Simulating a Pipelined Processor

COMPUTER ARCHITECTURE

Package 4

CSEN601

Authors:

Seif ElDin Diea Nassar (55-8584)
Yahia Tolba (55-8175)
Omar Ahmed (55-8819)
Amr Khaled (55-11434)
Abdullah Ashour (55-6603)
Ali Khaled (55-6888)

May 17, 2024

Contents

Introduction

The primary motivation for this project is to gain a deeper understanding of pipelined processor design and execution. By simulating a pipelined processor, we can:

- **Visualize the instruction fetch-decode-execute cycle:** The simulator will provide a step-by-step view of how instructions are processed within the pipeline stages.
- **Explore potential challenges:** The simulation can help identify potential issues like control, which are crucial considerations in real-world processor design.

1 Methodology

This project focuses on simulating a basic pipelined processor with a 3-stage pipeline (Fetch, Decode, Execute). The simulator will be able to:

- **Read assembly language instructions from a text file.**
- **Parse and decode instructions based on the ISA format.**
- **Execute instructions within the simulated pipeline.**
- **Handle basic data types and operations supported by the ISA.**
- **Detect and handle potential exceptions like overflow during arithmetic operations.**
- **Implement Control Hazards when using branching.**

1.1 Key Functions

- **load_instructions:** This function loads instructions from a text file. It takes a filename (`filename`) as input. It opens the file, reads each line, parses it using `parse_instruction`, and stores the resulting binary code in the `InstructionMemory` array.
- **parse_instructions:** This function parses an assembly instruction string and converts it into a binary code. It takes a character pointer (`assemblyCode`) as input. It uses strtok to tokenize the string based on spaces. It then compares the first token (mnemonic) with predefined instructions and assigns a corresponding binary code. Additionally, it extracts operands and combines them with the opcode to form the final instruction binary code.
- **fetch:** This function fetches an instruction from the `InstructionMemory` at the current program counter (`PC`). It increments the `PC` after fetching the instruction. It returns an `Instruction` struct containing the fetched instruction and its corresponding PC value.
- **decode:** This function decodes a fetched instruction. It takes an `Instruction` struct (`instruction`) as input. It extracts the opcode, operand1, and operand2 from the instruction binary code and stores them in a `DecodedInstruction` struct. It also includes the PC value from the original instruction.
- **execute:** This function executes the decoded instruction. It takes a `DecodedInstruction` struct (`decodedInstruction`) as input. It extracts the opcode and operands from the decoded instruction. Based on the opcode, it performs the corresponding operation (addition, subtraction, multiplication, etc.) It also updates the status register (`SREG`) according to the operation's result (carry, zero, overflow, negative flags). It updates the GPR registers and memory locations as required by the instruction. It also prints information about the executed instruction and its operands.
- **main:** This function is the main entry point of the program. It performs the following tasks:
  - Loads instructions from a file named "instructions.txt".
  - Initializes the GPR registers with some sample values.
  - Initializes the DataMemory with a sample value.
  - Finds the total number of instructions in memory.
  - Simulates a CPU pipeline with a fetch, decode, and execute cycle. It uses several helper variables (`instructionToBeFetched`, `instructionToBeDecoded`, `instructionToBeExecuted`) to manage the pipeline stages.
  - After execution, it prints the contents of all GPRs, Instruction Memory, and Data Memory.

1.2 Memory Architecture

The memory and registers have the following sizes:

Registers

- **Size**: 8 bits (1 byte)
- **Type**: `int8_t`
- **Number**:
  - General Purpose Registers (GPRs): 64 (`GPRS[0]` to `GPRS[63]`)
  - Special Purpose Registers:
    - Program Counter (PC): `short int unsigned` (16 bits, 2 bytes) - holds the address of the next instruction to be fetched.
    - Status Register (SREG): `uint8_t` (8 bits, 1 byte) - holds flags indicating the status of the previous operation (carry, zero, overflow, negative).

Memory

- **Instruction Memory**:
  - **Size**: 1024 elements of `short int` (16 bits, 2 bytes per element)
- **Data Memory**:
  - **Size**: 2048 elements of `int8_t` (8 bits, 1 byte per element)

1.3 Pipelined Execution for Improved Performance

This section describes a pipelined execution model implemented to enhance the performance of a simulated CPU. Pipelining allows the CPU to overlap the execution of multiple instructions, potentially improving instruction throughput compared to a non-pipelined approach.

1.3.1 Pipeline Stages

The implemented pipeline consists of three distinct stages:

1. Fetch: In this stage, the CPU retrieves an instruction from the Instruction Memory at the address pointed to by the Program Counter (PC). The fetch function is responsible for this operation.
2. Decode: The fetched instruction is decoded in this stage. The opcode, operands, and any additional control information needed for execution are extracted. The Decode function performs this task.
3. Execute: During the execute stage, the decoded instruction is executed. This involves performing the operation specified by the opcode, updating registers or memory as required, and setting the status register (SREG) based on the outcome of the operation. The Execute function handles this stage.

1.3.2 Implementation Details

The pipelining strategy is achieved using several techniques:

Instruction Buffers: Three instruction buffers (instructionToBeFetched, instructionToBeDecoded, and instructionToBeExecuted) are employed to store instructions at different pipeline stages. These buffers hold an Instruction struct containing the instruction's binary code and its corresponding PC value. Delayed Execution: After fetching an instruction, it's assigned to the instructionToBeDecoded buffer in the fetch function. Similarly, after decoding, the instruction is assigned to the instructionToBeExecuted buffer in the Decode function. This introduces a one-cycle delay between fetch and decode, and another one-cycle delay between decode and execute. PC Update: The PC is incremented within the fetch function after retrieving an instruction. This ensures the next fetch cycle retrieves the subsequent instruction in the program sequence.

1.3.3 Benefits and Limitations

Pipelining offers the potential to execute instructions concurrently, improving overall instruction throughput. However, there are limitations to consider:

Data Hazards: The current implementation doesn't explicitly handle data hazards. These occur when an instruction relies on the result of a previous instruction that hasn't been executed yet. In such scenarios, the pipeline might need to stall (delay) to avoid using outdated data.

This section has described a basic pipelined execution model implemented for a simulated CPU. While this implementation offers performance improvements, there's room for further development to address data hazards for a more robust pipelined CPU design.

2 Results

2.1 Instructions

```
ADD R1 R2
SUB R3 R4
MUL R5 R6
ANDI R7 3
EOR R8 R9
MOVI R10 42
BEQZ R11 1
ADD R12 R13
SAL R12 2
SAR R13 2
LDR R14 20
STR R14 25
BR R15 R16
ADD R1 R2
ADD R1 R2
ADD R1 R2
ADD R1 R2
ADD R1 R2
ADD R1 R2
ADD R17 R18
```

2.2 Initial values

- **General Purpose Registers (GPRS)**
  - GPRS[1] = 10
  - GPRS[2] = 5
  - GPRS[3] = 20
  - GPRS[4] = 4
  - GPRS[5] = 6
  - GPRS[6] = 7
  - GPRS[7] = 15
  - GPRS[8] = 9
  - GPRS[9] = 5
  - GPRS[10] = 42
  - GPRS[11] = 0
  - GPRS[12] = 3
  - GPRS[13] = -16
  - GPRS[14] = 75
  - GPRS[15] = 0
  - GPRS[16] = 15

  - GPRS[17] = 30
  - GPRS[18] = 12

- **Data Memory**
  - DataMemory[20] = 55

2.3 Output

Cycle 1

```
Fetched Instruction 0
PC: 0
----------------------------Cycle 1 ended-----------------------------
```

Cycle 2

```
Fetched Instruction 1
PC: 1
Decoded Instruction 0
----------------------------Cycle 2 ended-----------------------------
```

Cycle 3

```
Fetched Instruction 2
PC: 2
Decoded Instruction 1
Old Values : R1 = 10 , R2 = 5
Operation ADD : R1 = R1 + R2
New Value of R1 = 15
SREG: XXXCVNSZ
      00000000
Executed Instruction 0
----------------------------Cycle 3 ended-----------------------------
```

Cycle 4

```
Fetched Instruction 3
PC: 3
Decoded Instruction 2
Old Values : R3 = 20 , R4 = 4
Operation SUB : R3 = R3 - R4
New Value of R3 = 16
SREG: XXXCVNSZ
      00000000
Executed Instruction 1
----------------------------Cycle 4 ended-----------------------------
```

Cycle 5

```
Fetched Instruction 4
PC: 4
Decoded Instruction 3
Old Values : R5 = 6 , R6 = 7
Operation MUL : R5 = R5 * R6
New Value of R5 = 42
```

```
SREG: XXXCVNSZ
      00000000
Executed Instruction 2
----------------------------Cycle 5 ended-----------------------------
```

Cycle 6

```
Fetched Instruction 5
PC: 5
Decoded Instruction 4
Old Values : R7 = 15 , IMM = 3
Operation ANDI : R7 = R7 & 3
New Value of R7 = 3
SREG: XXXCVNSZ
      00000000
Executed Instruction 3
----------------------------Cycle 6 ended-----------------------------
```

Cycle 7

```
Fetched Instruction 6
PC: 6
Decoded Instruction 5
Old Values : R8 = 9 , R9 = 5
Operation EOR : R8 = R8 XOR R9
New Value of R8 = 12
SREG: XXXCVNSZ
      00000000
Executed Instruction 4
----------------------------Cycle 7 ended-----------------------------
```

Cycle 8

```
Fetched Instruction 7
PC: 7
Decoded Instruction 6
Operation MOVI : R10 = 42
SREG: XXXCVNSZ
      00000000
Executed Instruction 5
----------------------------Cycle 8 ended-----------------------------
```

Cycle 9

```
Fetched Instruction 8
PC: 8
Decoded Instruction 7
Operation BEQZ : IF(R11==0){
         PC + 1 + 1}
 This Instruction PC was : 6
New Value of PC = 8
Pipeline flushed!
SREG: XXXCVNSZ
      00000000
Executed Instruction 6
----------------------------Cycle 9 ended-----------------------------
```

Cycle 10

```
Fetched Instruction 8
PC: 8
----------------------------Cycle 10 ended-----------------------------
```

Cycle 11

```
Fetched Instruction 9
PC: 9
Decoded Instruction 8
----------------------------Cycle 11 ended-----------------------------
```

Cycle 12

```
Fetched Instruction 10
PC: 10
Decoded Instruction 9
Old Values : R12 = 3 , IMM = 2
Operation SAL : R12 = R12 << 2
New Value of R12 = 12
SREG: XXXCVNSZ
      00000000
Executed Instruction 8
----------------------------Cycle 12 ended-----------------------------
```

Cycle 13

```
Fetched Instruction 11
PC: 11
Decoded Instruction 10
Old Values : R13 = -16 , IMM = 2
Operation SAR : R13 = R13 >> 2
New Value of R13 = -4
SREG: XXXCVNSZ
      00000100
Executed Instruction 9
----------------------------Cycle 13 ended-----------------------------
```

Cycle 14

```
Fetched Instruction 12
PC: 12
Decoded Instruction 11
Operation LDR : R14 = MEM[14]
New Value of R14 = 55
SREG: XXXCVNSZ
      00000000
Executed Instruction 10
----------------------------Cycle 14 ended-----------------------------
```

Cycle 15

```
Fetched Instruction 13
PC: 13
```

```
Decoded Instruction 12
Operation STR : MEM[25] = R14
New Value of MEM[25] = 55
SREG: XXXCVNSZ
      00000000
Executed Instruction 11
----------------------------Cycle 15 ended-------------------------------
```

Cycle 16

```
Fetched Instruction 14
PC: 14
Decoded Instruction 13
Old Values : R15 = 0 , R16 = 15
Operation BR : PC = R15 CONCAT R16 , PC = 15
New Value of PC = 15
Pipeline flushed!
SREG: XXXCVNSZ
      00000000
Executed Instruction 12
----------------------------Cycle 16 ended-------------------------------
```

Cycle 17

```
Fetched Instruction 15
PC: 15
----------------------------Cycle 17 ended-------------------------------
```

Cycle 18

```
Fetched Instruction 16
PC: 16
Decoded Instruction 15
----------------------------Cycle 18 ended-------------------------------
```

Cycle 19

```
Fetched Instruction 17
PC: 17
Decoded Instruction 16
Old Values : R1 = 15 , R2 = 5
Operation ADD : R1 = R1 + R2
New Value of R1 = 20
SREG: XXXCVNSZ
      00000000
Executed Instruction 15
----------------------------Cycle 19 ended-------------------------------
```

Cycle 20

```
Fetched Instruction 18
PC: 18
Decoded Instruction 17
Old Values : R1 = 20 , R2 = 5
Operation ADD : R1 = R1 + R2
New Value of R1 = 25
```

```
SREG: XXXCVNSZ
      00000000
Executed Instruction 16
----------------------------Cycle 20 ended-----------------------------
```

Cycle 21

```
Fetched Instruction 19
PC: 19
Decoded Instruction 18
Old Values : R1 = 25 , R2 = 5
Operation ADD : R1 = R1 + R2
New Value of R1 = 30
SREG: XXXCVNSZ
      00000000
Executed Instruction 17
----------------------------Cycle 21 ended-----------------------------
```

Cycle 22

```
PC: 20
Decoded Instruction 19
Old Values : R1 = 30 , R2 = 5
Operation ADD : R1 = R1 + R2
New Value of R1 = 35
SREG: XXXCVNSZ
      00000000
Executed Instruction 18
----------------------------Cycle 22 ended-----------------------------
```

Cycle 23

```
PC: 21
Old Values : R17 = 30 , R18 = 12
Operation ADD : R17 = R17 + R18
New Value of R17 = 42
SREG: XXXCVNSZ
      00000000
Executed Instruction 19
----------------------------Cycle 23 ended-----------------------------
```

All Registers

```
Register 0: 0 | Register 1: 35 | Register 2: 5 | Register 3: 16 | Register 4: 4 |
Register 5: 42 | Register 6: 7 | Register 7: 3 | Register 8: 12 | Register 9: 5 |
Register 10: 42 | Register 11: 0 | Register 12: 12 | Register 13: -4 | Register 14: 55 |
Register 15: 0 | Register 16: 15 | Register 17: 42 | Register 18: 12 | Register 19: 0 |
Register 20: 0 | Register 21: 0 | Register 22: 0 | Register 23: 0 | Register 24: 0 |
Register 25: 0 | Register 26: 0 | Register 27: 0 | Register 28: 0 | Register 29: 0 |
Register 30: 0 | Register 31: 0 | Register 32: 0 | Register 33: 0 | Register 34: 0 |
Register 35: 0 | Register 36: 0 | Register 37: 0 | Register 38: 0 | Register 39: 0 |
Register 40: 0 | Register 41: 0 | Register 42: 0 | Register 43: 0 | Register 44: 0 |
Register 45: 0 | Register 46: 0 | Register 47: 0 | Register 48: 0 | Register 49: 0 |
Register 50: 0 | Register 51: 0 | Register 52: 0 | Register 53: 0 | Register 54: 0 |
Register 55: 0 | Register 56: 0 | Register 57: 0 | Register 58: 0 | Register 59: 0 |
Register 60: 0 | Register 61: 0 | Register 62: 0 | Register 63: 0 |
```

Instruction Memory

```
Instruction 0: 4162 | Instruction 1: 8388 | Instruction 2: 12614 | Instruction 3: 25027 |
Instruction 4: 29193 | Instruction 5: 17066 | Instruction 6: 21185 | Instruction 7: 4877 |
Instruction 8: -27902 | Instruction 9: -23742 | Instruction 10: -19564 | Instruction 11: -15463 |
Instruction 12: -31792 | Instruction 13: 4162 | Instruction 14: 4162 | Instruction 15: 4162 |
Instruction 16: 4162 | Instruction 17: 4162 | Instruction 18: 4162 | Instruction 19: 5202 |
```

Data Memory

```
MEM[20]: 55 | MEM[25]: 55 |
```