Fall ’11 CIS 212 Assignment 4 – 110/100 points possible – Due Wednesday, 10-26, 11:59 PM

The goal of this assignment is to increase understanding of computer-organization issues by building an ALU simulator. The book exercises are intended to provide practice with problems related to computer organization.

1. [10] Exercise 5.9, page 233. Describe the multiplexer in terms of its input, control, and output lines.


3. [5] Open Eclipse and create a new project (File -> New -> Project -> Java -> Java Project) named “ALU” (keep all other default settings). Import all of your classes from your LogicGates project except MainFrame. See me if you did not implement all of these classes.

4. [10] Add an abstract ALUCircuit class. The class is abstract because it will not be directly instantiated (i.e., only subclasses need be instantiated); it will serve as a base class for circuits within the ALU which perform bitwise calculations between binary strings. Add two protected instance variables of type SignalGenerator[] and one protected instance variable of type LogicComponent[]. These arrays will hold the generators for the A and B binary inputs and the logic components used to compute each bit, respectively. Implement a constructor which takes an integer width as an argument. The constructor should then create the SignalGenerator arrays based on the width parameter and populate the arrays with SignalGenerator instances (i.e., create arrays with the length of the width parameter). The constructor should not create the LogicComponent array because we do not yet know the type of LogicComponent required; each subclass will use a different type of LogicComponent. Add a public String evaluate(String, String) method which loops through the input strings creating an output string using the appropriate SignalGenerators and LogicComponent for each bit.

5. [5] Add an ANDALUCircuit class which extends ALUCircuit. Implement the constructor to first call the super constructor and then create and populate the LogicComponents array as an array of ANDGate instances using the appropriate SignalGenerator instances. This circuit will perform a bitwise AND on an arbitrary number of binary digits.

6. [5] Add an ORALUCircuit class which extends ALUCircuit. Implement the constructor to first call the super constructor and then create and populate the LogicComponents array as an array of ORGate instances using the appropriate SignalGenerator instances. This circuit will perform a bitwise OR on an arbitrary number of binary digits.
7. [5] Add a class named “XORGate” which extends BinaryLogicGate. Implement the evaluate() method to return the bitwise XOR of the inputs’ evaluate() results. Hint: bitwise XOR in Java (and most languages) is the ‘^’ character.

8. [10] Add a CECircuit class which implements LogicComponent. Add two private instance variables of type LogicComponent which will store the A and B inputs of this gate, similar to the BinaryLogicGate implementation. Add a constructor which takes two LogicComponent input arguments and assigns these parameters to the instance’s A and B input properties (again, similar to BinaryLogicGate). The constructor should also create a logic graph comprised of gate instances which will accomplish the bitwise Compare Equal operation. Hint: Compare Equal can be evaluated easily as (NOT (A XOR B)). Implement the evaluate() method to return the CE result of the inputs’ evaluate() results. Note that this class doesn’t extend BinaryLogicGate because a CECircuit is not itself a BinaryLogicGate, it’s a circuit which is comprised of logic gates.

9. [5] Add an CEALUCircuit class which extends ALUCircuit. Implement the constructor to first call the super constructor and then create and populate the LogicComponents array as an array of CECircuit instances using the appropriate SingalGenerator instances. This circuit will perform a bitwise CE on an arbitrary number of binary digits.

10. [10] Add a FullAddCircuit class which implements LogicComponent. Add three private instance variables of type LogicComponent which will store the A, B, and Carry inputs of this gate. Add a constructor which takes three LogicComponent input arguments and assigns these parameters to the instance’s A, B, and Carry input properties. The constructor should also create two logic graphs comprised of gate instances which will determine the Sum and Carry outputs of the circuit. Hint: the Sum can be evaluated as ((A XOR B) XOR C) and the Carry can be evaluated as ((A AND B) OR ((A XOR B) AND C)); the circuits can share an (A XOR B) gate. Implement the evaluate() method to return the Sum result of the inputs’ evaluate() results. Add an evaluateCarry() method which returns the Carry result of the inputs’ evaluate() results.

11. [10] Add an ADDALUCircuit class which extends ALUCircuit. Add a private variable of type SignalGenerator[] to store the intermediate Carry values. Implement the constructor to first call the super constructor and then create and populate the Carry SignalGenerator array and create and populate the LogicComponents array as an array of FullAddCircuit instances using the appropriate SingalGenerator instances. This circuit will perform a bitwise ADD on an arbitrary number of binary digits. To make this work, override the public String evaluate(String, String) method so that carry-out signals are correctly mapped to carry-in signals in least-to-most significant bit order.
12. [5] Create an ALU class with one private variable of each of the following types: ANDALUCircuit, ORALUCircuit, CEALUCircuit, and ADDALUCircuit. The constructor should take an integer width argument and use it to instantiate each of the circuits. Then add four public methods: `String evaluateAND(String, String), String evaluateOR(String, String), String evaluateCE(String, String) and String evaluateADD(String, String)`. Each of the aforementioned methods should use the appropriate circuit to evaluate a result based on the inputs.

13. [10] Add a class named MainFrame which extends JFrame. When created, the class should instantiate an ALU by passing a width of 8 and then set up a GUI which allows the user to input two binary strings, execute an AND, OR, CE, or ADD operation, and view the results. See extra credit below for validating that the user inputs are in the form of 8-bit binary strings. Your GUI should look something like:

![ALU GUI](image)

14. [+10] (Extra credit) Before executing an ALU operation, test to ensure that the user has entered two 8-bit binary strings as input. If not, alert the user using `javax.swing.JOptionPane` and clear the invalid input.

Zip the ALU folder in your Eclipse workspace directory along with your problem-set solution document (PDF or TXT) and upload the .zip file to Blackboard (see Assignment 4 assignment in the Course Documents area).