EEL 3701C: Digital Logic & Computer System

LAB 05: Control Path and Datapath

(25 Points)

Objectives:

The main objectives of this lab are:

- Understand the components of the Datapath
- Designing a simple ALU
- Specifying Controller
- Finally implement and simulate the behavior of simplified processor

Introduction:

In this lab, you will design a small processor which has an ALU. The ALU can be configured to several operations, such as 2's complement, bitwise OR, and add functions. Also, operands of the ALU have Datapath that is controlled by a controller. The Datapath ensures different possible routing of the input and output of the ALU. Alternately, the controller selects the required routing and changes the routing when required. In addition, the controller sets the operation in the ALU by extracting the received instruction. These modules provide a small processor architecture. The processor's advantage is that we can change the ALU function or select the operand we require by giving specific instructions at the processor's input.

Prelab Questions:

1. A left shift is equivalent to which mathematical operation? (Add, subtract, multiply, divide, etc.)
2. A right shift is equivalent to which mathematical operation?
3. What is the difference between a logical right shift and an arithmetic right shift? When must an arithmetic right shift be used to preserve algebraic correctness?
4. How many possible distinct operations can a CPU with a single 4-bit instruction field and two 2-bit register fields perform?
   For example, a single operation could be **ADD R1, R2**. A second operation could be **ADD R2, R1**. A third operation could be **ADD R1, R1**. A fourth operation **SUB R3, R2**. Based on the given parameters, how many possible operations are there?

There are two prelab problems. Submit your prelab report before your lab session begins.

There are no video demos for this lab.

Pre-Lab Problem 1: ALU Design

Figure 1 illustrates the architecture of a 4-bit ALU that must be designed for a simplify processor called the G-CPU processor.

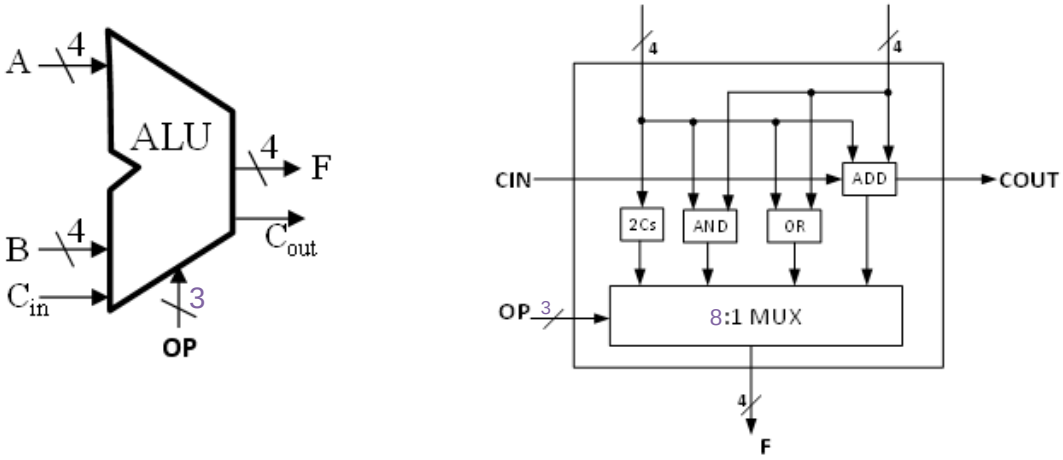

Figure 1: Schematic ALU representation

(Note: only 4/8 functions are shown in the right block diagram)

OP	Operation	Description	Meaning
000	2CMP	2's Complement	F = 2's Complement of A
001	AND	Logical AND	F = A AND B
010	OR	Logical OR	F = A OR B
011	ADD	Binary Addition	F = A + B + C_{in}, C_{out} = Carry of sum
100	OUTA	Output A	F = A
101	OUTB	Output B	F = B
110	SLL	Shift Left Logical	F = A << 1
111	SRL	Shift Right Logical	F = A >> 1

Table 1: ALU Operations. Note: Let C_{out} be a don't care OP != ADD.

1. Provide the VHDL implementation of the ALU according to the block diagram. Recall that multiplexers are case differentiations that can be represented with "if then/else," "case/when," or "when/else" statements. Processes are permissible.
2. Create an **annotated simulation** to prove the correctness of your design.

To implement the shift operator, use the concatenation operator **&**. For example, to implement a left-logical shift:

```
Y <= X(2 downto 0) & '0';
```

For your deliverable, provide the VHDL implementation and annotated simulation of the above.

Pre-Lab Problem 2: G-CPU Datapath Design

In the second part of this lab, the ALU designed in Problem 1 will be used to build a datapath. The datapath is connected to an input bus to read externally inputted operands, and an output bus to send results out.

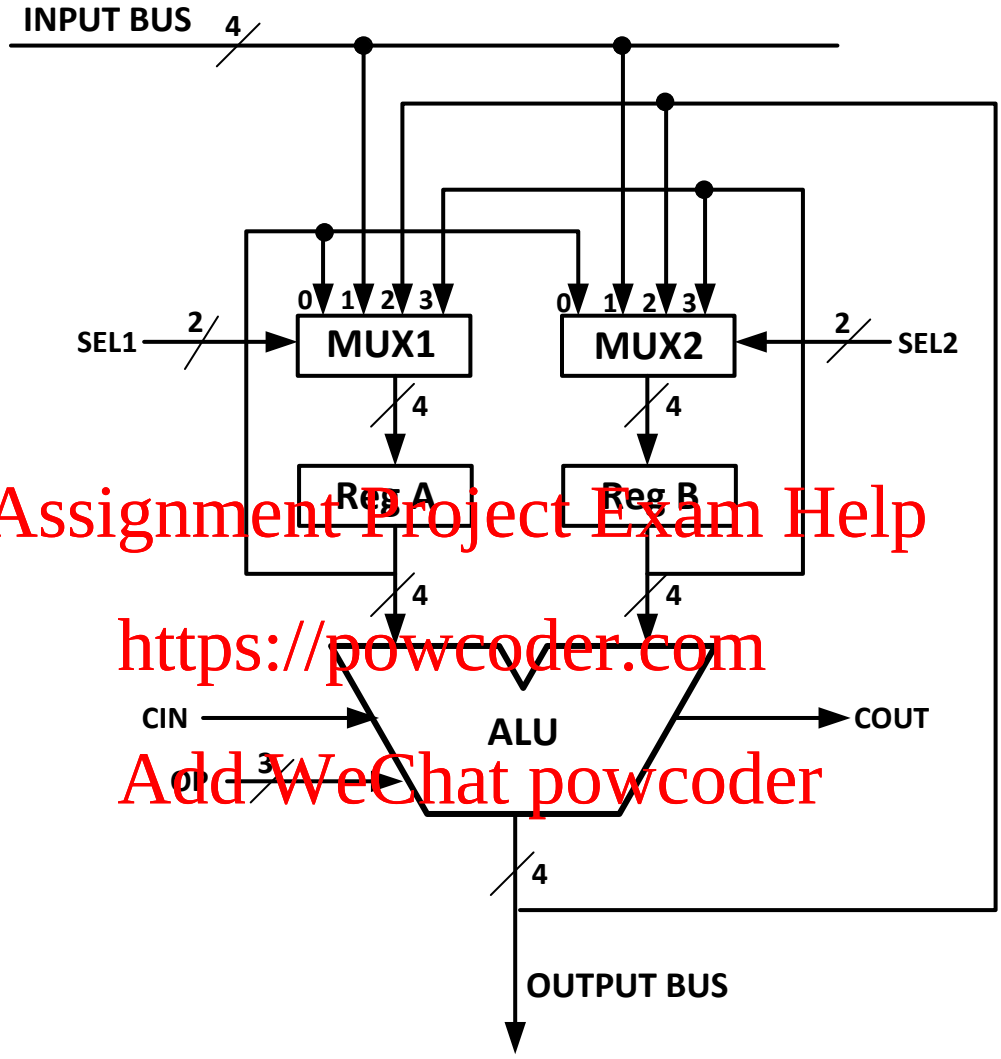

Figure 2: G-CPU Datapath

1. Provide the VHDL implementation of the data path design. Analyze Figure 2 carefully and complete the given template of *GCPU_Datapath.vhd*.
2. Create an **annotated simulation** to prove the correctness of your design.

For your deliverable, provide the VHDL implementation and annotated simulation of the above.

There is no prelab demo. Just simulations.

In-Lab Problem 3: G-CPU Control Path

The purpose of the controller is to take an instruction as an input and guide the datapath to execute said instruction. The reading of the operands from INPUT and the storage of the results into registers must all be controlled by your controller.

As shown on Figure 3, your **controller** should have **three** inputs: START, C_{OUT}, and the 5-bit machine code. (Remember: clocks and resets are implied). Note that C_{OUT} is an input here because it allows for us to put two GCPUs in a row and connect the C_{OUT} of the ALU to the C_{OUT} of the controller.

It should have **five** outputs: SEL1, SEL2, CIN, OP, and DONE.

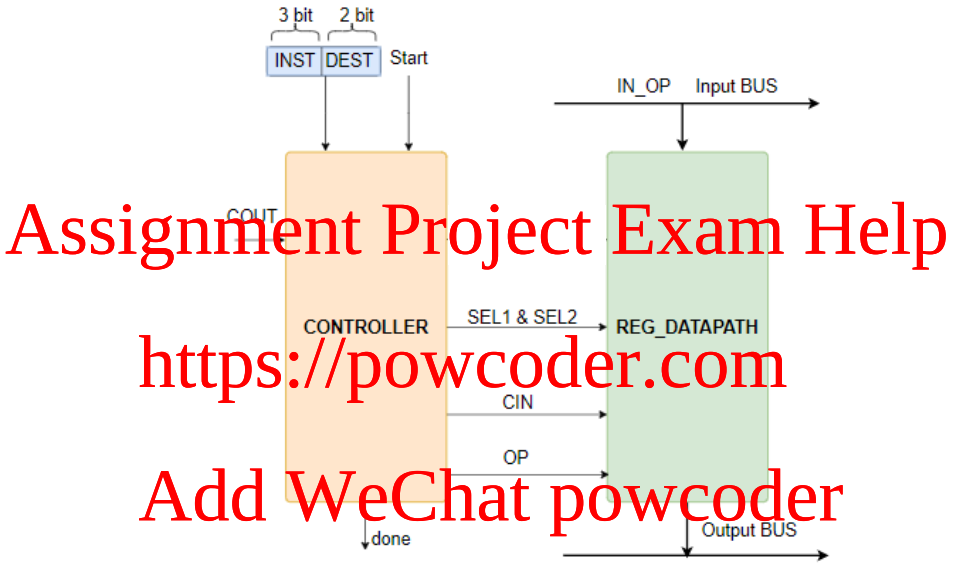

Figure 3: Control and Datapath for the G-CPU. For this part, you will create the **Controller**.

INST	Opcode	Meaning
000	LOAD	DEST = IN_OP
001	TAB	If DEST == RegA, RegA = RegB Else if DEST == RegB, RegB = RegA
010	AND	DEST = RegA AND RegB
011	OR	DEST = RegA OR RegB
100	ADD	DEST = RegA + RegB
101	SUB[1]	DEST = RegB – RegA; A is not preserved
110	SRL	DEST = Logical Right Shift(A)
111	OUT	Output Bus = DEST

Table 3: Possible INST values and their meanings

DEST	Register
00	RegA
01	RegB
1-	Nothing

Table 4: Interpretation of DEST.

[1] This instruction is **destructive**. This means that data in the specified register is permanently lost after execution of said instruction.

Your controller should take the form of a state machine. Below is only a small portion of an Algorithmic State Machine diagram you must complete. An ASM is like a mix between a Moore and a Mealy machine, in which outputs can depend on either state, inputs, or both.

Some instructions may require multiple cycles to execute.

Your controller should not begin execution of a malformed instruction; an instruction is malformed if **DEST** is pointing to **Nothing**. But you can assume that an instruction will not become malformed after execution begins (i.e., you leave the **Decode** state).

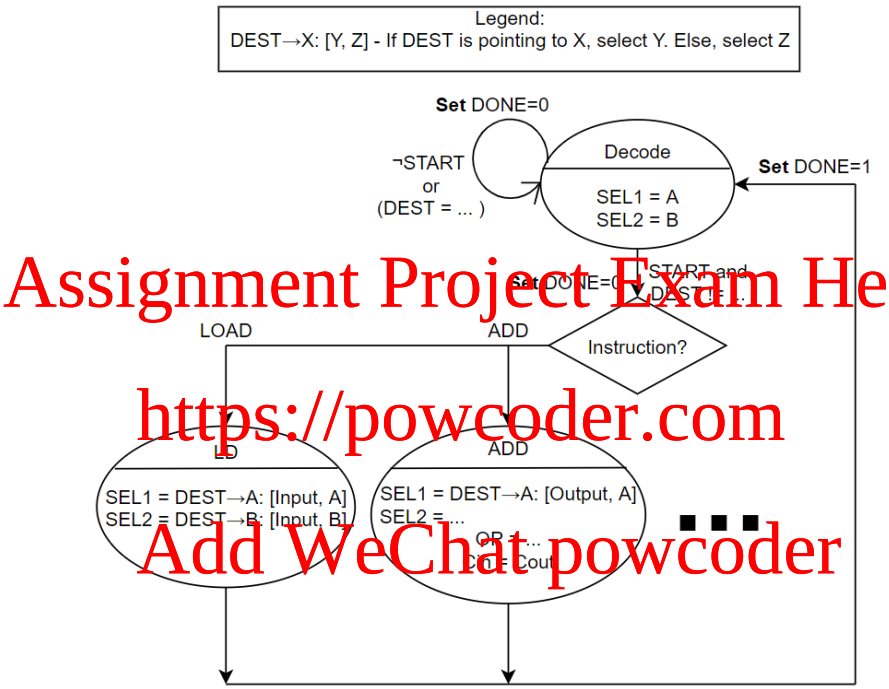

Figure 4 - ASM for the Control Path Note updates to the legend, and transitions near Decode
Hint: For the OUT instruction, you should use the syntax from the legend for the OP output

```
constant SEL_A   : std_logic_vector(1 downto 0) := "00";
constant SEL_B   : std_logic_vector(1 downto 0) := "11";
constant SEL_IN  : std_logic_vector(1 downto 0) := "01";
constant SEL_OUT : std_logic_vector(1 downto 0) := "10";
```

VHDL tip: You can define **constants** to help you be more expressive with your VHDL. These go in between the **architecture** and **begin** keywords. For example, it can be used like: **SEL1 <= SEL_B;** This is 100% not required, but it may be helpful to keep track of the select/destination values.

1. Finish the preceding ASM diagram for all defined instructions. Optionally, the **draw.io** file used to create the above is provided if you wish to use it. Use https://draw.io to edit it.
2. Provide the VHDL implementation of the controller design. If you carefully follow the state diagram you created, you will have no problems. You may use the provided template of *GCPU_ControlPath.vhd*. **(3 pts)**
3. Create an **annotated simulation** to prove the correctness of your design.

If you encounter bugs, it may help to comment one process out and manually test the other process using simulations. It may help to create a "debug" output that tells you the current state by translating it to a binary number (for example, assigning the Decode state to 0000 and the Add state to 0001, etc).

For your deliverable, provide the ASM, VHDL implementation and annotated simulation of the above.

In-Lab Problem 4: G-CPU Design

You have implemented the different modules in the GCPU. Now, you need to write a top-level file to complete the implementation. Carefully observe Figure 3 and complete a new VHDL file **Lab4.vhd**. This should only be an instantiation of the two modules you created before, the datapath and the control path.

You should have two inputs: a 9-bit **machine code** and START.

You should have two outputs: Output (from the datapath) and DONE.

(Note, **output** and **input** are reserved keywords in VHDL and should NOT be used as a name)

The format of the 9-bit machine code looks like this:

INST [Bits 8-6]	IN_OP [Bits 5-2]	DEST [Bits 1-0]
What the G-CPU should do	Operand	Destination register
3 bits	4 bits	2 bits
Refer to Table 3	IN_OP should be placed on INPUT	Refer to Table 4

Program: X = (A + 8) // 4 - B; (R 3

1. Create the VHDL implementation of **Lab4.vhd**. No template is provided for this.
2. Create an **annotated simulation** to create the program described above. Use A = 5, B = 7. Show X by using the OUT instruction once the calculation is completed.
3. Answer: Give one example of a combination of inputs A and B that would cause an unexpected/invalid result. What is limiting the GCPU to be unable to produce said result?

For your deliverable, provide the VHDL implementation, annotated simulation, and answer from above. No video demos for this lab.

The full lab report is due the Sunday after Thanksgiving at 11:59pm, 11/28.