Due Date: 01/01/2024

CSE3015 / CSE3215 DIGITAL LOGIC DESIGN TERM PROJECT

General Info

You are expected to design and implement a processor which supports instruction set: (*ADD, AND, NAND, NOR, ADDI, ANDI, LD, ST, CMP, JUMP, JE, JA, JB, JAE, JBE*). Processor will have **10 bits address** width and **18 bits data** width. Processor will have 5 parts as follows. **Register File** will hold register values and signal to write into any register. There will be **16 registers** in processor. **Instruction Memory** will be a *read-only memory* and instructions will be stored in this component. If the current instruction is not one of the *JRA*, *JUMP, JE, JA, JB, JAE, JBE*; the next instruction will be fetched and executed consecutively from this memory. **Data Memory** will be *read-write memory* which will store data. Program will be able to read data from data memory, and also store data to this memory. **Data Memory** will have **10 bits address** width, and **18 bits data** width. **Control Unit** will produce proper signals to all datapath components. For example, if the instruction is *ST*, control unit should produce memWrite signal which will allow Data Memory component to write data value on its data input to the address on its address input. **Arithmetic Logic Unit (ALU)** will compute arithmetic operations *ADD, AND, NAND, NOR, ADDI, ANDI*. Operands will be fetched from *register+register* or *register+ immediate value*. Result will be stored to the Register File. Control unit should produce proper signals to ALU according to instruction opcode (*Every instruction should have distinct operational code*). Detailed information about instructions is given below.

- *ADD, AND, NAND, NOR will have same form.*

ADD instruction will add two register, and store result into another register. Structure of instruction is:

ADD DST, SRC1, SRC2

where SRC1 and SRC2 are source registers, and DST is destination register for the operation.

- *ADDI, ANDI* will have same form.

ADDI instruction will add a register value and immediate value, and store the result into another register. Form of instruction is:

ADDI DST, SRC1, IMM

where SRC1 is a register, DST is destination register and IMM is immediate value. IMM size will be max available size on your processors design.

- *JUMP* instruction will set the Program Counter (PC will hold current instruction's address) to the given value in the instruction.

JUMP ADDR

where ADDR will be in PC relative mode. ADDR will be offset and it can be negative. For example:

JUMP 3

instruction will set PC to: next instructions address + 3(go forward 3 instruction)

JUMP -5

instruction will set PC to: next instructions address – 5 (go back 5 instruction)

- *LD* instruction will load a value from Data Memory to any register.

LD DST, ADDR

where DST is a register to load and ADDR is an address in max available bit size. Upper bits of ADDR will be zero extended.

- *ST* instruction will store value from a register to Data Memory.

ST SRC, ADDR

where SRC is a register to fetch data and ADDR is a Data Memory address to store content of the register. Upper bits of ADDR will be zero extended.

- Compare instruction will compare two operands, then will update the flag values of zero flag **ZF** and carry flag **CF**. Following conditional jump instruction will set the destination address if the condition holds.

CMP OP1, OP2

will compare registers OP1 and OP2 if they are equal, flag values will be **ZF=1**, **CF=0**, if the first operands value is greater flag values will be **ZF=0**, **CF=0**, if the first operands value is less than the second operand flag values will be **ZF=0, CF=1**.

As in the **JUMP** instruction, but conditional with the flag values, **JE, JA, JB, JAE, JBE** jumps to a new destination address.

JE ADDR

If flag values are (**ZF=1, CF=0**), PC will be set to ADDR(PC-relative).

JA ADDR

if flag values are (**ZF=0, CF=0**), PC will be set to ADDR(PC-relative).

JB ADDR

if flag values are **(ZF=0, CF=1**), PC will be set to ADDR(PC-relative).

JAE ADDR

if flag values are (**CF=0**), PC will be set to ADDR(PC-relative).

JBE ADDR

if flag values are (**CF=1 or ZF=1**), PC will be set to ADDR(PC-relative).

Design Guide

Since you already have your instructions and instruction length (18 bits), your design process includes; instruction set architecture and control unit design. For instruction set architecture you should decide on fields of your instruction like, what will be size of opcode field? How many bits to reserve for register addressing in instruction? What is the maximum possible size for immediate part? Control unit design will include proper signal generation for all datapath. First, you must define a Finite State Machine for your processor (*Micro-programmed control unit is not allowed!*). For every instruction you should decide, how many states an instruction will need to execute? What will be the control signals for each state of an instruction?

For example, LD instruction will do following operations consecutively:

1. Read PC value
2. Fetch instruction from Instruction Memory with address given in PC
3. Read Data Memory with given address in instruction
4. Write content from Data Memory to destination register.

LD instruction will require 4 states to complete. At each state some signals should control components to do right operations. For example, at state 1, **PCread signal**, which will allow PC to put its content on its output, should be activated. Remember that PC is an 18-bit register which holds the current instructions address. At state 2, instruction fetch, instruction will be fetched from Instruction Memory. Therefore, **instRead signal** will be activated. At state 3, extended address will be put on address input of Data Memory, and **dataMemRead signal** will be activated. At state 4, we will write output from Data Memory to the register given in instruction. So, **regWrite signal** will be activated. Remember that, your full Datapath will contain additional combinatorial components like multiplexer, adder, ALU. Control signals have to be designed also for these parts of processor. For example, if current instruction is not setting PC value (not one of *JUMP, BR*), your next PC value will be PC+1. Since there is two possible next value for PC, there should be a multiplexer at input of PC register with two inputs: PC+1 and PC relative calculated address from instructions JUMP, BR. You should produce a signal for this multiplexer **Pcselect signal**. Your design must include following signals also. (**Remember that there can be additional signals and these signals are dependent on your design! - Mux signals or other additional signals**).

- **ALUcontrol** signal should select operation on the Arithmetic Logic Unit (ALU). For example, there are 4 arithmetic or logic operation on ALU (*AND, OR, XOR, ADD*). **ALUcontrol** must be at least 2 bits and for every operation 2-bit value has to be assigned.
- **MemWrite** signal must be 1 for the final state of *ST* instruction.
- **PCwrite** at the final state of *JUMP* and *Branch* instructions, PC will be modified. PCwrite signal must be 1.

Assembly Language – Due To: 01/12/2023

You should first design your instruction set architecture(ISA). Draw your ISA to the paper by hand or by any schematic tool. When the details are clear about your processor, you can write an assembler to produce machine code input for your processor. Assembler will convert given mnemonics to the binary codes. You can use any programming language. In assembly language refer registers as *R0, R1... R15*. Some example instructions:

ADD R5, R0, R2	*// which will do R5=R0+R2*
ADDI R3, R1,12	*// which will do R3=R1+Extend[12]*
LD R5,12	*// which will do R5=DataMemory[Extend[12]]*

Assembler input will be a code sequence of assembly language given in above instruction set. Conversion of mnemonics to `.hex` output will depend on your ISA. Since our processor is 18 bits, for each instruction assembler should output 5-digit hexadecimal number like following:

04000

34202

28440

18683

0C60A

386EC

12801

22063

3A000

This output file will be input to your Logism design. Logisim memory file with above machine code looks like:

Due Date: 01/01/2024

v2.0 raw

04000 34202 28440 18683 0c60a 386ec

12801 22063 3a000

Logisim Component Design - Due To: 08/12/2023

You are expected to design your processor on Logisim schematic design software. In this phase of the project, you will design your components on the datapath except Control Unit. You have to make sure all components working properly by unit testing.

Logisim Design with Control Unit - Due To: 22/12/2023

Last phase of the project includes whole datapath with Control Unit. Datapath signals on datapath must be generated in Control Unit using a Finite State Machine. Since there will be different designs, you are responsible for preparing test inputs for your own processor. You are expected to write an assembly code with all instructions defined above. Then, your assembler will convert assembly code to machine code which will be input for your logisim design. You can load this machine code to the Instruction Memory by right clicking on Memory component on Logisim software.

Verilog Design - Due To: 01/01/2024

You are required to design your processor on Verilog HDL as final part. Each component in your design should be defined as a module in Verilog HDL. Your assemblers machine code output will be input for your Verilog design. You are responsible for generating test instructions, testing and debugging your design on ModelSim software.

Submission Details

You can submit the project as group of **at most 4** students. You should submit your all work under file name **`StudentID1_StudentID2_StudentID3_StudentID4.zip` to the Canvas system before the midnight of 01/01/2024** including **report** that explains every step of your implementation in details. **There will be a project quiz after the final submission deadline.**

Notes: If you do not design a ISA (first part), you won't have any design to implement on Logisim. Project Report is limited to 5 pages, must include your implementation details, do not copy and paste project document sentences. Explain your implementation with your own sentences. You cannot implement control unit as a micro-programmed control unit, you must implement it as Finite State Machine.