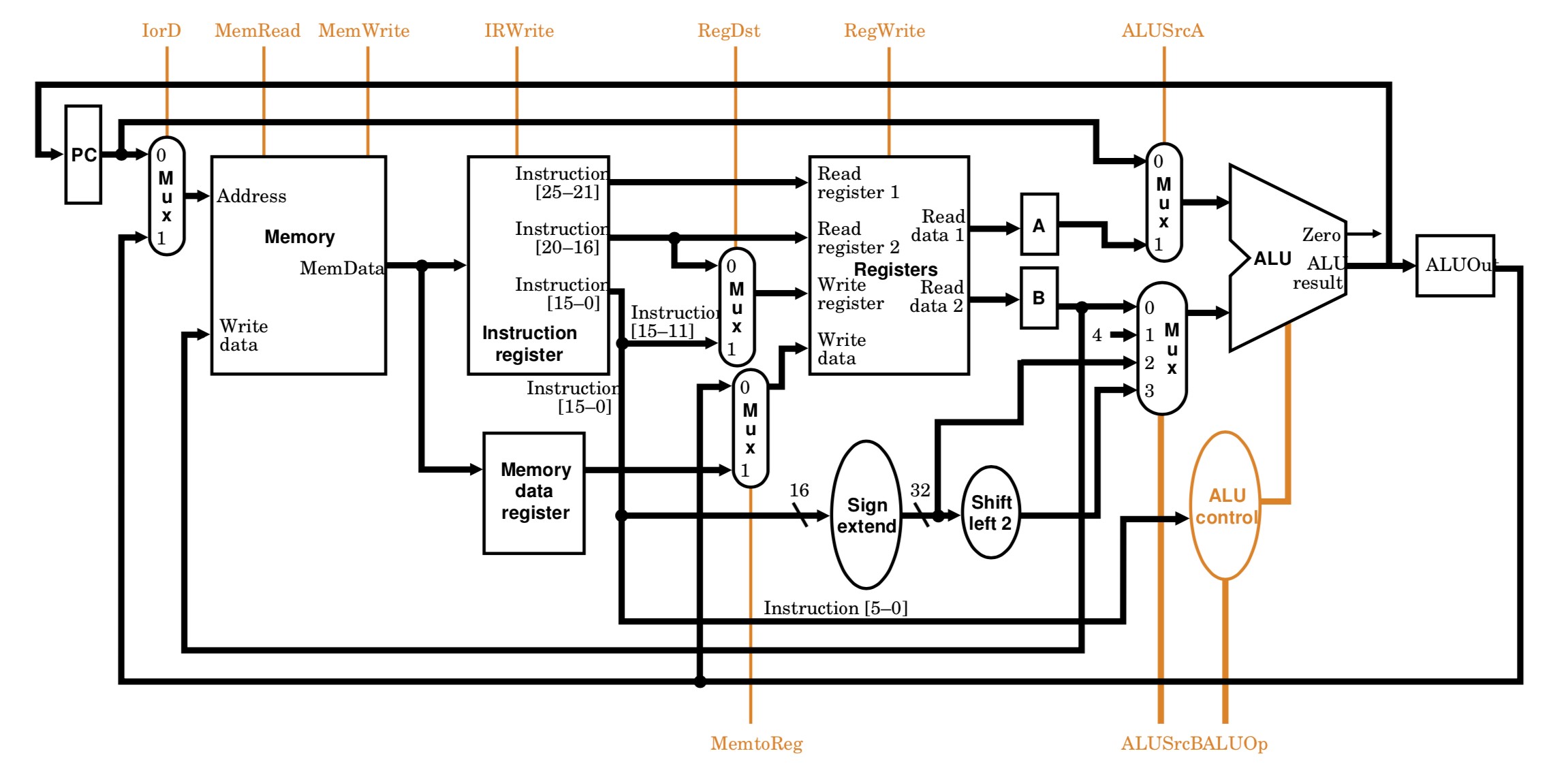
of the multicycle processor and compare it with the single-cycle processor.

A diagram of a computer

Description automatically generated

Fig. 1. MIPS Multi cycle design

* We now have a single memory element that interacts with both instructions and data.
* Single ALU unit, no dedicated adders



* Several temporary registers –Instruction Register, Memory data register, A, B.

.

Fig. 2. Multi-Cycle Datapath.

We have same old data path elements that we are already used to from single cycle implementation –Memory, PC, ALU, Registers, sign-extend, shift left 2. However, the memory element is now pulling double-duty as both the instruction memory and data memory element.

New temporary registers:

* **Instruction registers (IR)** – holds the instruction after it’s been pulled from memory.
* **Memory data register (MDR)** – temporarily holds data grabbed from memory until the next cycle.
* **A** – temporarily holds the contents of read register 1 until the next cycle.
* **B** – temporarily holds the contents of read register 2 until the next cycle.
* **ALUout** – temporarily holds the contents of the ALU until the next cycle.

Control Signals:

A diagram of a computer

Description automatically generated

Fig. 3. Multi-Cycle Datapath and control.

A screenshot of a computer error

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Fig. 4. MIPS multicycle control signals and its function table



Fig. 5. ALU opcode table

Unit Overview

The three units have the following inputs and outputs. Although the signal names are in upper case here to match the diagram, remember to use lower case for all names in your Verilog files.

|  |  |  |
| --- | --- | --- |
| CLK |  | Input |
| Reset |  | Input |
| MemWrite |  | Input |
| Adr | [5:0] | Input |
| WriteData | [31:0] | Input |
| ReadData | [31:0] | Input |

**Controller Datapath Memory**

|  |  |  |
| --- | --- | --- |
| CLK |  | Input |
| Reset |  | Input |
| Op | [5:0] | Input |
| Funct | [5:0] | Input |
| Zero |  | Input |
| lorD |  | Output |
| MemWrite |  | Output |
| IRWrite |  | Output |
| RegDst |  | Output |
| MemtoReg |  | Output |
| RegWrite |  | Output |
| ALUSrcA |  | Output |
| ALUSrcB | [1:0] | Output |
| ALUControl | [2:0] | Output |
| PCSrc | [1:0] | Output |
| PCEn |  | Output |

|  |  |  |
| --- | --- | --- |
| CLK |  | Input |
| Reset |  | Input |
| PCEn |  | Input |
| lorD |  | Input |
| IRWrite |  | Input |
| RegDst |  | Input |
| MemtoReg |  | Input |
| RegWrite |  | Input |
| ALUSrcA |  | Input |
| ALUSrcB | [1:0] | Input |
| ALUControl | [2:0] | Input |
| PCSrc | [1:0] | Input |
| ReadData | [31:0] | Input |
| Op | [5:0] | Output |
| Funct | [5:0] | Output |
| Zero |  | Output |
| Adr | [5:0] | Output |
| WriteData | [31:0] | Output |

FSM Diagram:



Example

R-Format

R-format instructions in MIPS multicycle require 4 cycles to complete. Let’s imagine that we’re executing an add instruction.

add $s0, $s1, $s2

Which has the following fields:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Opcode** | **Rs** | **Rt** | **Rd** | **Shamt** | **funct** |
| 000000 | 10001 | 10010 | 10000 | 00000 | 100000 |

**Cycle 1: Cycle 2: Cycle 4:**

|  |  |
| --- | --- |
| **PCWrite** | **1** |
| **lorD** | **0** |
| **MemRead** | **1** |
| **MemWrite** | **0** |
| **IRWrite** | **1** |
| **PCSource** | **00** |
| **ALUOp** | **00** |
| **ALUSrcB** | **01** |
| **ALUSrcA** | **0** |
| **RegWrite** | **0** |

|  |  |
| --- | --- |
| **ALUOp** | **00** |
| **ALUSrcB** | **11** |
| **ALUSrcA** | **0** |

|  |  |
| --- | --- |
| **MemtoReg** | **0** |
| **RegWrite** | **1** |
| **RegDst** | **1** |

**Cycle 3:**

|  |  |
| --- | --- |
| **ALUOp** | **10** |
| **ALUSrcB** | **00** |
| **ALUSrcA** | **1** |

Lab Deliveries

Prelab:

**1.** Draw the table for I and J type instruction. Use above tables as an example.

Lab Experiments

1. Adding the instruction

Implement the MIPS multicycle design with all the necessary components. Your design should include the instruction shown in above FSM diagram. You can add other instruction if you wish to.

Verify this operation in ModalSim and DE2.

See the next page for postlab report.

Postlab Report

Include the following elements in the report document:

|  |  |  |
| --- | --- | --- |
| **Section** | **Element** | |
| 1 | Theory of operation  *Include a brief description of every element and phenomenon that appears during the*  *experiments.*  *a. Describe the difference between MIPS single cycle processor vs MIPS multicycle processor.* | |
| 2 | Prelab report | |
| 3 | Results of the experiments | |
| **Experiment** | **Experiment Results** |
| 1 | * MIPS Code * Screenshots from ModalSim |
| 4 | Answer the questions | |
| **Question no.** | **Question** |
| 1 | What are the typical stages in MIPS multicycle implementation. List them all. |
| 5 | Conclusions  *Write down your conclusions, things learned, problems encountered during the lab and how they were solved, etc.* | |