CS 211: Computer Architecture, Fall 2015
Programming Assignment 4: Circuit Simulator in C

Instructor: Prof. Santosh Nagarakatte

Due: November 27th, 2015 at 5pm.

1 Introduction

This assignment is designed to get you more experience in C programming while also increasing your understanding of circuits. You will be writing a C program to simulate the output of combinational circuits. There are two parts to this programming assignment. The first part is for regular credit and the second for extra credit.

2 The circuit description directives

For both the parts, one of the inputs to your program will be a circuit description file that will describe a circuit using various directives. We will now describe the various directives.

The input variables used in the circuit are provided using the **INPUTVAR** directive. The **INPUTVAR** directive is followed by the number of input variables and the names of the input variables. All the input variables will be named with upper case alphabets (i.e., A,B,C, ...).

An example specification of the inputs for a circut with three input variables: A, B, C is as follows:

```
INPUTVAR 3 A B C
```

The outputs produced by the circut is specified using the **OUTPUTVAR** directive. The **OUTPUTVAR** directive is followed by the number of outputs and the names of the outputs. All the output variables will be named with upper case alphabets.

An example specification of the circuit with output P is as follows:

```
OUTPUTVAR 1 P
```

The circuits used in this assignment will be built using the following building blocks: **NOT**, **AND**, **OR**, **DECODER**, and **MULTIPLEXER**.

The building blocks can produce temporary variables as outputs. Further, these building blocks can use either the input variables, temporary variables, a boolean '1' or a '0' as input.
Note: Output variables will never be used as inputs in a building block.

All the temporary variables will be named with lower case alphabets (i.e., a, b, c, ...).

The specification of each building block is as follows:

- **NOT**: This directive represents the *not* gate in logic design. The directive is followed by the name of an input and the name of an output.

  An example circuit for a NOT gate ($B = \bar{A}$) is as follows.

  ```
  NOT A B
  ```

- **AND**: This directive represents the *and* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an AND gate ($C = A.B$) is as follows:

  ```
  AND A B C
  ```

- **OR**: This directive represents the *or* gate in logic design. The directive is followed by the names of the two inputs and the name of the output.

  An example circuit for an OR gate ($C = A + B$) is as follows:

  ```
  OR A B C
  ```

- **DECODER**: This directive represents the *decoder* in logic design. The directive is followed by the number of inputs, names of the inputs, and the names of the outputs. The output are ordered in **gray code** sequence.

  An example decoder with two inputs A and B is specified as follows:

  ```
  DECODER 2 A B P Q R S
  ```

  P represents the $\bar{A}\bar{B}$ output of the decoder, Q represents the $\bar{A}B$ output of the decoder, R represents the AB output of the decoder, S represents the $A\bar{B}$ output of the decoder. Note that the outputs of the decoder (i.e., P, Q, R, and S) are in gray code sequence.

- **MULTIPLEXER**: This directive represents the *multiplexer* in logic design. The directive is followed by the number of inputs, names of the inputs, names of the selectors, and the name of the output. The inputs are ordered in **gray code** sequence.

  A multiplexer implementing a AND gate ($P = A.B$) using a 4:1 multiplexer is specified as follows:

  ```
  MULTIPLEXER 4 0 0 1 0 A B P
  ```

  The above description states that there are 4 inputs to the multiplexer. The four inputs to the multiplexer in gray code sequence are 0 0 1 0 respectively. The two selector input signals are A and B. The name of the output is P.

3 Describing circuits using the directives

It is possible to describe any circuit using the above set of directives. For example, the circuit $Q = AB + AC$ can be described as follows:

```
INPUTVAR 3 A B C
OUTPUTVAR 1 Q
AND A B w
AND A C x
OR w x Q
```

Note that Q is the output variable, A,B,C are input variables, and w and x are temporary variables. Here is another example:

```
INPUTVAR 4 A B C D
OUTPUTVAR 1 P
AND A B w
MULTIPLEXER 4 0 1 0 1 w v P
OR C D v
```

As seen above, a circuit description is a sequence of directives. If every temporary variable occurs as a output variable in the sequence before occuring as an input variable, we say that the circuit description is **sorted**. For example, the circuit description in the first example is sorted whereas the one in the second example is not.
Note: A temporary variable can occur as an output variable in at most one directive.

4 Format of the input files

As you will see in the problem statement below, your program will be given two files as input. One of the files will contain the description of a circuit using the directives described above. The other file will be an input values file. Each line of the input values file will be an assignment to variables in the INPUTVAR specification in the circuit description file.

For example, say that the circuit description files contains the following:

```
INPUTVAR 3 A B C
OUTPUTVAR 1 Q
AND A B w
AND A C x
OR w x Q
```

Then an example of an input values file is

```
1 0 1
1 1 1
```

Here the first line corresponds to the assignment $A = 1$, $B = 0$, and $C = 1$, and the second line to the assignment $A = 1$, $B = 1$, and $C = 1$.

5 Common instructions

Here are some instructions common to both parts of the assignment:

- You have to a write a C program that takes two file names as command line arguments.
- The first file name will be that of the circuit description file, and the second that of the input values file.
- For every line in the input values file, the program should interpret it as an assignment to the input variables, evaluate the circuit on that assignment, and output the values of the output variables.
- The values of the output variables should be space separated and be in the same order as the output variables in the OUTPUTVAR directive, e.g., if the circuit description file has the directive OUTPUTVAR 3 P Q R, then the first value should be that of the output variable P, followed by that of Q, and then that of R.
- For every line in the input values file, the output should be on a new line.

6 The regular credit problem (100 points)

For the regular credit problem, you have to write a program called `first` as described above. For this part, you are guaranteed that the circuit descriptions given as input to your program will be **sorted**. Let's look at an example we've encountered before.

Example Execution 1

Suppose a circuit description file named circuit.txt has the description for the circuit $Q = AB+AC$:

```
INPUTVAR 3 A B C
OUTPUTVAR 1 Q
AND A B w
AND A C x
OR w x Q
```

and an input values file named input.txt for the above circuit is as follows:

```
1 0 1
0 0 1
```

Then, on executing the program with the above circuit description file and input file, your program should produce the following output (remember: one line of output for each input in the input file).

```
./first circuit.txt input.txt
1
0
```

The output of the circuit $Q = AB + AC$ when A is 1, B is 0 and C is 1 is 1. Hence the first line is 1 in the output. Similarly, the output of the circuit is 0 when A is 0, B is 0, and C is 1.

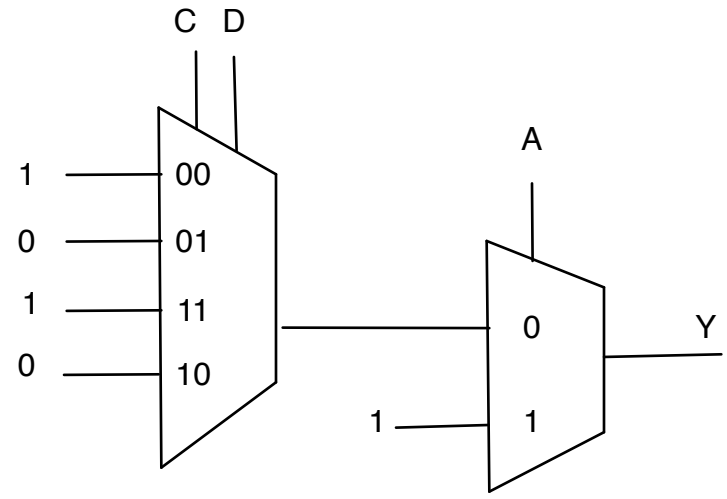

Figure 1: Circuit with Multiplexers

Example Execution 2

The circuit description file, circuit.txt, for the circuit in Figure 1 is as follows:

```
INPUTVAR 3 A C D
OUTPUTVAR 1 Y
MULTIPLEXER 4 1 0 1 0 C D w
MULTIPLEXER 2 w 1 A Y
```

The input values file, input.txt, for the circuit is as follow:

```
1 1 1
1 0 0
0 1 0
```

When we execute the program the output should be as follows:

```
./first hw1-circuit.txt hw1-input.txt
1
1
0
```

7 The extra credit problem (Extra credit)

For the extra credit problem, you have to write a program called `second` as described above. For this part, the circuit descriptions given as input to your program **need not be sorted**. Let's take up an example we saw before:

Example Execution 1

Suppose the circuit description file, circuit.txt, contains the following:

```
INPUTVAR 4 A B C D
OUTPUTVAR 1 P
AND A B w
MULTIPLEXER 4 0 1 0 1 w v P
OR C D v
```

(Note that the description is **not sorted**.), and the input values file, input.txt, contains:

```
1 0 0 1
1 1 1 0
0 0 0 0
```

Then the execution of the program should result in the following:

```
./second circuit.txt input.txt
1
0
0
```

8 Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa4.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa4`. Then, `cd` into the directory containing `pa4` (that is, `pa4`'s parent directory) and run the following command:

```
tar cvf pa4.tar pa4
```

To check that you have correctly created the tar file, you should copy it (`pa4.tar`) into an empty directory and run the following command:

```
tar xvf pa4.tar
```

This should create a directory named `pa4` in the (previously) empty directory.

The `pa4` directory in your tar file must contain 2 subdirectories, one each for each of the parts. The name of the directories should be `first` and `second` (in lower case). Each directory should contain the necessary source files, header files, and a make file. Running the makefile in the `first` folder, should produce the binary `first`, and doing the same in the `second folder` should produce the binary `second`.

We will provide a autograder and many more test cases, which can be used to test your submission and your program.

9 Grading Guidelines

This is a large class so that necessarily the most significant part of your grade will be based on programmatic checking of your program. That is, we will build a binary using the Makefile and

source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **up to 100% penalty**. There should be no additional information or newline. That means you will probably not get any grade is you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask.