CS 2110 Homework 03
Arithmetic Logic Units

Jered Tupik, Cem Gokmen, Austin Adams, Madison Grams, Vivian De Sa Thiebaut

Fall 2018

Contents

1 Overview

All computer processors have a very important component known as the Arithmetic Logic Unit (ALU). This component allows the computer to do, as the name suggests, arithmetic and logical operations. For this assignment, you're going to build an ALU of your own, along with creating some of the gates.

For this assignment you will:

1. Create the standard logic gates (NAND, NOR, NOT, AND, OR)
2. Create an 8-input multiplexer and an 8-output decoder
3. Create a 1-bit full adder
4. Create an 8-bit full adder using the 1-bit full adder
5. Use your 8-bit full adder and other components to construct an 8-bit ALU

This assignment will be demoed alongside Homework 4. More information on this and the sign-up schedule will be posted on Canvas. An announcement will be sent out and it will also be announced in Lecture/Lab when the schedule is up. **You have to be present for the demo in order to get credit for the demo-portion of the assignment.**

2 Instructions

2.1 Requirements

For the first part of this assignment (the logic gates), you may use only those components found in the Wiring tab (being the input/output pins, constants, probes, clocks, splitters, tunnels, and transistors), along with any of the sub-circuits that you create or that already exist.

For the second part of this assignment (the multiplexer and decoder), you may use only those components found in the Wiring and Gates tabs along with any of the sub-circuits that you create or that already exist.

For the third part of this assignment, you may use only those components found in the Wiring and Gates tabs as well as any of the sub-circuits that you create or that already exist. **For the ALU specifically, you may use the Plexer tab as well.**

Use of anything not listed above will result in heavy deductions. Your need to have everything in their correctly named sub-circuit. More information on sub-circuits is given below.

Use tunnels where necessary to make your designs more readable, but do not overdo it! For gates, muxes, adders and decoders you can often get clean circuits just by placing your components well rather than using tunnels everywhere.

2.2 Part 1: 1-Bit Logic Gates

All of the circuits in this file are in the *gates.sim* file, and you may use the **Wiring** tab for this part.

For this part of the assignment, you will create a transitor-level implementation of the NAND, NOT, NOR, AND, and OR logic gates.

Remember for this section that you are only allowed to use the components listed in the Wiring section, along with any of the logic gates you are implementing in CircuitSim. For example, once you implement the NOT gate you are free to use that subcircuit in implementing other logic gates. Implementing the gates in the order of the subcircuit tabs can be the easiest option.

As a brief summary of the behavior of each logic gate:

A	**B**	**A NAND B**
0	0	1
0	1	1
1	0	1
1	1	0

A	**B**	**A NOR B**
0	0	1
0	1	0
1	0	0
1	1	0

A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

A	**B**	**A OR B**
0	0	0
0	1	1
1	0	1
1	1	1

A	**NOT A**
0	1
1	0

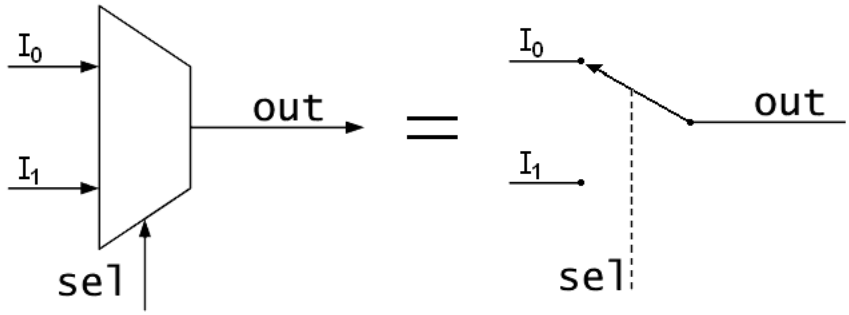
Hint: Start by creating the NAND gate from transistors. Then use this gate as a subcircuit for implementing the others.

All of the logic gates must be within their named sub-circuits.

2.3 Part 2: Plexers

All of the circuits in this file are in the *plexers.sim* file, and you may use the **Wiring, Gates** tabs for this part.

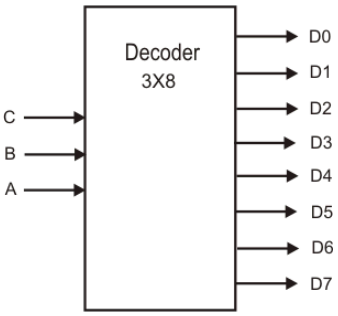
The multiplexer you will be creating has 8 1-bit inputs (labeled appropriately as `A`, `B`, `C`, ..., `H`), a single 3-bit selection input (`SEL`), and one 1-bit output (`OUT`). The multiplexer uses the `SEL` input to choose a specific input line for forwarding to the output.



For example:

```
SEL = 000 ==> OUT = A
SEL = 001 ==> OUT = B
SEL = 010 ==> OUT = C
SEL = 011 ==> OUT = D
SEL = 100 ==> OUT = E
SEL = 101 ==> OUT = F
SEL = 110 ==> OUT = G
SEL = 111 ==> OUT = H
```

The decoder you will be creating has a single 3-bit selection input (`SEL`), and eight 1-bit output (labeled `A`, `B`, `C`, ..., `H`). The decoder uses the `SEL` input to raise a specific output line, as seen below.



For example:

```
SEL = 000 ==> A = 1, BCDEFGH = 0
SEL = 001 ==> B = 1, ACDEFGH = 0
SEL = 010 ==> C = 1, ABDEFGH = 0
SEL = 011 ==> D = 1, ABCEFGH = 0
SEL = 100 ==> E = 1, ABCDFGH = 0
SEL = 101 ==> F = 1, ABCDEGH = 0
SEL = 110 ==> G = 1, ABCDEFH = 0
SEL = 111 ==> H = 1, ABCDEFG = 0
```

2.4 Part 3: Adders & ALUs

All of the circuits in this file are in the *alu.sim* file, and you may use the **Wiring and Gates** tabs for this part. **For the ALU specifically, you may use the Plexer tab as well.**

2.4.1 1-Bit Adder

The full adder has three 1-bit inputs (`A`, `B`, and `CIN`), and two 1-bit outputs (`SUM` and `COUT`). The full adder adds `A + B + CIN` and places the sum in `SUM` and the carry-out in `COUT`.

For example:

```
A = 0, B = 1, CIN = 0 ==> SUM = 1, COUT = 0
A = 1, B = 0, CIN = 1 ==> SUM = 0, COUT = 1
A = 1, B = 1, CIN = 1 ==> SUM = 1, COUT = 1
```

Hint: Making a truth table of the inputs will help you.

2.4.2 8-Bit Adder

For this part of the assignment, you will daisy-chain 8 of your 1-bit full adders together in order to make an 8-bit full adder.

This circuit should have two 8-bit inputs (`A` and `B`) for the numbers you're adding, and one 1-bit input for `CIN`. The reason for the `CIN` has to do with using the adder for purposes other than adding the two inputs.

There should be one 8-bit output for `SUM` and one 1-bit output for `COUT`.

2.4.3 8-Bit ALU

You will create an 8-bit ALU with the following operations, using the 8-bit full adder you created in for 2.4.2:

000. Addition	`[A + B]`
001. Subtraction	`[A - B]`
010. 2's Complement Negation	`[-A]`
011. Multiply by 13	`[A * 13]`
100. isMultipleOf8	`[A % 8 == 0]`
101. isPowerOf2	[$A == 2^n$ for some $n \in N$, including $2^0 = 1$]
110. XOR	`[A XOR B]`
111. Constant Output	`[0x69]`

Notice that 2's Complement Negation, Multiply by 13, isMultipleOf8, and isPowerOf2 only operate on the `A` input. **They should NOT rely on `B` being a particular value.** Likewise, Constant Output does not use either input and should not rely on their value.

This ALU has two **8-bit** inputs for `A` and `B` and one **3-bit** input for `OP`, the op-code for the operation in the list above. It has one **8-bit** output named `OUT`.

The provided autograder will check the op-codes according to the order listed above (Addition (`000`), Subtraction (`001`), etc.) and thus it is important that the operations are in this exact order.

Hint: For check if a number is a power of two, remember your implementation in Homework 2. The best implementation of this is the expression $(a > 0 \&\& ((a \& (a-1)) == 0))$ which is easy to implement in digital logic.

3 Deliverables

Please upload the following files onto the assignment on Gradescope:

1. `gates.sim`
2. `plexers.sim`
3. `alu.sim`

Be sure to check your score to see if you submitted the right files, as well as your email frequently until the due date of the assignment for any potential updates.

No partial credit will be given for incorrect outputs for Part 1, Part 2, and the adder-portion of Part 3. For the ALU, partial credit will be awarded on a per-operation basis, wherein each operation must perform successfully to be awarded credit. Because of this, we urge you to check your score before the due date.

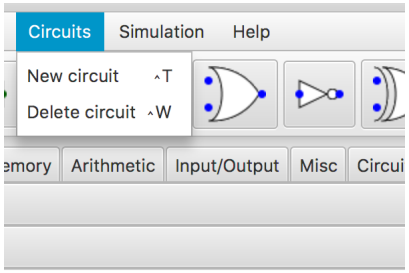
Once again, this assignment will be demoed alongside Homework 4! More information on this and the sign-up schedule will be posted on Canvas. An announcement will be sent out and it will also be announced in Lecture/Lab when the schedule is up. **You have to be present for the demo in order to get credit for the demo-portion of the assignment.**

4 Sub-Circuit Tutorial

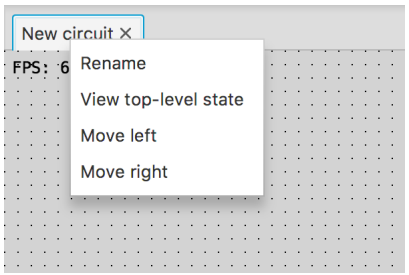
As you build circuits that are more and more sophisticated, you will want to build smaller circuits that you can use multiple times within larger circuits. Sub-circuits behave like classes in Object-Oriented languages. Any changes made in the design of a sub-circuit are automatically reflected wherever it is used. The direction of the IO pins in the sub-circuit correspond to their locations on the representation of the sub-circuit.

To create a sub-circuit:

1. Go to the "Circuits" menu and choose "New circuit"

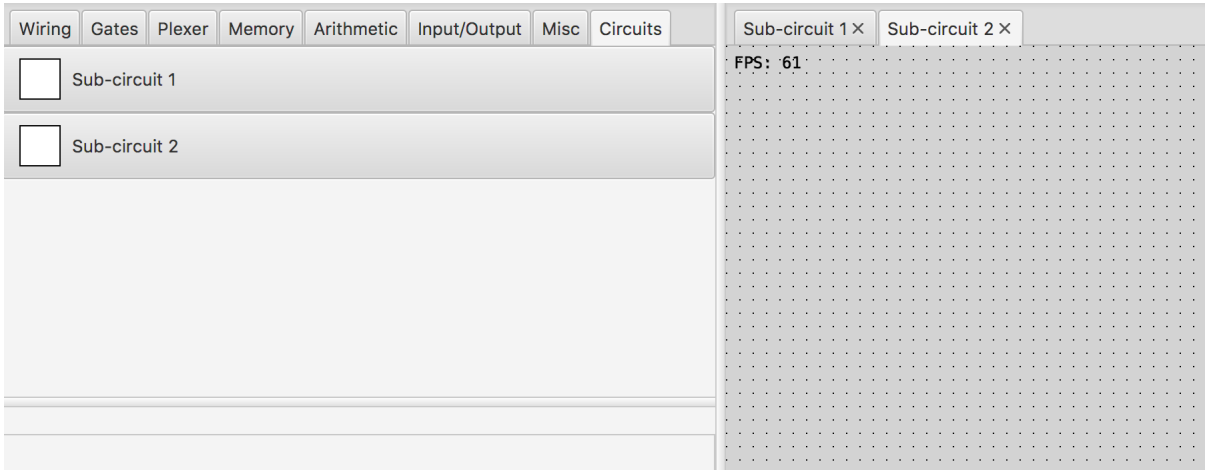


2. Name your circuit by right-clicking on the "New circuit" item and selecting "Rename"

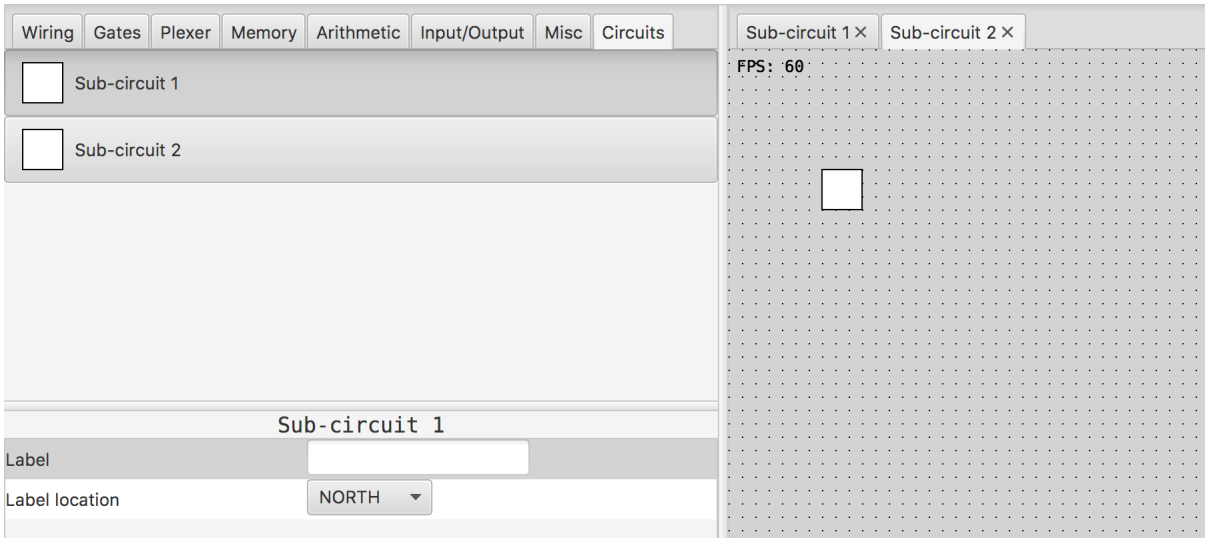


To use a sub-circuit:

1. Click the "Circuits" tab next to the "Misc" tab



2. Select the circuit you wish to use and place it in your design



5 Rules and Regulations

5.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.

2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

3. Please read the assignment in its entirety before asking questions.

4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

5.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

5.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

5.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

5.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own. Consider instead using a screen-sharing collaboration app, such as `http://webex.gatech.edu/`, to help someone with debugging if you're not in the same room.

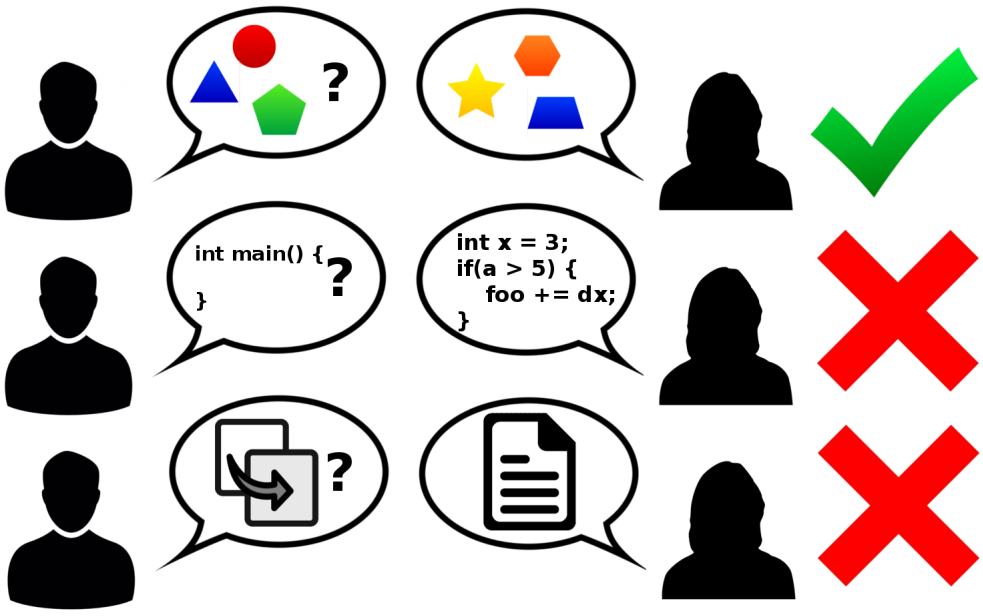


Figure 1: Collaboration rules, explained colorfully