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

Computer System Architecture, Spring Semester 2023
Project Description
Processor X Meals
Deadline: Saturday 27/05/2023 at 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.

Package Selection Form: `https://forms.gle/XsqrP4vzxuQuRU1E7`
Package Selection Deadline: Tuesday 28/03/2023 at 11:59 pm
Note: The processor package choice is based on a **first come, first served** basis *(each package has a maximum capacity of 26 teams).*

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 27/05/2023 at 11:59 pm**.

1 Package 1: Spicy Von Neumann Fillet *with extra shifts*

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	
	32 Bits / Row
2048 Rows	**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.

c) **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 Immediate	MULI	I	MULI R1 R2 IMM	R1 = R2 * IMM
Add Immediate	ADDI	I	ADDI R1 R2 IMM	R1 = R2 + IMM
Branch if Not Equal	BNE	I	BNE R1 R2 IMM	IF(R1 != R2) { PC = PC+1+IMM }
And Immediate	ANDI	I	ANDI R1 R2 IMM	R1 = R2 & IMM
Or Immediate	ORI	I	ORI R1 R2 IMM	R1 = R2 | IMM
Jump	J	J	J ADDRESS	PC = PC[31:28] || ADDRESS
Shift Left Logical*	SLL	R	SLL R1 R2 SHAMT	R1 = R2 << SHAMT
Shift Right Logical*	SRL	R	SRL R1 R2 SHAMT	R1 = R2 >>> SHAMT
Load Word	LW	I	LW R1 R2 IMM	R1 = MEM[R2 + IMM]
Store Word	SW	I	SW R1 R2 IMM	MEM[R2 + IMM] = R1

* SLL and SRL: 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 1 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.

2 Package 2: Double McHarvard *with cheese circular shifts*

2.1 Memory Architecture

a) **Architecture:** Harvard

- Harvard Architecture is the digital computer architecture whose design is based on the concept where there are **separate storage** and **separate buses (signal path)** for **instruction** and **data**. It was basically developed to overcome the bottleneck of Von Neumann Architecture.

b) **Instruction Memory Size:** 1024 * 16

Instruction Memory	
1024 Rows	**16 Bits / Row**

- The instruction memory addresses are from 0 to $2^{10} - 1$ (0 to 1023).
- Each memory block (row) contains 1 word which is 16 bits (2 bytes).

- The instruction memory is word addressable.
- The program instructions are stored in the instruction memory.

c) **Data Memory Size:** 2048 * 8

Data Memory	
	8 Bits / Row
2048 Rows	

- The data memory addresses are from 0 to $2^{11} - 1$ (0 to 2047).
- Each memory block (row) contains 1 word which is 8 bits (1 byte).
- The data memory is word/byte addressable (1 word = 1 byte).
- The data is stored in the data memory.

d) **Registers:** 66

- Size: 8 bits
- 64 General-Purpose Registers (GPRS)
  - Names: R0 to R63
- 1 Status Register

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	C	V	N	S	Z

  - Name: SREG
  - A status register, flag register, or condition code register (CCR) is a collection of status flag bits for a processor.
  - The status register has 5 flags updated after the execution of specific instructions:
    - Carry Flag (C): Indicates when an arithmetic carry or borrow has been generated out of the most significant bit position.
      - Check on 9th bit (bit 8) of UNSIGNED[VALUE1] OP UNSIGNED[VALUE2] == 1 or not.
      - Example: https://piazza.com/class/le8zgqxowmd6e7/post/17
    - Two's Complement Overflow Flag (V): Indicates when the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
      - If 2 numbers are added, and they both have the same sign (both positive or both negative), then overflow occurs (V = 1) **if and only if** the result has **the opposite sign**. Overflow never occurs when adding operands with different signs.
      - If 2 numbers are subtracted, and their signs are different, then overflow occurs (V = 1) **if and only if** the result has the same sign as the **subtrahend**.
      - The difference between carry and overflow is explained in: https://piazza.com/class/le8zgqxowmd6e7/post/18
    - Negative Flag (N): Indicates a negative result in an arithmetic or logic operation.
      - N = 1 if result is negative.
      - N = 0 if result is positive or zero.
    - Sign Flag (S): Indicates the expected sign of the result (not the actual sign).
      - $S = N \oplus V$ (XORing the negative and overflow flags will calculate the sign flag).
    - Zero Flag (Z): Indicates that the result of an arithmetic or logical operation was zero.
      - Z = 1 if result is 0.

· Z = 0 if result is not 0.

∗ Since all registers are 8 bits, and we are only using 5 bits in the Status Register for the flags, you are required to keep Bits7:5 cleared "0" at all times in the register.

- 1 Program Counter
  - Name: PC
  - Type: Special-purpose register with a size of 16 bits (not 8 bits).
  - 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.

2.2 Instruction Set Architecture

a) **Instruction Size:** 16 bits

b) **Instruction Types:** 2

R-Format		
OPCODE	**R1**	**R2**
4	6	6

I-Format		
OPCODE	**R1**	**IMMEDIATE**
4	6	6

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	R1 = R1 + R2
Subtract	SUB	R	SUB R1 R2	R1 = R1 - R2
Multiply	MUL	R	MUL R1 R2	R1 = R1 * R2
Load Immediate	LDI	I	LDI R1 IMM	R1 = IMM
Branch if Equal Zero	BEQZ	I	BEQZ R1 IMM	IF(R1 == 0) { PC = PC+1+IMM }
And	AND	R	AND R1 R2	R1 = R1 & R2
Or	OR	R	OR R1 R2	R1 = R1 | R2
Jump Register	JR	R	JR R1 R2	PC = R1 || R2
Shift Left Circular	SLC	I	SLC R1 IMM	R1 = R1 << IMM | R1 >>> 8 - IMM
Shift Right Circular	SRC	I	SRC R1 IMM	R1 = R1 >>> IMM | R1 << 8 - IMM
Load Byte	LB	I	LB R1 ADDRESS	R1 = MEM[ADDRESS]
Store Byte	SB	I	SB R1 ADDRESS	MEM[ADDRESS] = R1

"||" symbol indicates concatenation (0100 || 1100 = 01001100).

d) The Status Register (SREG) flags are affected by the following instructions:

- The Carry flag (C) is updated every ADD instruction.
- The Overflow flag (V) is updated every ADD and SUB instruction.
- The Negative flag (N) is updated every ADD, SUB, MUL, AND, OR, SLC, and SRC instruction.

- The Sign flag (S) is updated every ADD and SUB instruction.
- The Zero flag (Z) is updated every ADD, SUB, MUL, AND, OR, SLC, and SRC instruction.
- A flag value can only be updated by the instructions related to it.

2.3 Datapath

a) **Stages:** 3

- All instructions regardless of their type must pass through all 3 stages.
- **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. Moreover, it 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. Finally, for instructions that have a result (a destination register), it writes this result back to the register file.

b) **Pipeline:** 3 instructions (maximum) running in parallel

- **Number of clock cycles:** $3 + ((n - 1) * 1)$, where n = number of instructions
  - Imagine a program with 7 instructions:
    - $3 + (6 * 1) = 9$ clock cycles
  - You are required to understand the pattern in the example and implement it.

Package 2 Pipeline			
	Instruction Fetch (IF)	**Instruction Decode (ID)**	**Execute (EX)**
Cycle 1	Instruction 1		
Cycle 2	Instruction 2	Instruction 1	
Cycle 3	Instruction 3	Instruction 2	Instruction 1
Cycle 4	Instruction 4	Instruction 3	Instruction 2
Cycle 5	Instruction 5	Instruction 4	Instruction 3
Cycle 6	Instruction 6	Instruction 5	Instruction 4
Cycle 7	Instruction 7	Instruction 6	Instruction 5
Cycle 8		Instruction 7	Instruction 6
Cycle 9			Instruction 7

3 Package 3: Fillet-O-Neumann *with moves on the side*

3.1 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.

c) **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.

3.2 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).

3.3 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.

4 Package 4: Double Big Harvard *combo large arithmetic shifts*

4.1 Memory Architecture

a) **Architecture:** Harvard

- Harvard Architecture is the digital computer architecture whose design is based on the concept where there are **separate storage** and **separate buses (signal path)** for **instruction** and **data**. It was basically developed to overcome the bottleneck of Von Neumann Architecture.

b) **Instruction Memory Size:** 1024 * 16

Instruction Memory	
	16 Bits / Row
1024 Rows	

- The instruction memory addresses are from 0 to $2^{10} - 1$ (0 to 1023).
- Each memory block (row) contains 1 word which is 16 bits (2 bytes).

- The instruction memory is word addressable.
- The program instructions are stored in the instruction memory.

c) **Data Memory Size:** 2048 * 8

Data Memory	
2048 Rows	**8 Bits / Row**

- The data memory addresses are from 0 to $2^{11} - 1$ (0 to 2047).
- Each memory block (row) contains 1 word which is 8 bits (1 byte).
- The data memory is word/byte addressable (1 word = 1 byte).
- The data is stored in the data memory.

d) **Registers:** 66

- Size: 8 bits
- 64 General-Purpose Registers (GPRS)
  - Names: R0 to R63
- 1 Status Register

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	C	V	N	S	Z

  - Name: SREG
  - A status register, flag register, or condition code register (CCR) is a collection of status flag bits for a processor.
  - The status register has 5 flags updated after the execution of specific instructions:
    * Carry Flag (C): Indicates when an arithmetic carry or borrow has been generated out of the most significant bit position.
      · Check on 9th bit (bit 8) of UNSIGNED[VALUE1] OP UNSIGNED[VALUE2] == 1 or not.
      · Example: https://piazza.com/class/le8zgqxowmd6e7/post/17
    * Two's Complement Overflow Flag (V): Indicates when the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
      · If 2 numbers are added, and they both have the same sign (both positive or both negative), then overflow occurs (V = 1) **if and only if** the result has **the opposite sign**. Overflow never occurs when adding operands with different signs.
      · If 2 numbers are subtracted, and their signs are different, then overflow occurs (V = 1) **if and only if** the result has the same sign as the **subtrahend**.
      · The difference between carry and overflow is explained in: https://piazza.com/class/le8zgqxowmd6e7/post/18
    * Negative Flag (N): Indicates a negative result in an arithmetic or logic operation.
      · N = 1 if result is negative.
      · N = 0 if result is positive or zero.
    * Sign Flag (S): Indicates the expected sign of the result (not the actual sign).
      · $S = N \oplus V$ (XORing the negative and overflow flags will calculate the sign flag).
    * Zero Flag (Z): Indicates that the result of an arithmetic or logical operation was zero.
      · Z = 1 if result is 0.

        · Z = 0 if result is not 0.

      * Since all registers are 8 bits, and we are only using 5 bits in the Status Register for the flags, you are required to keep Bits7:5 cleared "0" at all times in the register.

- 1 Program Counter
  - Name: PC
  - Type: Special-purpose register with a size of 16 bits (not 8 bits).
  - 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.

4.2 Instruction Set Architecture

a) **Instruction Size:** 16 bits

b) **Instruction Types:** 2

R-Format		
OPCODE	**R1**	**R2**
4	6	6

I-Format		
OPCODE	**R1**	**IMMEDIATE**
4	6	6

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	R1 = R1 + R2
Subtract	SUB	R	SUB R1 R2	R1 = R1 - R2
Multiply	MUL	R	MUL R1 R2	R1 = R1 * R2
Move Immediate	MOVI	I	MOVI R1 IMM	R1 = IMM
Branch if Equal Zero	BEQZ	I	BEQZ R1 IMM	IF(R1 == 0) { PC = PC+1+IMM }
And Immediate	ANDI	I	ANDI R1 IMM	R1 = R1 & IMM
Exclusive Or	EOR	R	EOR R1 R2	R1 = R1 ⊕ R2
Branch Register	BR	R	BR R1 R2	PC = R1 || R2
Shift Arithmetic Left	SAL	I	SAL R1 IMM	R1 = R1 << IMM
Shift Arithmetic Right	SAR	I	SAR R1 IMM	R1 = R1 >> IMM
Load to Register	LDR	I	LDR R1 ADDRESS	R1 = MEM[ADDRESS]
Store from Register	STR	I	STR R1 ADDRESS	MEM[ADDRESS] = R1

"||" symbol indicates concatenation (0100 || 1100 = 01001100).

d) The Status Register (SREG) flags are affected by the following instructions:

- The Carry flag (C) is updated every ADD instruction.
- The Overflow flag (V) is updated every ADD and SUB instruction.
- The Negative flag (N) is updated every ADD, SUB, MUL, ANDI, EOR, SAL, and SAR instruction.
- The Sign flag (S) is updated every ADD and SUB instruction.
- The Zero flag (Z) is updated every ADD, SUB, MUL, ANDI, EOR, SAL, and SAR instruction.
- A flag value can only be updated by the instructions related to it.

4.3 Datapath

a) **Stages:** 3

- All instructions regardless of their type must pass through all 3 stages.
- **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. Moreover, it 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. Finally, for instructions that have a result (a destination register), it writes this result back to the register file.

b) **Pipeline:** 3 instructions (maximum) running in parallel

- **Number of clock cycles:** $3 + ((n - 1) * 1)$, where n = number of instructions
  - Imagine a program with 7 instructions:
    - $3 + (6 * 1) = 9$ clock cycles
  - You are required to understand the pattern in the example and implement it.

Package 4 Pipeline

	Instruction Fetch (IF)	**Instruction Decode (ID)**	**Execute (EX)**
Cycle 1	Instruction 1		
Cycle 2	Instruction 2	Instruction 1	
Cycle 3	Instruction 3	Instruction 2	Instruction 1
Cycle 4	Instruction 4	Instruction 3	Instruction 2
Cycle 5	Instruction 5	Instruction 4	Instruction 3
Cycle 6	Instruction 6	Instruction 5	Instruction 4
Cycle 7	Instruction 7	Instruction 6	Instruction 5
Cycle 8		Instruction 7	Instruction 6
Cycle 9			Instruction 7

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]_[PackageNumber]
- *PackageNumber = {1, 2, 3, or 4} based on the selected/assigned project.*

Email Subject: Team_[TeamNumber]_[PackageNumber]
Example: Team_15_2
Submission Email: csen601.2023@gmail.com