Department of Electrical and Computer Engineering
First Semester, 2022/2023

Project No. 2
ENCS4370| Computer Architecture
Due Friday February 10, 2023

---

1. Objectives:

1. Designing and testing a simple Multi-Cycle RISC processor using HDL language (Verilog or VHDL)

2. Instruction Set Architecture

In this project, you are required to design a simple Multi-Cycle RISC processor with the following specifications:

1. The instruction size is 24 bits (word size 24-bit)
2. All instructions are conditionally executed.
3. Eight 24-bit general-purpose registers: R0 through R7.
4. R0 is hardwired to zero and cannot be written. Any instruction attempts to write R0 is discarded.
5. The program counter (PC) is a 24-bit special-purpose register (24-bit memory address space).
6. 8-bit status register. At this project, consider only the least significant bit in this status register, namely, the **`zero` (Z)** flag bit. This bit is set if the last ALU operation result is zero, and it is cleared otherwise.
7. The programmer can determine whether the ALU instruction should update the flag bits or not by appending the suffix SF to the instruction mnemonic, e.g., **`ADD`** just adds, but **`ADDSF`** adds and updates the flag bits (**`Z`** flag). The ALU instruction binary format contains a 1-bit field to encode this.
8. Three instruction types (R-type, I-type, and J-type).
9. Five addressing modes as in MIPS32 ISA.

3. Instruction Types (Formats)

As mentioned above, this ISA has three instruction formats, namely, R-type, I-type, and J-type. These three types have the following two common fields:

a. Opcode: a 5-bit field to tell the processor which operation to perform

b. Condition: a 2-bit field that enables every instruction to be conditionally executed. For this project, we will consider two conditions, i.e., equal and not equal as shown in the following table:

Condition Field Value	Effect
00	Always execute the current instruction. In assembly, there is no change on the instruction mnemonic. For example, **`ADD R1, R2, R3`** is always executed.
01	Execute if equal, i.e., execute the current instruction if the previous ALU instruction resulted in a zero result, in other words, if the zero-flag bit is set. Otherwise, the current instruction can be treated as a NOP (no operation). This can be reflected in assembly by appending **`EQ`** suffix to the instruction mnemonic, such as, **`ADDEQ`**, **`ANDEQ`**, **`SUBEQ`** etc. For example, **`ADDEQ R1, R2, R3`** is executed if and only if the **`zero`** flag bit has the value of **`1`**
10	Execute if not equal, i.e., execute the current instruction if the previous ALU instruction resulted in a nonzero result, in other words, if the zero-flag bit is cleared. Otherwise, the current instruction can be treated as a NOP (no operation). This can be reflected in assembly by appending **`NE`** suffix to the instruction mnemonic, such as, **`ADDNE`**, **`ANDNE`**, **`SUBNE`** etc. For example, **`ADDNE R1, R2, R3`** is executed if and only if the **`zero`** flag bit has the value of **`0`**
11	Unused

R-type format

- 2-bit condition (Cond)
- 5-bit opcode (Op)
- 1-bit whether to set the flag bits or not (**`SF`**). For this project and for simplicity, this bit is used only with the subtraction instructions only (**`SUBSF`**, **`SUBISF`**). For other instructions, this bit is always **`0`**. For example, **`SUBSF, R1, R2, R3`** will perform the following
  - Reg [R1] = Reg[R2] – Reg[R3]
  - Zero-Flag = Reg[R2] == Reg[R3]
- 3-bit destination register (Rd)
- 3-bit source 1 register (Rs)
- 3-bit source 2 register (Rt)
- 7-bit unused (for future use)

$Cond^2$	Op^5	SF^1	Rd^3	Rs^3	Rt^3	$Unused^7$

I-type format

In addition to the common fields with R-type, it has:

- 3-bit destination register (Rt)
- 3-bit source 1 register (Rs)

- 10-bit signed immediate value in two's complement representation (2^{nd} operand)

$Cond^2$	Op^5	SF^1	Rt^3	Rs^3	$Immediate^{10}$

J-type format

In addition to the condition and opcode fields, the J-type instruction contains a 17-bit signed immediate constant in two's complement representation, i.e., the jump offset

$Cond^2$	Op^5	$Immediate^{17}$

4. Instructions' Encoding

The following table shows the different instructions you are required to implement. It shows their type, format, and their meaning in RTN. Note that for each instruction listed below, there are two additional instruction mnemonics. For example, the addition operation ADD in the R-Type format has other two mnemonics: ADDEQ and ADDNE. The three mnemonics have the same Op-Code but differ in the condition field as following:

ADD: Op = 00011, Cond = 00, SF = 0
ADDEQ: p = 00011, Cond = 01, SF = x
ADDNE: Op = 00011, Cond = 10, SF = x

ADDI: Op = 01000, Cond = 00, SF = 0
ADDIEQ: Op = 01000, Cond = 01, SF = x
ADDINE: Op = 01000, Cond = 10, SF = x

Moreover, only for SUB operations, there is another instruction mnemonic as following

SUBSF: Op = 00011, Cond = 00, SF = 1
SUBISF: Op = 01000, Cond = 00, SF = 1

No.	Instr	Meaning	Encoding			
R-Type Instructions						
0	AND	Reg(Rd) = Reg(Rs) & Reg(Rt)	Op = 00000	Rs	Rt	Rd
1	CAS	Reg(Rd) = Max[Reg(Rs) , Reg(Rt)]	Op = 00001	Rs	Rt	Rd
2	Lws	Reg(Rd) = Mem[Reg(Rs) + Reg(Rt)]	Op = 00010	Rs	Rt	Rd
3	ADD	Reg(Rd) = Reg(Rs) + Reg(Rt)	Op = 00011	Rs	Rt	Rd
4	SUB	Reg(Rd) = Reg(Rs) – Reg(Rt)	Op = 00100	Rs	Rt	Rd
5	CMP	zero-flag = Reg(Rs) < Reg(Rt)	Op = 00101	Rs	Rt	0000
6	JR	PC = Reg(Rs)	Op = 00110	Rs	0000	0000
I-Type Instructions						
7	ANDI	Reg(Rt) = Reg(Rs) & $Immediate^{10}$	Op = 00111	Rs	Rt	$Immediate^{10}$
8	ADDI	Reg(Rt) = Reg(Rs) + $Immediate^{10}$	Op = 01000	Rs	Rt	$Immediate^{10}$

9	Lw	Reg(Rt) = Mem(Reg(Rs) + Imm^{10})	Op = 01001	Rs	Rt	$Immediate^{10}$
10	Sw	Mem(Reg(Rs) + Imm^{10}) = Reg(Rt)	Op = 01010	Rs	Rt	$Immediate^{10}$
11	BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 01011	Rs	Rt	$Immediate^{10}$
J-Type Instructions						
12	J	PC = PC + $Immediate^{17}$	Op = 01100	$Immediate^{17}$		
13	JAL	R7 = PC + 1, PC = PC + $Immediate^{17}$	Op = 01101	$Immediate^{17}$		
14	LUI	R1 = $Immediate^{17}$ << 4	Op = 01110	$Immediate^{17}$		

The Load Upper Immediate (LUI) is of the J-type to have a 17-bit immediate constant loaded into the upper 17 bits of register R1. The LUI can be combined with ORI (or ADDI) to load any 24-bit constant into a register. Although the instruction set is reduced, it is still rich enough to write useful programs. We can have procedure calls and returns using the JAL and JR instructions.

5. Multi-Cycle

Design and implement a Multi-Cycle -Datapath and its control logic. A five-stage Datapath should be constructed similar to the one presented in the class lectures. Add a register between stages. Design the control logic using state machine approach. For mor information on Multi-Cycle Datapath and its control unit, please see the attached documents.

6. Testing

To test the implementation, write a sample code to test all the instructions that you have implemented. Note that, the program will be loaded and will start at address 0 in the instruction memory. The data segment will be loaded and will start at address 0 in the data memory, as well.

7. Project Report

The report document must contain sections highlighting the following:

1 – Design and Implementation

1. Specify clearly the design giving detailed description of the data path, its components, control, and the implementation details (highlighting the design choices you made and why, and any notable features that your processor might have.)
2. Provide drawings of the component circuits and the overall data path.
3. Provide a complete description of the control logic and the control signals. Provide a table giving the

control signal values for each instruction. Provide the logic equations for each control signal.

4. Provide a list of sources for any parts of your design that are not entirely yours (if any).
5. Carry out the design and implementation with the following aspects in mind:
   - Correctness of the individual components
   - Correctness of the overall design when wiring the components together
   - Completeness: all instructions were implemented properly.

2 – Simulation and Testing

1. Carry out the simulation of the processor developed using Verilog or VHDL.
2. Describe the test programs that you used to test your design with enough comments describing the program, its inputs, and its expected output. List all the instructions that were tested and work correctly. List all the instructions that do not run properly.
3. Also, provide snapshots of the simulator window with your test program loaded and showing the simulation output results.

3 – Teamwork

1. Work in groups of two or three students.
2. Group members are required to coordinate the work equally among themselves so that everyone is involved in all the following activities:
   - Design and Implementation
   - Simulation and Testing
3. Clearly show the work done by each group member.

8. Submission Guidelines

Attach one zip file containing all the design circuits, the programs source code and binary instruction files that you have used to test your design, their test data, as well as the report document to ritaj.

9. Grading policy

The grade will be divided according to the following components:

1. Correctness: whether your implementation is working
2. Completeness and testing: whether all instructions and cases have been implemented, handled, and tested properly
3. Participation and contribution to the project
4. Report document
5. Discussion

Note that you can implement the above-mentioned architecture using single cycle approach but you will get at max 80% from your achieved grade (at max you will get 12 from 15)

Grading policy:

Item	Weight
Designing, implementing, and testing Control Unit (Building state diagram and Deriving expression for each signal)	20
Designing Final Data path and drawing final schematic that support all instructions	40
Implementing Final Data Path	30
Testing Final Data Path	20
Report	20
Discussion	20
Total	**120**