CIS 2400 Fall 2022: Final Exam

Dec 15, 2022

First Name : _____

Last Name : _____

Penn ID : _____

Please fill in your information above, read the following pledge, and sign in the space below:

I neither cheated myself nor helped anyone cheat on this exam. All answers on this exam are my own. Violation of this pledge can result in a failing grade.

Sign Here : _____

Exam Details & Instructions:

- There are 7 questions (and a short bonus 8th question) worth a total of 100 points.
- You have 120 minutes to complete this exam
- You will be provided with two pages containing references sheets.
  Do not put any answers on these pages, they will not be graded.
- The exam is closed book. This includes textbooks, phones, laptops, wearable devices, other electronics, and any notes outside of what is mentioned below.
- You are allowed two 8.5 x 11 inch sheets of paper (double sided) for notes.
- Any electronic or noise-making devices you do have should be turned off and put away.
- Remove all hats, headphones, and watches.

Advice:

- Remember that there are 7 questions (and a short bonus 8th question). Please budget your time so you can get to every question.
- Do not be alarmed if there seems to be more space than needed for answer, we try to include a lot of space just in case it is needed.
- **Try to relax and take a deep breath.** Remember that we also want you to learn from this.

Please put your initials at the top of each page in case the pages become separated.

If you need extra space, the back of the last page of this exam is blank for you as scratch space and to write answers. If you use it, please clearly indicate on that page and under the corresponding question prompt that you are using the extra page to answer that question. Please also write your full name and PennID at the top of the sheet.

Question 1 {15 pts}

Your job is to design a circuit that will take as input a 3-bit value representing an unsigned integer and produces the output 1 if and only if the value is 3 or 5.
For example, if I = 101 then the circuit should output 1. In your diagram I_2, I_1 and I_0 should indicate the 3 bits of the input where I_2 is the MSB and I_0 is the LSB. Remember that we cannot grade what we cannot read so please make your diagrams as neat and clear as possible.

Part 1 {7 pts}: Design a proper **CMOS** circuit which takes in all 3 bits of the input I and produces the output bit. You can assume that you also have access to negated versions of all of the input bits. Your solution must be a single CMOS circuit consisting of complementary pull-up and pull-down transistor networks. It should not involve cascading multiple CMOS circuits. Please label the inputs and outputs of your circuit clearly on your schematic.

Part 2 {8 pts}: Design a gate level circuit that takes in the 3-bit input I and produces the correct output. In addition to producing the correct output, **your solution should only use inverters and 2-input gates.**

Your solution should also minimize the number of gates used to make the circuit. These include: NOT (inverter), AND, OR, XOR, NAND, NOR, XNOR.

You may use a single type of gate more than once, but **to get full credit the total number of used gates (including inverters) must be at most 3**. Partial credit will be given for correct solutions that use 4 gates, less partial credit for 5 gates, even less partial credit for 6 gates and no credit for anything that uses 7 or more gates. Note that using a PLA will not work for this question.

Please label the inputs and outputs of your circuit clearly on your schematic.

Question 2 {15 pts}

Your job is to write an LC4 assembly subroutine that checks to see if an array is sorted. The subroutine should take as inputs:

- R0: an address to the start of an array of integers in memory
- R1: The length of that array '`n`'.

Your subroutine should check the array specified to see if it is sorted, storing 1 in R2 if it is sorted, and 0 in R2 if it is not sorted. The array is sorted if each element after the first element is greater than or equal to the previous element in the array.

For example: [-12, 2, 5, 5, 7] is sorted but [3, 4, 3, 4, 5] is not sorted.

We have provided the assembler directives, some labels and some comments to start the program. Your program should execute a RET pseudo-instruction when it is finished.

Additionally, you should also note that:

- Since this is a subroutine, **you can NOT modify R7** in your program
- **The data stored in the array that R0 points should not be modified**
- The inputs in R0, and R1 have been set for you, you do not need to set the input values in your program
- R0 contains a valid address to an array in data memory
- R1 can be assumed to be at least 2
- R2 is where the result should be stored for this subroutine
- You are free to modify any of the registers that aren't R7.
- You are free to add additional labels and comments as needed

The rest of this page can be used as work space to write pseudo code or whatever you like. Your answer should be written on the next page.

```
.CODE
.FALIGN

SORTED
        ; start writing code here









END     ; end of program
    RET
```

Question 3 {5 pts}:

The LC4 architecture is designed to be small with only a few supported operations. While we can do a lot with the instructions provided, there can be desires to add new instructions to the language.

Adding new instructions to the LC4 single cycle processor requires changing the hardware to some extent (at minimum in the decoder so that it can decode new instructions). Adding more transistors and logical units to our LC4 single-cycle processor design would have costs. One of these costs is that if we ran the same LC4 assembly code on our updated single-cycle processor and on the version before the change, the updated processor would either run the program with the same speed or slower.

Please explain why adding new instructions could make our processor slower. We are looking for a specific concept/idea though it doesn't have to be named directly. Please use 4 sentences at maximum to write your answer.

Question 4 {10 pts}
Consider the Single Cycle LC4 instruction set that we walked through in class. Suppose we wanted to add a new instruction called INCR that incremented the value of a register by 1.

Mnemonic: INCR Rd
Semantics: Rd = Rd + 1
Encoding: 1110 dddx xx00 0001

The usual rules for updating the Privilege, PC and NZP apply.

Indicate how each of the following control signals should be set to execute this instruction. If a control signal doesn't matter for this instruction, you must instead write an X.

Answer:

Control Singal Name	Value
PCMux.CTL	
rsMux.CTL	
rtMux.CTL	
rdMux.CTL	
regFile.WE	
regInputMux.CTL	
NZP.WE	
DATA.WE	
Privilege.CTL	
ALUInputMux.CTL	
ALU.CTL	

Question 5 {22 pts}
Answer the following questions about the following LC4 code. This code was generated by lcc based off a C function that takes in a single parameter **arg** and returns an integer.

```
;;;;;;;;;;;;;;;;;;;;;;;;mystery;;;;;;;;;;;;;;;;;;;;;;;;
                .CODE
                .FALIGN
mystery
        ;; prologue
        STR R7, R6, #-2 ;; save return address
        STR R5, R6, #-3 ;; save base pointer
        ADD R6, R6, #-3
        ADD R5, R6, #0
        ADD R6, R6, #-1 ;; allocate stack
                        ;; space for local variables
        ;; function body
        LDR R7, R5, #3
        CONST R3, #0
        CMP R7, R3
        BRnp L2_mystery
        CONST R7, #0
        JMP L1_mystery
L2_mystery
        LDR R7, R5, #3
        STR R7, R5, #-1
        SRL R3, R7, #1
        ;; ********* HERE *********
        ADD R6, R6, #-1
        STR R3, R6, #0
        ;; ********* HERE *********
        JSR mystery
        LDR R7, R6, #-1
        ADD R6, R6, #1
        LDR R3, R5, #-1
        AND R3, R3, #1
        ADD R7, R3, R7
L1_mystery
        ;; epilogue
        ADD R6, R5, #0  ;; pop locals off stack
        ADD R6, R6, #3  ;; free space for return address,
                        ;; base pointer, and return value
        STR R7, R6, #-1 ;; store return value
        LDR R5, R6, #-3 ;; restore base pointer
        LDR R7, R6, #-2 ;; restore return address
        RET
```

Part 1 {5 pts}:
There were two lines of comments saying "`;; ********* HERE *********`".
In between those two comments are two LC4 instructions. What are the purpose of those two lines of assembly? What is it doing? Please describe what it does in terms used to describe a C program. **Do NOT just explain the exact actions that are carried out and do not say just say "A value is pushed on the stack".**

Part 2 {15 pts}: The LC4 Assembly above fits into the following code skeleton. Fill in the missing parts below. Multiple blank lines have been given in some spots, and it is possible you may not need to fill in every line to get the correct answer.

```
int mystery(unsigned int arg) {
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  ____________________________
  return ____________________;
}
```

Part 3 {2 pts}: At a high-level, what does this function appear to be doing? Please justify your answer in 4 or fewer sentences.

Question 6 {20 pts}

Complete the following function:

```
// Creates a copy of the array `arr` but all instances
// of `x` are removed. The new array should be dynamically
// allocated and returned through `output`.
//
// Arguments:
// - arr: the input array, whose contents should not be
//        modified. You can assume this pointer is valid.
// - len: the length of the input array. You can assume
//        len is >= 1.
// - x: The value that we are filtering on. There should be
//      no elements with the value `x` in the output array.
// - output: an output parameter to "return" the resulting
//           array. You can assume this pointer is valid.
// Return Value:
// - The length of the newly allocated array
// - Negative one (-1) on error

int filter(int * arr, int len, int x, int** output) {









}
```

Question 7 {12 pts }
Consider the following two C functions that perform the same task:

```
int factorial_iter(int n) {
  int res = 1;
  int i;
  for (i = 2; i <= n; i++) {
    res = res * i;
  }
  return res;
}

int factorial_recursive(int n) {
  if (n <= 1) {
    return 1;
  }
  return n * factorial_recursive(n - 1);
}
```

When both functions are compiled with the lcc compiler (C to LC4 compiler), each function ends up being around 30 instructions if you include the prologue and the epilogue. Despite this, one of these is slower to execute for larger inputs than the other.

Consider the case where a user wants to calculate 10 factorial, which implementation would execute faster? That is, which one would complete the task in the shortest amount of time. **Please justify your answer in 4 or fewer sentences.**

Note: You do not need to calculate an exact number of instructions that are executed by each implementation to get full credit.

Question 8 {1 pt; all exam submissions receive this point}
Put anything you'd like the course staff to see here. This can be nothing, a piece of art, a message to members of the staff, anything you like! If you can't come up with anything, then a suggestion would be to write down your favorite aspect(s)/topic(s) of the course.