BIRZEIT UNIVERSITY

Department of Electrical and Computer Engineering
Computer Organization and Microprocessor (ENCS2380)
Project: Single Cycle Processor Design (**INEMER**)
Due date: Sunday Dec. 22, 2024 @ 11:59 PM

Objectives:

After finishing this project, you should be able to:

- Design a 16-bit processor with 16-bit instructions
- Use the Logisim simulator to model and test the single cycle processor
- Work in a team

Instruction Set Architecture:

You are required to design a RISC processor that has seven 16-bit registers (i.e. **R1** to **R7**). Register **R0** is hardwired/connected to zero, where reading **R0** always returns zero value and writing **R0** has no effect. The written value to **R0** is neglected.

All instructions are **16-bit** long and aligned in memory. Memory is **16-bit** addressable. The **PC** register (**16-bit wide**) contains the instruction address. There are four instruction formats: **R-type**, **I-type**, **SB-type**, and **J-type** as shown below:

R-type:

5-bit opcode (Op), 3-bit register numbers (*rs1*, *rs2*, and *rd*), and 2-bit function field (*f*)

f^2		rs2^3			rs1^3			rd^3			Op^5				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

I-type:

5-bit opcode (Op), 3-bit register numbers (*rs1* and *rd*) and 5-bit Immediate

imm5					rs1^3			rd^3			Op^5				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

SB-type:

5-bit opcode (Op), 3-bit register numbers (*rs1* and *rs2*) and 5-bit Immediate split into (imm2U and imm3L)

imm2U^2		rs2^3			rs1^3			imm3L^3			Op^5				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

J-type:

5-bit opcode (Op) and 11-bit Immediate

Imm11^{11}											Op^5				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Register Use:

Register	**Number**	**Definition**	**Function**
Register d (Rd)	Destination register	**Rd** is the name and value of register **rd**	This register is always written (never read)
Register s1 (Rs1)	First source register	**Ra** is the name and value of register **rs1**	This register is always read (never written)
Register s2 (Rs2)	Second source register	**Rb** is the name and value of register **rs2**	This register is always read (never written)

Instruction Encoding:

The R-type, I-type, SB-type, and J-type instructions, definitions, and encodings are shown below:

Instruction	**Meaning**	**Encoding**				
XOR	Rd=Ra ^ Rb	f=0	rs2	rs1	rd	Op=0
OR	Rd=Ra | Rb	f=1	rs2	rs1	rd	Op=0
AND	Rd=Ra & Rb	f=2	rs2	rs1	rd	Op=0
SLL	Rd=ShiftLeftLogical(Ra, Rb[3:0])	f=0	rs2	rs1	rd	Op=1
SRL	Rd=ShiftRightLogical(Ra, Rb[3:0])	f=1	rs2	rs1	rd	Op=1
SRA	Rd=ShiftRightArith(Ra, Rb[3:0])	f=2	rs2	rs1	rd	Op=1
ADD	Rd=Ra + Rb	f=0	rs2	rs1	rd	Op=2
SUB	Rd=Ra – Rb	f=1	rs2	rs1	rd	Op=2
SLT	Rd=(Ra < Rb) signed	f=2	rs2	rs1	rd	Op=2
SLTU	Rd=(Ra < Rb) unsigned	f=3	rs2	rs1	rd	Op=2
ADDI	Rd=Ra + sign_extend(imm5)	imm5		rs1	rd	Op=3
SLTI	Rd=(Ra < sign_extend(imm5)) signed	imm5		rs1	rd	Op=4
SLTIU	Rd=(Ra < sign_extend(imm5)) unsigned	imm5		rs1	rd	Op=5
XORI	Rd=Ra ^ zero_extend(imm5)	imm5		rs1	rd	Op=6
ORI	Rd=Ra | zero_extend(imm5)	imm5		rs1	rd	Op=7
ANDI	Rd=Ra & zero_extend(imm5)	imm5		rs1	rd	Op=8
SLLI	Rd=ShiftLeftLogical(Ra, sa)	0	sa	rs1	rd	Op=9
SRLI	Rd=ShiftRightLogical(Ra, sa)	0	sa	rs1	rd	Op=10
SRAI	Rd=ShiftRightArith(Ra, sa)	0	sa	rs1	rd	Op=11
LW	Rd=Mem[Ra+imm5]	imm5		rs1	rd	Op=12
SW	Mem[Ra+{imm2U, imm3L}]=Rb	imm2	rs2	rs1	imm3	Op=13
BEQ	if (Ra == Rb) PC=PC+sign_extend({imm2U, imm3L})	imm2	rs2	rs1	imm3	Op=14
BNE	if (Ra != Rb) PC=PC+sign_extend({imm2U, imm3L})	imm2	rs2	rs1	imm3	Op=15
BLT	if (Ra < Rb) PC=PC+sign_extend({imm2U, imm3L})	imm2	rs2	rs1	imm3	Op=16
BGE	if (Ra >= Rb) PC=PC+sign_extend({imm2U, imm3L})	imm2	rs2	rs1	imm3	Op=17

BLTU	if (Ra < Rb) unsigned PC=PC+sign_extend({imm2U, imm3L})	imm2	rs2	rs1	imm3	Op=18
BGEU	if (Ra >= Rb) unsigned PC=PC+sign_extend({imm2U, imm3L})	imm2	rs2	rs1	imm3	Op=19
LUI	R1=imm11 << 5	imm11				Op=20
J	PC=PC+sign_extend(imm11)	imm11				Op=21
JAL	PC=PC+sign_extend(imm11), R7=PC+1	imm11				Op=22
JALR	PC=Ra+sign_extend(imm5), Rd=PC+1	imm5		rs1	rd	Op=23

Instruction Description:

Opcode Number	**Definition**
0 to 2	are used for R-format ALU instructions. There are 10 instructions in total.
3 to 11*****	are used for I-format instructions. There are 10 instructions in total.
12	define load word (**LW**) instruction. This instruction addresses 2-byte words in memory. The effective memory address = **Ra + sign_extend(imm5)**.
13	define store word (**SW**) instruction. This instruction addresses 2-byte words in memory. The effective memory address = **Ra + sign_extend({imm2U, imm3L})**.
14 to 19	are six branch instructions **BEQ** to **BGEU** and PC-relative addressing. If the branch is taken, then **PC = PC + sign_extend({imm2U, imm3L})**. Otherwise, **PC = PC + 1**. The conditional branch range is **[-16, +15]**.
20	define load upper immediate (**LUI**) instruction. **LUI** is used to build 16-bit constants and uses the J-type format. **LUI** places the **imm11** value in the upper 11 bits of the destination register ***R1***, filling the lowest 5 bits with zeros.
21	define the Jump (**J**) instruction. PC-relative addressing is used to compute the jump target address: **PC = PC + imm11**.
22	define the Jump-and-Link (**JAL**) instruction. PC-relative addressing is used to compute the jump target address: **PC = PC + imm11**. In addition, the **JAL** instruction saves the return address in **R7** (**R7 = PC + 1**).
23	defines the **JALR** (Jump-And-Link-Register) instruction. It saves the return address in **Rd** (**Rd = PC+1**) and does an indirect register jump with an offset (**PC = Ra + imm5**). If **Rd** is **R0** then the return address (**PC+1**) is not saved, and **JALR** becomes a Jump Register (**JR**) pseudo-instruction. To use **JALR** instruction, follow the following syntax: **JALR RD, RS1, imm5**.

***NOTE:** I-format ALU instructions (**ADDI** through **SRAI**) have identical functionality as their corresponding R-format instructions (**ADD** through **SRA**), except that the second ALU operand is an immediate constant. The **imm5** immediate value is sign extended for **ADDI**, **SLTI**, and **SLTIU**, while it is zero extended for **XORI**, **ORI**, and **ANDI**.

Memories:

Your processor will have the following features:

- Separate instruction and data memories.
- Instruction memory
    - PC register should be 16 bits.
    - can store 2^{16} instructions, where each instruction occupies two bytes.
    - there are 65536 (64K) instructions in the instruction memory.
- Data memory
    - will also be to 2^{16} words = 128 Ki Bytes.
    - is *word addressable*, since only the LW and SW instructions address memory.
    - words should be always aligned in memory.
    - ALU result represent the address for the data memory.

Addressing Modes:

Your processor will have the following addressing modes:

- PC-relative addressing mode is used for branch and jump instructions.
- For taken branches: PC = PC + sign_extend({imm2U, imm3L})
- For J and JAL: PC = PC + sign_extend(imm11)
- For JALR: PC = Ra + sign_extend(imm5)
- For LW, displacement addressing is used: Memory address = Ra + sign_extend(imm5)
- For SW, displacement addressing is used: Memory address = Ra+sign_extend ({imm2U, imm3L})

Register File:

Implement a Register file containing 8 (eight) 16-bit registers R0 to R7 with two read ports and one write port. R0 is a special register that can only be read not written (hardwired to zero). Other descriptions related to the sources and destination registers are mentioned in the **Register Use**.

Arithmetic and Logic Unit (ALU):

Develop a 16-bit ALU to handle all the required operations in this sequence:

ADD, SUB, SLT, SLTU, XOR, OR, AND, SLL, SRL, SRA

In addition, you should have special support for the LUI instruction.

Program Execution:

- The program will be loaded and will start at address 0 in the instruction memory.
- The data segment will be loaded and will start also at address 0 in the data memory.
- To terminate the execution of a program, the last instruction in the program can jump to itself indefinitely.
- To execute procedures, you can use a stack segment structure:
    a. stack segment can occupy the upper part of the data memory
    b. can grow backwards towards lower memory addresses.
    c. stack segment is implemented completely in software.
    d. you can dedicate register R6 as the stack pointer.

Develop a Single-Cycle Processor:

- Develop the control and datapath components for the single-cycle processor and verify their correctness.
- You should have enough test cases that check ALL instructions mentioned in the **Instruction Encoding part**.
- You should also write full codes (as requested in the **Testing and Verification part**) that demonstrate the execution of complete programs.
- To ensure the correctness of the design, display the contents of ALL registers from **R1** to **R7** clearly in your design's top-level (i.e. you can add labels in the Logisim). You can create output pins for each register and make their values easily observable for testing and validation.

Testing and Verification:

To show that your processor design is working probably, you need to do the following points:

1. Write a set of instructions to check the correctness of ALL instructions in the processor design. Show the correctness of all ALU R-type and I-type instructions. Show the correctness of load and store instructions. Also, show the correctness of all branch and jump instructions.
2. Write short codes and loops to check the correctness of a couple of instructions in one program.
3. Write any comparison/conditional based procedure of your choice like the following:
   If (x1>x2) {f=y^2 +2} else {f=y+1}.

Document and record all test cases and files and include them in the project document.

Project Report:

The report must contain parts highlighting the following points:

I. Cover Page

- Contains the course name, project title, your name (and id) for all team members and their section, and date.

II. Design and Implementation Part

- List the components required to build the chosen design, and the reason why each component is used. Provide drawings of all components and sub-components, and the overall datapath of the processor with full description on the figures. Don't forget to add captions for all figures and call them in the description.
- Provide a detailed description on the control logic and the control signals. Support your description with a **table** giving the control signal values for each instruction. You need to define the control signals and give all possible values.

III. Simulation and Testing Part

- Describe the test cases that you tried in the "**Testing and Verification Part**" to test your processor with the comments describing the cases, their input, and expected output. Don't forget to provide the **hex values** for all instruction used in the testing cases.
- List all instructions that were tested and worked probably. List all instructions that do not work properly.
- Provide snapshots showing test cases and their output results, each snapshot must contain the time and the date of your machine.

IV. Design Alternatives, Issues and Limitations Part

- Provide the drawings of the design alternatives for any component in the design (at least one module).
- Show the limitations and issues you have faced during the design. You need also to show the not working parts (if any).

V. Teamwork Part

- Three students can form a team at max. Write the names of all team members on the project report cover page.
- Team members need to coordinate their work among themselves, so everyone will participate in design, implementation, simulation, and testing.
- Show the work done with the percentage by each member in the team using a chart (i.e. bar, histogram, pie).

Project Deadlines:

This project consists of three main phases:

A. **Phase One (P1)**: you need to submit only the Logisim circuit file with (**.circ)** extension (named by the students' IDs, e.g. **Part1_ID1_ID2_ID3.circ**), which includes the implementation of the **Register File** and the **ALU** through ITC before end of **Friday, Nov. 29, 2024**.

B. **Phase Two (P2)**: you need to submit only the Logisim circuit file with (**.circ**) extension (named by the students' IDs, e.g. **Part2_ID1_ID2_ID3.circ**), which includes the implementation of the fully connected datapath unit without the control units (i.e. PC register, Instruction Memory, Register File, ALU, Data Memory) through ITC before end of **Sunday, Dec. 08, 2024**.

C. **Phase Three (P3)**: you need to design the control units (i.e. Main control unit and PC control unit) and do the proper connections to finalize the single-cycle processor. At this part, the single-cycle processor project should be finalized before end of **Sunday, Dec. 22, 2024**. You should have enough test cases ready to show that your processor design is fully operational. If your design is not fully functional then define which instructions/components do not work correctly to avoid losing more points while grading. By this phase, you are required to submit a **.zip** file (named by the students' IDs, e.g. **Part3_ID1_ID2_ID3.zip**) which contains the Logisim circuits for all parts (**.circ**) in a **Folder** named by **Circuits**, the test cases (i.e. codes with hex values as **.txt** files) in a **Folder** named by **Cases**, and the project report (**.pdf**) on **ITC** before the above mentioned deadline.

NOTE: One submission per team is enough and last submission will be only graded for all phases.

Grading Criteria

This is some sort of project competition. So one factor of the evaluation depends on how a project distinguishes itself. The evaluation factors that may distinguish your project:

- The more properly working features the better (i.e. a project of three well designed and implemented features is better than four poorly designed or implemented features).
- How well the parts are integrated with each other to serve the project goal.
- Modular design (Building/using sub-modules for different parts of the project).
- The level of understanding the details of implementation.
- How difficult it is to interface and use the parts (relatively).

The following table summarizes the grading criteria and submission deadlines of all components of the course project.

Project Component	**Percentage**	**Submission Deadline**
Phase One	10%	Friday, Nov. 29, 2024
Phase Two	15%	Sunday, Dec. 08, 2024
Phase Three & Project Report	50%	Sunday, Dec. 22, 2024
Discussion & Demo	25%	To be Announced Later

Generally, just by following the guidelines presented in this document, you should get a good score. However, failing to stick to these guidelines may result in a reduction proportional in magnitude to the deviation.

Good luck, *ENCS2380 Instructor*