German University in Cairo
Media Engineering and Technology
Assoc. Prof. Dr. Hassan Soubra

CSEN 601: Computer System Architecture, Spring Semester 2023
Project Description
Deadline: 10/6/2023 11:59 PM

Project Overview

"Processor design is the design engineering task of creating a processor, a key component of computer hardware. The design process involves choosing an instruction set and a certain execution paradigm, and results in a microarchitecture. The mode of operation of any processor is the execution of lists of instructions. Instructions typically include those to compute or manipulate data values using registers, change or retrieve values in read/write memory, perform relational tests between data values and to control program flow."

In this project, you will simulate a fictional processor design and architecture using **Java**. You are asked to choose **one of four processor packages** described in the upcoming sections.

Project Instructions

Please read the following instructions carefully:

a) Any case of plagiarism will result in a zero.

b) Any case of cheating will result in a zero.

c) A **cheating detection** tool will be used to compare the submitted projects against all **online and offline implementations** similar to the project idea.

- The projects that have more than **50% similarity** percentage will receive a zero.

d) It is your responsibility to ensure that you have:

- Submitted before the deadline.
- Submitted the correct file(s).
- Submitted the correct file(s) names.

e) The project deadline is on **Saturday 10/06/2023 at 11:59 pm**.

1 Processor's Description

1.1 Memory Architecture

a) **Architecture:** Von Neumann

- Von Neumann Architecture is a digital computer architecture whose design is based on the concept of stored program computers where **program data** and **instruction data** are stored in the **same memory**.

b) **Memory Size:** 2048 * 32

Main Memory	
2048 Rows	**32 Bits / Row**
	Data (1024 to 2047)
	Instructions (0 to 1023)

- The main memory addresses are from 0 to $2^{11} - 1$ (0 to 2047).
- Each memory block (row) contains 1 word which is 32 bits (4 bytes).
- The main memory is word addressable.
- Addresses from 0 to 1023 contain the program instructions.
- Addresses from 1024 to 2048 contain the data.

Registers: 33

- Size: 32 bits
- 31 General-Purpose Registers (GPRS)
  - Names: R1 to R31
- 1 Zero Register
  - Name: R0
  - Hard-wired value "0" (cannot be overwritten by any instruction).
- 1 Program Counter
  - Name: PC
  - A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time.
  - As each instruction gets fetched, the program counter is incremented to point to the next instruction to be executed.

1.2 Instruction Set Architecture

a) **Instruction Size:** 32 bits

b) **Instruction Types:** 3

R-Format				
OPCODE	**R1**	**R2**	**R3**	**SHAMT**
4	5	5	5	13

I-Format			
OPCODE	**R1**	**R2**	**IMMEDIATE**
4	5	5	18

J-Format	
OPCODE	**ADDRESS**
4	28

c) **Instruction Count:** 12

- The opcodes are from 0 to 11 according to the instructions order in the following table:

Name	Mnemonic	Type	Format	Operation
Add	ADD	R	ADD R1 R2 R3	R1 = R2 + R3
Subtract	SUB	R	SUB R1 R2 R3	R1 = R2 - R3
Multiply	MUL	R	MUL R1 R2 R3	R1 = R2 * R3
Move Immediate*	MOVI	I	MOVI R1 IMM	R1 = IMM
Jump if Equal	JEQ	I	JEQ R1 R2 IMM	IF(R1 == R2) { PC = PC+1+IMM }
And	AND	R	AND R1 R2 R3	R1 = R2 & R3
Exclusive Or Immediate	XORI	I	XORI R1 R2 IMM	R1 = R2 $\oplus$ IMM
Jump	JMP	J	JMP ADDRESS	PC = PC[31:28] || ADDRESS
Logical Shift Left**	LSL	R	LSL R1 R2 SHAMT	R1 = R2 << SHAMT
Logical Shift Right**	LSR	R	LSR R1 R2 SHAMT	R1 = R2 >>> SHAMT
Move to Register	MOVR	I	MOVR R1 R2 IMM	R1 = MEM[R2 + IMM]
Move to Memory	MOVM	I	MOVM R1 R2 IMM	MEM[R2 + IMM] = R1

* MOVI: R2 will be 0 in the instruction format.
** LSL and LSR: R3 will be 0 in the instruction format.
"||" symbol indicates concatenation (0100 || 1100 = 01001100).

1.3 Datapath

a) **Stages:** 5

- All instructions regardless of their type must pass through all 5 stages even if they do not need to access a particular stage.
- **Instruction Fetch (IF):** Fetches the next instruction from the main memory using the address in the PC (Program Counter), and increments the PC.
- **Instruction Decode (ID):** Decodes the instruction and reads any operands required from the register file.
- **Execute (EX):** Executes the instruction. In fact, all ALU operations are done in this stage.
- **Memory (MEM):** Performs any memory access required by the current instruction. For loads, it would load an operand from the main memory, while for stores, it would store an operand into the main memory.

- **Write Back (WB):** For instructions that have a result (a destination register), the Write Back writes this result back to the register file.

b) **Pipeline:** 4 instructions (maximum) running in parallel

- **Instruction Fetch (IF)** and **Memory (MEM)** can not be done in parallel since they access the same physical memory.
- At a given clock cycle, you can either have the **IF, ID, EX, WB** stages active, or the **ID, EX, MEM, WB** stages active.
- **Number of clock cycles:** $7 + ((n-1) * 2)$, where n = number of instructions
  - Imagine a program with 7 instructions:
    - $7 + (6 * 2) = 19$ clock cycles
  - You are required to understand the pattern in the example and implement it.

Package 3 Pipeline					
	Instruction Fetch (IF)	**Instruction Decode (ID)**	**Execute (EX)**	**Memory (MEM)**	**Write Back (WB)**
Cycle 1	Instruction 1				
Cycle 2		Instruction 1			
Cycle 3	Instruction 2	Instruction 1			
Cycle 4		Instruction 2	Instruction 1		
Cycle 5	Instruction 3	Instruction 2	Instruction 1		
Cycle 6		Instruction 3	Instruction 2	Instruction 1	
Cycle 7	Instruction 4	Instruction 3	Instruction 2		Instruction 1
Cycle 8		Instruction 4	Instruction 3	Instruction 2	
Cycle 9	Instruction 5	Instruction 4	Instruction 3		Instruction 2
Cycle 10		Instruction 5	Instruction 4	Instruction 3	
Cycle 11	Instruction 6	Instruction 5	Instruction 4		Instruction 3
Cycle 12		Instruction 6	Instruction 5	Instruction 4	
Cycle 13	Instruction 7	Instruction 6	Instruction 5		Instruction 4
Cycle 14		Instruction 7	Instruction 6	Instruction 5	
Cycle 15		Instruction 7	Instruction 6		Instruction 5
Cycle 16			Instruction 7	Instruction 6	
Cycle 17			Instruction 7		Instruction 6
Cycle 18				Instruction 7	
Cycle 19					Instruction 7

- The pattern is as follows:
  - You fetch an instruction every 2 clock cycles starting from clock cycle 1.
  - An instruction stays in the Decode (ID) stage for 2 clock cycles.
  - An instruction stays in the Execute (EX) stage for 2 clock cycles.
  - An instruction stays in the Memory (MEM) stage for 1 clock cycle.
  - An instruction stays in the Write Back (WB) stage for 1 clock cycle.
  - You can not have the Instruction Fetch (IF) and Memory (MEM) stages working in parallel. Only one of them is active at a given clock cycle.

Guidelines

The following guidelines must be followed in all packages:

Program Flow

a) You must write your program in **assembly language** in a **text file**.

b) Your must read the instructions from the text file, and parse them according to their types/formats (opcode and other relevant fields).

c) You must store the parsed version of the instructions in the memory (instruction segment of main memory or instruction memory according to your package).

d) You should start the execution of your **pipelined implementation** by **fetching** the first instruction from the **memory** (instruction segment of main memory or instruction memory) at **Clock Cycle 1**.

e) You should continue the execution based on the example provided in the **Datapath section** of each package reflecting the different stages working in parallel.

f) The Clock Cycles can be simulated as a variable that is incremented after finishing the required stages at a given time.

- Example:

```
fetch();
decode();
execute();
// memory();
// writeback();

cycles++;
```

Printings

The following items **must be printed** in the console after each Clock Cycle:

a) The Clock Cycle number.

b) The Pipeline stages:

- Which instruction is being executed at each stage?
- What are the input parameters/values for each stage?

c) The updates occurring to the registers in case a register value was changed.

d) The updates occurring in the memory (data segment of main memory or data memory according to your package) in case a value was stored or updated in the memory.

e) The content of all registers **after the last clock cycle**.

f) The full content of the memory (main memory or instruction and data memories according to your package) **after the last clock cycle**.

Submission

You should submit **a ZIP** file to the course email containing the following items:

- All ".java" code files used in the project.
- Any additional library used.
- A text file containing the team number, team name, package number and name, and team members' names, IDs, and tutorials.

The ZIP file should be named in the following format: Team_[TeamNumber]

Email Subject: Team_[TeamNumber]
Example: Team_ 15
Submission Email: csen601s23berlin@gmail.com