EEE-445 Computer Architecture I - Fall 2019

Project Assignment

Due on Sunday December 27th through ODTUCLASS electronic submission as a .zip file

Write a simple assembler using a high level programming language such as C or C++, Python, Java, or a scripting language such as Perl or JavaScript to convert any MIPS assembly program containing some of the main MIPS instructions to **hexadecimal** machine language or object code. A list of these instructions can be found in Fig. 2.27 (Page 139) of the textbook 5th Edition. Your assembler should also be able to handle the pseudo-instructions provided in the same figure. Assume the first line of the code is stored at MIPS memory location 0x80001000. The solution should support an ***interactive mode*** and a ***batch mode***. The interactive mode reads an instruction from command line, assembles it to hexadecial, and outputs the result to the screen. The batch mode reads a source file with extension .src, assembles to hexadecimal, and outputs the result to an object code file with extension .obj.
Note: You can verify your results using a MIPS simulator from a version of the textbook or one of the online MIPS simulator tools.

- Your code will be **evaluated based on**:
    - (30 pts.) Correctness – does it do the right thing?
    - (30 pts.) Readability – Did you pay attention to structure, modularity i.e. use of function calls for common tasks, proper commenting of the code?
    - (20 pts.) Flexibility – how easy is it to extend support to a new instruction added to the MIPS ISA? Are simple ***lookup tables*** utilized over hardcoded numbers or many conditional statements in the code?
    - (20 pts.) User friendliness – Can the program properly handle errors made by the assembly programmer? Does it give a good idea to the user on what to fix? Can the program handle labels in the program?

- **Submit** one report for the team, containing **ALL** of the following items **in a .zip file** for full credit:
    1. The name of your assembler
    2. Names and IDs of the team members
    3. One paragraph introduction to the problem.
    4. One paragraph on your assembler design choices, which includes comments on which language and compiler (if any) you used to implement and test your assembler (and why).
    5. One paragraph conclusion (did you achieve your goals, what have you learnt, etc.).
    6. Soft copy of your code with ***clear instructions*** in your report on how to run both interactive and batch modes.
    7. Soft copy of a file containing the assembly code in Fig. 2.27 (5th Edition, Page 139), and the object code file that corresponds to it, **obtained with your assembler in *batch* mode**.
    8. Screen capture soft copy of a session to show how your program handles the input instruction “addi $s1, $s1, -17” in *interactive* mode.