CS231 Digital Logic and Computer Architecture Lab Mid Sem

24-Sep-2024, 2.15pm-4.45pm, Max. Marks: 20

1 General instructions

- You must have logged onto a **lab machine**.
- You must have used the given **local login**, if not logout and login again. Specifically, you MUST NOT your CSE login.
- Use **ONLY** the suggested file names. Template files have already been created with the suggested file names.
- Within the given template, be sure to edit only the part you are allowed to edit.
- Apart from code, some questions require you to answer in text. In such cases, use the file **answers.txt**
- As in CS230 exams, you are allowed to carry any amount of your own **hand-written notes**.
- The required references are provided (they are also available via BodhiTree).
- You **cannot** carry any print-outs, xerox-copies, or books.
- Mobiles, laptops, pen-drives, memory cards, any such electronic devices are not allowed.
- If you carry your mobile to the lab, it must be **switched off** and kept on the table next to you for the invigilators to see.
- To ensure fairness for all, any **attempt to cheat**, successful or not, will be punished strictly.

Please see the next page

2 Submission Instructions

1. Relevant URL for announcements, references:
   `http://palantir.cse.iitb.ac.in/cs231/midsem.html`
2. Terminology:
   (a) **midsem-workdir:** If you are looking at this file, then you have already untarred your midsem tgz file and entered a directory called midsem-workdir. This will be your working directory.
   (b) **ROLLNUM:** This is your roll number – use UPPER CASE if your roll number has an alphabet
   (c) **ACCESS_PASSWORD:** This is the password you would have entered in your browser, to login and download your midsem from palantir.cse.iitb.ac.in
3. Where to work? You must do all your work in the midsem-workdir files only. E.g. you can invoke `code . &` from within midsem-workdir and edit the given template files. Remember to SAVE the files periodically in your editor.
4. How to submit?
   (a) Open a terminal in midsem-workdir
   (b) Export your access password as an environment variable. E.g. suppose the access password is 123456 , then type in the terminal: `export ACCESS_PASSWORD=123456`
   (c) Export also your roll number as an environment variable. E.g. suppose the roll number is 2345A67 , then type in the terminal: `export ROLLNUM=2345A67`
   (d) After the above export, in the same terminal, simply run: `bash autosubmit.sh`. This will submit your files automatically to palantir, every 5 minutes.
   (e) You can check the forupload.tgz and forupload directory to know what was submitted – it will be the latest copy of the relevant files in midsem-workdir.
   (f) You can also check the `upload` directory in palantir
   (g) Leave the autosubmit.sh script running throughout the exam, in this terminal
   (h) At the time of exam end, you simply have to ensure that you have saved your latest file, wait for the next submission, logout and leave after the instructor gives permission.
   (i) To submit immediately, you can just stop (hit Ctrl-C) and re-run autosubmit.sh

Please see the next page

3 Questions

1. (a) Read the above instructions
   (b) Read the above instructions again
   (c) Read the above instructions once again

Please see the next page

2. **File to use: recurse.s**

   Translate the given code in recurse.c to MIPS assembly code. Your translation must match the given C code in recurse.c . Specifically you MUST use recursion for the function 're-curse', a non-recursive solution will fetch you ZERO marks. The translation for the main() function in recurse.c is already given - *do not* change this part - write *only* the code for the recurse function, in recurse.s .

   You must follow the MIPS coding conventions in terms of caller and callee saved registers and other register usage, with the following EXCEPTION: for argument passing, you can use only register a0, and use the stack for any further arguments. Be sure to note how the recurse function is called from main().

   For the recurse function, you can define and use your own stack frame format. Specify your stack frame format in the answers.txt file.

   Marking scheme (your code must work at least for some scenarios for you to get marks for any of these parts):

   (a) **2 marks:** Commenting (appropriate level: not too much, not too little), indentation, appropriate label/file names. There **will be no partial marks** for this part: its either 0 or 2 marks.

   (b) **1 mark:** Works correctly for base cases in recursion.

   (c) **2 marks:** Works correctly for first recursive case.

   (d) **2 marks:** Works correctly for second recursive case.

   (e) **2 marks:** MIPS calling convention correctly followed in terms of caller/callee saved registers, and argument passing in a0 and stack as specified above. Note: you can get **negative marks** here, if you are way off the convention/specification.

   (f) **1 mark:** Specify your stack frame format, in the file answers.txt and your code must be consistent with the specified stack frame format

Next question in next page

3. **File to use:**

   (a) fifo.v, tb_fifo.v

   In this question, you have to implement a FIFO Queue. More precisely, you have to fill in the module provided in the template files. You also have to write your testbench using the provided testbench template file (tb_fifo.v).

   Hint 1:
   The code will be similar to the reg_with_enable module provided with the iterative Karatsuba multiplier code (we have also pasted that module in fifo.v). In short, you do not need to **explicitly** implement any state machine. Rather, you need to use the provided 2D-array to implement FIFO in a behavioural manner. You are free to use "+" for addition and "-" for subtraction. The FIFO is supposed to be synchronous, that is, it will change its state with the clock. It should also have a reset signal. However, do not try to reset the entire 2D-array. Only reset some of the internal registers used to maintain the FIFO status.

   Hint 2:

   (a) FIFO queue is maintained using two pointers ReadPointer and WritePointer. Both are initialized to 0. The ReadPointer is incremented upon a read, and the WritePointer is incremented upon a write.

   (b) You also need to maintain two flags full and empty to check if the FIFO buffer is full or empty. They are declared as output ports in the modules.

   (c) There are two input signals push and pop to insert and delete from the data structure. With push, you must also provide some data. The pop returns data according to the FIFO algorithm.

   Marks:

   (a) **2 marks:** Declare the pointer(s) with proper size and type.

   (b) **2 marks:** Implement a proper reset mechanism.

   (c) **6 marks:** Implement the logic for insertion and deletion.

========== THE END ==========