# **CSC 335: Comp Org Midterm**

It is expected that you may use your text and resources found online (but not forums where you actively ask questions) in completing this exam. It should represent your own work, and not collaborative work with others.

If you have questions, you may address them to the instructor. I will reply promptly and to the best of my ability. My email address remains [jadudm@berea.edu](mailto:jadudm@berea.edu), and my phone number remains 1-440-334-6685. I do turn my phone off when I go to sleep, so late-night texts/phone calls 1) will not wake me up and 2) will not reach me faster than email.

## **Submission**

Your answers can be typed or hand-written. Your code should probably be written in Cloud9’s text editor. Diagrams may be hand-drawn, or drawn with a diagramming tool.

Anything you hand-write or hand-draw (all of which must be NEAT) you may hand in, in person, at the start of class on Thursday. Anything digital (diagrams, code, etc.) should be uploaded to Moodle by the same time.

## 

## 

## **Number Bases**

Complete the following conversions.

|  |  |  |
| --- | --- | --- |
| Decimal | Hex | Binary |
| 2 | 2 | 10 |
| 10 | A | 1010 |
| 16 | 10 | 00010000 |
| 42 | 2A | 000000101010 |
| 65535 | FFFF | 1111111111111111 |
| 4294967296 | 100000000 | 100000000000000000000000000000000 |
| 812018375029 | BD10111975 | 1011110100010000000100010001100101110101 |

## 

## 

## **Numbers, in Context**

1. The Arduino is a small, low-cost computing platform that students in TAD 265 build. It is powered by the Atmel ATMega328P processor. How much flash does it have, in bytes? **When I looked the value up for the arduino it stated that there was 32,768 bytes.**
2. A 32-bit computer can uniquely address 232 locations in memory. How many bits are needed to uniquely address that many bytes? **32 bits (it's a 32-bit computer which means it needs 32 bits to uniquely address the locations in memory.)**
3. Based on your answer to #2, how many bits are needed to address all of the flash on an Arduino? **16 bit**
4. This suggests how many bits are used to represent an int on an Arduino. Would you say that an Arduino is an 8-, 16-, 32-, or 64-bit processor based on this information? **I believe that the arduino has a 16 bit processor**
5. Assuming the Arduino represents negative numbers using a 2’s complement notation, what is the largest positive integer that can be represented on an Arduino? **I believe the answer to be 32,767 which is (2^15)-1**
6. Likewise, what is the largest negative integer that can be represented on an Arduino? **The largest negative integer is -32,768**

## 

## 

## **Running the ‘Net**



Due to limitations of the drawing tool used, the inputs for any given gate appear to merge into one. For example, the topmost AND gate has the inputs {A, B}, while the lower AND gate has the inputs {B, B}.

1. Please write out the truth table for the above network.

## 

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **OUT** |
| **0** | **0** | **1** |
| **0** | **1** | **1** |
| **1** | **0** | **1** |
| **1** | **1** | **0** |

## 

## 

## **A Bit of HDL**

The following is the HDL for the Half Adder.

CHIP HalfAdder {

IN a, b;

OUT sum,carry;

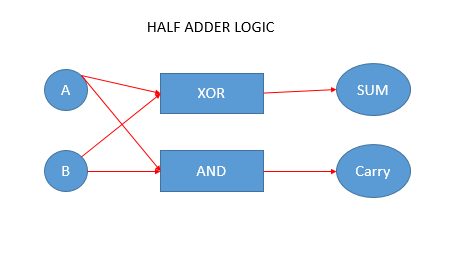
PARTS:

Xor(a=a, b=b, out=sum);

And(a=a, b=b, out=carry);

}

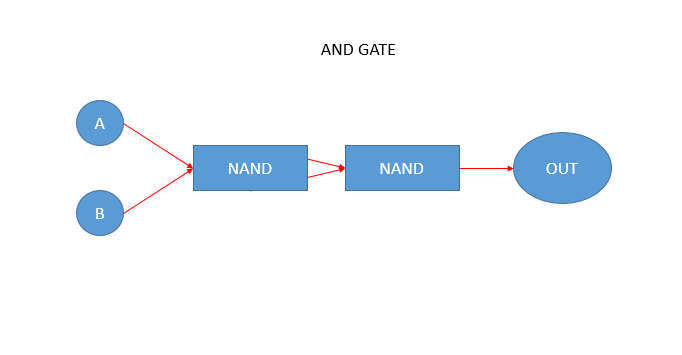
1. Draw the logic network that represents this part.



1. You have previously defined AND and XOR in terms of only NAND; draw the same network using only NAND gates.

## 

The out for the image above represents the Sum in the half-adder



The Out in the image above represents the carry in the half adder.

## **A Bit of C**

Matt wants to test his HalfAdder code in C. He was going to have you just answer questions about his code, but he ran out of time, and left it unfinished. As a result, you’ll have to both write some C as well as answer questions about the code you wrote when you’re done.

1. Create a new file in your C9 workspace called halfadder.c.
2. Copy and paste the code below into this file. (Note the **Raw** button.)
3. Compile the file with the command:  
   gcc -o ha halfadder.c
4. Run the file with the command:  
   ./ha

The code can be had here:

<https://gist.github.com/jadudm/5b22e19c6a182ae70e33>

You can get an easily copy-pasted version of the code by clicking the **Raw** button in the Gist interface.

Your task is to work through the following questions, some of which ask you to explain the code that is there, and others that ask you to add to the code that is there. Line numbers refer to the code before you edit it.

The code should compile and run as given. If it does not, you should contact Matt immediately. If you have forgotten how to change directories and run commands on the command line, however, that speaks to a lack of practice, not a problem that Matt can help you with.

### Questions for A Bit of C

1. We typically think of the boolean gate AND as having two inputs and one output; in C, this might be thought of as having two parameters and one return value. Looking at the code Matt wrote, what kind of information does the And() function return (line 10)? **The beginning of the And function starts with a void. Therefore the function does not return anything. However, the function does insert a boolean value which can be either true or false into the out\*.**
2. Based on your reading of the And() function, how does it “return” the result of its logical computation?**The And function returns the result by assigning the boolean value into the space that is allocated by the out pointer.**
3. What do lines 37 and 38 do?

**Lines 37 and 38 allocate space in the memory for the variables r1 and r2.**

1. Comment out lines 37 and 38. Recompile and re-run the code. What happened? Explain why this happened, to the best of your understanding. Detail matters.

**This action would result in a segmentation fault. By getting rid of the malloc the program would then call for space that it doesn’t have. Which would result in your program trying to access an area of memory that was not requested in the program. Malloc has the unique ability of asking for space during the run time, and so when the space is not requested the space is unable to be accessed.**

1. The pointer r1 is passed to the function And() on line 42. What role does it play? Or, put another way, how is it used by the And() function?

**R1 on line 42 is responsible for passing in a reference of the output of the And gate.**

1. Lines 29-33 define a 2-dimensional array. What, given its values, does it look like Matt is defining?

**Matt is defining all the inputs that are being tested in the And gate.**

1. There are three constants defined on lines 6, 7, and 8. In particular, Matt defines the constant A as being 0, and B as being 1. On line 42, he uses these two constants. Specifically, he writes:  
     
   tests[ndx][A]  
     
   This is referencing into the array. What value does this expression simplify to when the value of ndx is 0? **{false}**
2. Following from the previous question, what value does tests[ndx][A] simplify to when ndx is 3? {**true}**
3. Why do you think Matt defined the constants A and B in this code? Explain.

**A and B are defined as constants so that it can make our lives easier. Matt defines them as constants with the forethought that one of the values may need to be changed later on in life. Therefore, if the variables are constants this can easily be done inside of the initialization of the constant variable. Otherwise, if Matt wanted to change A to 1 he would be forced to find every occurrence of the value and manually change it.**

1. Where in memory is the space for Matt’s testing array allocated? Explain.

**The testing array is defined inside of the main function. Therefore, the space for the memory containing that array would be stored on the stack.**

1. When is space for Matt’s testing array allocated? Explain.

**The space for the testing array is a form of static allocation. Therefore, the space created for the data would occur during the compiling process.**

1. Where in memory is the space for r1 and r2 allocated? Explain.

**r1 and r2 are initiated using the malloc therefore they are dynamically accessed which means that the space for those variables should be located on the heap.**

1. When is memory for r1 and r2 allocated? Explain.

**As I mentioned in my statement above the memory is dynamically allocated. This means that the space is requested in memory after the compiling process. Which means that they are asked for during the run time process of the program.**

1. Complete the implementation for Xor() (lines 18-20). You may find the implementation of And() to be inspirational.

**I completed this in the files I submitted**

1. Matt left the for loop on lines 54-56 empty; you should add a call to Xor() and a printf() statement to that loop to show that your implementation is correct. You may find the previous loop to be inspirational.

**I completed this in the files I submitted**

1. On line 22, Matt wrote bool\* sum instead of bool \*sum. Is there a difference? Explain, as best you can. (*Hint: leave it as-is, and then try it both ways when you’re done.*)

**I believe changing the spaces for the two values to have absolutely no effect on the program. The bool\* method is a bool type pointer that is called sum whereas the opposite bool \*sum is a pointer called sum that is of type bool. I believe that they are just different methods for labeling the values and have no effect on the program in total.**

1. Complete the implementation for HalfAdder() (lines 22-24). Your solution should only make use of the functions And() and Xor(), which you have completed previously.

**I completed this in the files I submitted**

1. Matt left you another for loop. In it, call HalfAdder(), and include a printf() statement that demonstrates that it performs as expected.

**I completed this in the files I submitted**

1. From your homework, you implemented the full adder. What would it look like, in C, using your HalfAdder() function? Provide an implementation that you believe would work. (That is, this is “representative C code,” or “barely-pseudo-C code” that I am asking for, not a complete implementation.)

**My representive C code is symbolic to the logic we did in the full adder hdl.**

**//The full adder still needs to take in 3 values and output a sum and carry**

void FullAdder ( bool a, bool b, bool c, bool \*sum, bool \*carry)

**//Therefore I initialized by function with the mindset**

**/////////////////////////////////Variable declarations///////////////////////////////////////**

bool \*sumAB //Record the sum of variables A and B

bool \*carryAB //Record the carry of variables A and B

bool \*carryABC //Record the carry of variables A, B, and C put together

//These variables can be defined and created inside of the function, and are used to track

the outcome of other functions.

///////////////////////////////Variable Declaration Over//////////////////////////////////////

HalfAdder(a,b, sumAB, carryAB) //Use a half adder to find the sum and carry of A and B

HalfAdder(sumAB , c , sum, carryABC) //Use a half adder to and c to the sum of a and b and to find the carry of A, B and C

//Create an Or function that opperates simliarly to the Or gate that takes in two values and returns the a true or false

Or(carryAB, carryABC, carry) //Use the Or gate to find the overall carry

1. In your implementation of the ALU, Matt suggested you should only return the output value. Describe, given this exercise, how you might rewrite the ALU so that it can return the output as well as values for ng and zr. (*Hint: this is meant to be a question of what parameters you would pass to the ALU function, and how you would use them.*)

**I believe the best possible solution to this problem would be to pass the ALU two bool pointers. One pointer would be used for zr and the other for ng. Then we could manipulate the values that are stored into the pointers with some condition statements that would be implemented after the Out value has been assigned at the end of the alu. if out == 0 then we could store a true into the pointer for zr. Then for ng pointer, if out < 1 store the value true into the pointer. Then an else statement could store false into both of these pointers.**

1. Copy your alu() from the homework into a new file. Explain, with comments, what each line does. Where appropriate, you may describe several similar lines at the same time.  
   **I completed this in the files I submitted**
2. Copy your main() from your homework into a new file. Explain, with comments, what each line does. Where appropriate, you may describe several similar lines at the same time.

**I completed this in the files I submitted**

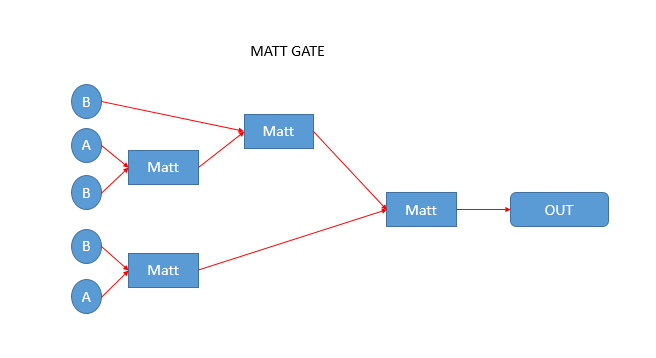
## **The New NAND: MATTGATE**

At the start of Elements of Computing Systems, we built a series of gates from only the NAND gate. Matt has designed a new gate, which he calls the MATTGATE. It obeys the following truth table:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **Out** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Like quantum computing, this gate promises to revolutionize the computing world! Remember: *you saw it here first*.

1. Using only the MATTGATE, propose a network that, given inputs A and B, produces the output for AND.



**In the above diagram if an arrow is above another that means that it is the first input into the gate.**

1. As part of our work in class, we constructed a NOT gate from a NAND gate. Can you construct a NOT gate from an AND gate? Demonstrate/explain.  
   **I don’t any combination of and gates put together could change the values, because a Not gate takes in only one input. Which means you would have to construct all the and gates with only one input. Which means that the gates would only ever output the value that they were given.**
2. Matt’s original version of this question assumed that all gates were *universal*. That is, he was confident, based on his first step of creating an AND gate, that he could *obviously* create all other gates. Research the notion of “gate universality” and explain this concept to Matt.

**Gate universality basically says that given enough gates, either type type of gate is able to mimic the operation of any other gate type.**

1. Has Matt actually developed the next great thing? More specifically, do you think he can produce a general-purpose CPU (with an ALU, RAM, and all the other good bits) from his new gate?**I do not find this to be possible with the matt gate for one major reason. That reason being that there is no way to make a NAND gate, because we can’t manipulate the input A=0 and B=0 to be one.**