Computer Organization and Microprocessor (ENCS2380)
Project: Single Cycle Processor Design

Objectives

- Designing a 32-bit processor with 16-bit instructions
- Using the Logisim simulator to model and test the single cycle processor
- Teamwork

Instruction Set Architecture

Here, you are required to design a RISC processor that has seven 32-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 value written to **R0** is neglected.

All instructions are **16-bit** long and aligned in memory. **PC** register (20-bit wide) contains the instruction address. All instruction addresses are even. There are **four** instruction types, **R-type**, **I-type**, **S-type**, and **J-type** as shown below:

R-type

5-bit Operation code (Op), 3-bit register numbers (x, y, and z), and 2-bit function field f

5-opcode	3-x	3-y	3-z	2-f
MSB				LSB

I-type

5-bit Op, 3-bit register numbers (x and y), and 5-bit Immediate

5-opcode	3-x	3-y	Imm5

S-type

5-bit Op, 3-bit register numbers (y and z), and 5-bit Immediate split into (Imm3 and Imm2)

5-opcode	Imm3	3-y	3-z	Imm2

J-type

5-bit Op and 11-bit Immediate

5-opcode	Imm11

Register Use:

Register	Number	Definition	Function
Register x (**Rx**)	Destination register	**Rx** is the name and value of register **x**	Always written and never read
Register y (**Ry**)	First source register	**Ry** is the name and value of register **y**	Always read and never written
Register z (**Rz**)	Second source register	**Rz** is the name and value of register **z**	Always read and never written

Instruction Encoding:

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

Instruction	Definitions	Encoding				
AND	Rx = Ry & Rz	Op = 0	x	y	z	f = 0
CAND	Rx = ~Ry & Rz	Op = 0	x	y	z	f = 1
OR	Rx = Ry | Rz	Op = 0	x	y	z	f = 2
XOR	Rx = Ry ^ Rz	Op = 0	x	y	z	f = 3
ADD	Rx = Ry + Rz	Op = 1	x	y	z	f = 0
NADD	Rx = –Ry + Rz	Op = 1	x	y	z	f = 1
SEQ	Rx = (Ry == Rz) (result is 0 or 1)	Op = 1	x	y	z	f = 2
SLT	Rx = (Ry < Rz) (result is 0 or 1)	Op = 1	x	y	z	f = 3
ANDI	Rx = Ry & zero_extend(Imm5)	Op = 2	x	y	Imm5	
CANDI	Rx = ~Ry & zero_extend(Imm5)	Op = 3	x	y	Imm5	
ORI	Rx = Ry | zero_extend(Imm5)	Op = 4	x	y	Imm5	
XORI	Rx = Ry ^ zero_extend(Imm5)	Op = 5	x	y	Imm5	
ADDI	Rx = Ry + sign_extend(Imm5)	Op = 6	x	y	Imm5	
NADDI	Rx = –Ry + sign_extend(Imm5)	Op = 7	x	y	Imm5	
SEQI	Rx = (Ry == sign_extend(Imm5)) (result is 0 or 1)	Op = 8	x	y	Imm5	
SLTI	Rx = (Ry < sign_extend(Imm5)) (result is 0 or 1)	Op = 9	x	y	Imm5	
SLL	Rx = Shift_Left_Logical(Ry, Imm5)	Op = 10	x	y	Imm5	
SRL	Rx = Shift_Right_Logical(Ry, Imm5)	Op = 11	x	y	Imm5	
SRA	Rx = Shift_Right_Arithmetic(Ry, Imm5)	Op = 12	x	y	Imm5	
ROR	Rx = Rotate_Right(Ry, Imm5)	Op = 13	x	y	Imm5	
BEQ	Branch if (Rx == Ry) (PC += sign_extend(Imm5<<1))	Op = 14	x	y	Imm5	
BNE	Branch if (Rx != Ry) (PC += sign_extend(Imm5<<1))	Op = 15	x	y	Imm5	
BLT	Branch if (Rx < Ry) (PC += sign_extend(Imm5<<1))	Op = 16	x	y	Imm5	
BGE	Branch if (Rx >= Ry) (PC += sign_extend(Imm5<<1))	Op = 17	x	y	Imm5	
LW	Rx ← 4byte MEM[Ry + sign_extend(Imm5)]	Op = 18	x	y	Imm5	
SW	MEM[Ry + sign_extend({Imm3,Imm2})] ← 4byte Rz	Op = 19	Imm3	y	z	Imm2
J	PC = PC + sign_extend(Imm11<<1)	Op = 20	Imm11			
JAL	R7 = PC + 2; PC=PC+ sign_extend(Imm11<<1)	Op = 21	Imm11			

Instruction Description:

Opcodes 0 and 1: **R-type** ALU instructions and there are in total 8 instructions.

NOTES:

SEQ: set Rx to 1 if Ry equal Rz, otherwise set Rx to 0.

SLT: set Rx to 1 if Ry less than Rz, otherwise set Rx to 0.

Opcodes 2 to 13: **I-type** ALU instructions. The immediate value considers as the second ALU operand and register **x** is the destination operand.

NOTES:

I-type ALU instructions from **ANDI** to **SLTI**: have same functionality of **R-type** instructions from **AND** to **SLT**, the destination register is **Rx** (not **Rz**) and the second operand is an immediate.

I-type encodes a 5-bit signed immediate/constant (**Imm5**) with range -16 to +15.

Opcodes 10 to 13: four shift and rotate instructions (**SLL** to **ROR**). Shift or rotate value is the **Imm5** with values 0 to 31.

Opcodes 14 to 17: four branch instructions (**BEQ** to **BGE**) and PC-relative addressing mode. It updates the PC value as follows:

Branch is taken then **PC = PC + sign_extend(Imm5<<1)**. Else, **PC = PC + 2**.

NOTE:

Imm5<<1: shift the Imm5 by one bit to the left.

Opcode 18 define load word (**LW**) instruction. This instruction addresses 4-bytes in memory. The effective memory address = Ry + sign_extend(imm5).

Opcode 19 define store word (**SW**) instruction. This instruction addresses 4-bytes in memory. The effective memory address = Ry + sign_extend({Imm3, Imm2}).

NOTE:

Loading and storing one byte or a half word is not defined here to simplify the design.

Opcodes 20 and 21: Jump (**J**) and Jump-and-Link (**JAL**) instructions. These instructions are PC-relative addressing and to compute the jump target address:

PC = PC + sign_extend(Imm11<<1).

Also, **JAL** instruction stores the return address in **R7** (**R7 = PC + 2**).

Programming Notes:

NADD	**R1, R2, R3**	computes R1 = R3 – R2	(Rx = R1, Ry = R2, and Rz = R3)
SUB	**R1, R2, R3**	is a pseudo-instruction ➔	**NADD R1, R3, R2**
ADDI	**R1, R2, –1**	computes R1 = R2 – 1	(Rx = R1, Ry = R2, and Imm5 = –1)
NADDI	**R1, R2, 1**	computes R1 = 1 – R2	(Rx = R1, Ry = R2, and Imm5 = 1)
LI	**R1, 5**	is a pseudo-instruction ➔	**ORI R1, R0, 5**

PC Register and Memories

Your processor will have separate instruction and data memories. The PC register should be 20 bits. The instruction memory can store 2^{20} instructions, where all instruction addresses are multiple of 2 bytes (i.e. where each instruction occupies two bytes).

The data memory will also be to 2^{20} words = 4 Mi Bytes. The data memory will be also restricted to 2^{20} bytes. Although the architecture is 32 bits, the size of the instruction and data memories will be restricted to 2^{20} bytes. This is because the **Logisim** tool supports only small memories. The ALU result represent the address for the data memory.

Addressing Modes:

PC-relative addressing mode is used by all branch and jump instructions.

For taken branches: **PC = PC + sign_extend(Imm5<<1)**

For jumps (**J** and **JAL**): **PC = PC + sign_extend(Imm11<<1)**, where **Imm11** is only 11 bits.

For **LW**, displacement addressing is used: **Address = Ry + sign_extend(Imm5)**

For **SW**, displacement addressing is used: **Address = Ry + sign_extend(Imm3, Imm2)**

Register File:

Develop a Register file comprising seven 32-bit registers labeled from **R1** to **R7**, serving one write port and two read ports. **R0** is fixed at 0 and remains immutable. **R0** always reads as 0 and cannot undergo any write operations.

Arithmetic and Logic Unit (ALU):

Develop a 32-bit ALU to handle all the required operations:
XOR, AND, OR, CAND, ADD, SEQ, NADD, SLT, SRA, SRL, SLL, and ROR.

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 because there is no underlying operating system to terminate the program.

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 set. 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). 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 LW and SW 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 your 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 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 work 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 (if any).
- 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, students could be from any section. 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 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**: you need to submit only the Logisim circuit file with (**.circ)** extension (named by the students' IDs, e.g. ID1_ID2_ID3.circ), which includes the implementation of the Register File and the ALU through ITC before end of Friday, May 24, 2024.

B. **Phase Two**: you need to submit only the Logisim circuit file with (**.circ**) extension (named by the students' IDs, e.g. 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 Friday, May 31, 2024.

C. **Phase Three**: 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 phase the single-cycle processor project should be finalized before end of Saturday, Jun. 08, 2024. It should be fully working and will be assessed by your instructor during this week. 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. ID1_ID2_ID3.zip) which contains the Logisim circuits for all parts (**.circ**), the test cases (i.e. codes with hex values as **.txt** files), and the project report (**.pdf**) on **ITC** before the above mentioned deadline.

NOTE: One submission per team is enough and the last submission will be only considered 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	15%	Friday, May 24, 2024
Phase Two	15%	Friday, May 31, 2024
Phase Three & Project Report	45%	Saturday, Jun. 08, 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 Instructors*