CA Project Report

Processor Design and Simulation

TEAM 52

Amr Negm: 55-25195

Rabie Zewil: 55-10164

Ali Fadl: 55-22840

Ali Hossam: 55-6777

Mohamed Abdelbaki: 55-23897

Abdelrahman Zakzouk: 55-16760

Computer Systems Architecture

German University in Cairo

May 16, 2024

Contents

1 Introduction

1.1 Overview

In this project, we are tasked with simulating a fictional processor design and architecture using the C programming language. The processor designed for this simulation employs the Harvard architecture, which has different memory spaces and buses for instructions and data. This project involves parsing the text file containing the assembly instructions in addition to implementing and simulating the instruction fetch, decode, and execute stages of the processor pipeline.

1.2 Project Objectives

The primary objective of the Processor Simulation Project is to gain a deeper understanding of processor architecture, instruction execution, and pipeline operation through practical implementation. By simulating a fictional processor design, the project aims to reinforce concepts related to computer systems architecture and assembly pro-

gramming.

Key Functions

-parseText: Reads the instructions from a text file and converts them into binary format stored in instruction memory.

- fetch: Fetches the next instruction from instruction memory.

- decode: Decodes the fetched instruction to determine the operation and operands.

- execute: Executes the decoded instruction, performing ALU operations and updating registers, memory, and status flags.

2 Implementation Overview

2.1 Processor Components

The Harvard architecture features separate storage and separate buses for instructions and data, overcoming the bottleneck of the Von Neumann architecture by allowing simultaneous access to both instruction and data memory.

Memory Configuration:

Instruction Memory: 1024 words, each 16 bits wide (total 2 KB).

Data Memory: 2048 words, each 8 bits wide (total 2 KB).

Registers:

General-Purpose Registers (GPRs): 64 registers (R0 to R63), each 8 bits wide.

Status Register (SREG): 8 bits wide, with 5 flags used (Carry, Overflow, Negative, Sign, Zero).

Carry Flag (C) for ADD.

Overflow Flag (V) for ADD and SUB.

Negative Flag (N) for ADD, SUB, MUL, ANDI, EOR, SAL, SAR.

Sign Flag (S) for ADD and SUB.

Zero Flag (Z) for ADD, SUB, MUL, ANDI, EOR, SAL, SAR.

Program Counter (PC): 16 bits wide.

Instruction Set Architecture (ISA):

Instruction Size: 16 bits.

Instruction Types: Two main types with 12 different operations.

R-Format : 4 bits opcode , 6 bits first input register and 6 bits second input register

I-Format : 4 bits opcode , 6 bits first input register and 6 for the immediate value

3 Research methodology

The methodology for implementing the fictional processor simulation involves several steps, focusing on designing the processor architecture, creating the instruction set, and implementing a pipelined execution model. Below is a detailed explanation of the ideas and concepts behind the code:

Instruction Loading:

The program reads a text file containing assembly instructions. Each instruction is parsed and converted to a format that can be stored in the instruction memory.

Clock cycles:

Number of clock cycles: 3 + ((n - 1) * 1), where n = number of instructions

If we are in the first clock cycle we only fetch first instruction and we neither decode nor execute

In the second clock cycle we fetch the second instruction and decode the first instruction and we don't execute

Finally we reach the general case where in the third clock cycle we

fetch the third instruction, decode the second and execute the first instruction.

Pipeline Stages:

Fetch Stage: The current instruction is fetched based on the PC value.

Decode Stage: The fetched instruction is decoded to determine the operation and operands.

Execute Stage: The decoded instruction is executed, updating the appropriate registers or memory locations.

State Management:

Registers and memory are updated based on the executed instruction. The PC is incremented to point to the next instruction unless a control flow instruction (e.g., BR) modifies it.

Output Generation:

For each clock cycle, the state of the processor (including register values and memory content) is printed to the console. This output helps in understanding the behavior of the processor and verifying the correct execution of the instructions.

4 Results

After implementing the processor simulation program according to the specified architecture and instruction set, the next step is to analyze the results obtained from its execution. So here is an example of test files and their corresponding outputs

Data file :

BEQZ R1 1

SUB R1 R3

MUL R2 R4

STR R4 10

ADD R3 R2

Initial Values :

R1 = 0

R2 = -3

R3 = 3

R4 = 2

Results :

cycle 1

fetch instruction 1 : 16449

There is no instruction neither decoded nor executed in this cycle

_____

cycle 2

fetch instruction 2 : 4163

decode instruction 1 : 16449

There is no instruction executed in this cycle

_____

cycle 3

fetch instruction 3 : 8324

decode instruction 2 : 4163

execute instruction 1 : 16449

register 1 has : 0

A jump has occured

_____

cycle 4

fetch instruction 3 : 8324

There is no instruction neither decoded nor executed in this cycle

_____

cycle 5

fetch instruction 4 : 45322

decode instruction 3 : 8324

There is no instruction executed in this cycle

_____

cycle 6

fetch instruction 5 : 194

decode instruction 4 : 45322

execute instruction 3 : 8324

register 4 has : 2

register 2 before change : -3

register 2 after change : -6

_____

cycle 7

There is no instruction fetched in this cycle

decode instruction 5 : 194

execute instruction 4 : 45322

R4 : 2

memory 10 before change : 0

memory 10 after change : 2

_____

cycle 8

There is no instruction neither fetched nor decoded in this cycle

execute instruction 5 : 194

register 2 has : -6

register 3 before change : 3

register 3 after change : -3

SREG :

Carry iS : 0

Overflow iS : 0

Negative bit iS : 1

Sign Flag iS : 1

Zero Flag iS : 0

PC : 6

Explanation:

Cycle 1: The first instruction (BEQZ R1 1) is fetched but not decoded or executed.

Cycle 2: The second instruction (SUB R1 R3) is fetched and the first

one is decoded, but non is executed.

Cycle 3: The third instruction (MUL R2 R4) is fetched, the second is decoded, and the first executed Since R1 is zero, the branch condition is satisfied, causing a jump to the third instruction.

Cycle 4: The third instruction is fetched again due to the jump from the previous cycle but neither decoded nor executed.

Cycle 5: The fourth instruction (STR R2 10) is fetched, the third is decoded, and non is executed.

Cycle 6: The fifth instruction (ADD R3 R2) is fetched, and the fourth is decoded, and the third is executed. R2 is multiplied by R4, resulting in R2 being doubled to -6.

Cycle 7: Non is fetched since the instruction memory is empty, the fifth is decoded and the fourth is executed. The memory is accessed to store the updated value of R4 (which is 2) at memory address 10.

Cycle 8: Finlay the fifth instruction is executed.R2 is added to R3, and the result (R3 + R2 = 0) is stored in R3. The final execution cycle involves viewing the updated flags in the Status Register (SREG) and the Program Counter (PC). The flags are updated based on the results of previous instructions, and the PC is set to the address of the next instruction.

5 Conclusion

The implemented simulation successfully demonstrates the execution of instructions in a fictional processor design. The code follows the defined architecture and instruction set, providing insights into the functioning of the processor's data path and instruction execution flow.

Overall, the project provides a comprehensive understanding of processor simulation and showcases the practical implementation of concepts related to computer architecture and assembly programming.