Assignment #6

In this project you will design, implement, and test a log file parser that parses raw command data produced by another program.

Raw command data have the following characteristics:

1. In the raw command log file:
    a. Address column indicates memory addresses in hex
    b. Data column contains the data in hex at the memory address
    c. Size column indicates whether data is 32-bit (D32) or 64-bit (D64)
    d. Cycle column indicates whether data is being written into or read from the memory address (Wr=Write, Rd=Read)
    e. RelTime column indicates how long the action in the previous row took (ex: the action in row 2 took 157.5ns, which is logged on row 3)
2. Address 0x40000810 contains the length of the S-to-D command in number of bytes
3. Address 0x40000818 to 0x40000C14 contains the data of the S-to-D command
4. Address 0x40000C18 contains the length of the D-to-S command in number of bytes
5. Address 0x40000C20 to 0x4000101C contains the data of the D-to-S command
6. A command can have anywhere between 1 to 42 words
7. A word is always 16-bit
8. The command words can have one of the following two memory layouts:
    a. Little endian: least significant word (word 0) at lowest address and most significant word (word N) at highest address
    b. Big endian: most significant word (word N) at lowest address and least significant word (word 0) at highest address
    c. To learn more about little vs big endian: https://en.wikipedia.org/wiki/Endianness
9. The data of S-to-D command use little endian word ordering
10. The data of D-to-S command use little endian word ordering
11. The bits of a word can have one of the following two bit numberings:
    a. LSB 0: least significant bit is bit 0 and most significant bit is bit 15
    b. MSB 0: most significant bit is bit 0 and least significant bit is bit 0
    c. To learn more about LSB vs MSB: https://en.wikipedia.org/wiki/Bit_numbering
12. All command words in the tables below use LSB 0 bit numbering scheme.
13. Word fields:

Word	Bits	Field Name	Description
0	14-13	Rec_Ctrl	0=no recording 2=no processing 3=processing & recording
1	15-13	Cmd_Type	4=Type A, 5=Type B, 6=Type C
4	0	Rec_Raw	0=disable, 1=enable
5	6-0	Cmd_ID	Command ID
10	15-11	Num_Responses	Number of responses

15	2	Reset_Enable	0=disable, 1=enable
22	3	Direction	0=Right, 1=Left
32	14-0	Num_Samples	Number of samples
37	15	Parity	0=even, 1=odd
38	14	Test	0=disable, 1=enable
40	7	Ctrl_Enable	0=disable, 1=enable
41	14-8	Code	Code value

Software Requirements:

1. The parser shall read a log file that contains raw command data and produce a text file with parsed command data
2. The parser shall parse rows with addresses 0x40000810, 0x40000818 to 0x40000C14, 0x40000C18, 0x40000C20 to 0x4000101C
3. For each command in the log file (as explained in the above section) the parser shall output the following:
    a. The line number (of the data in input log file), the type of cycle (i.e. “Read” or “Write”), the type of command (i.e. “S-to-D” or “D-to-S”) and number of words (1 to 42) on the same line. Examples:
        i. Line 496: Read S-to-D command: 16 words
        ii. Line 17698: Write D-to-S command: 42 words
    b. The parsed data of each word in the command, one field per line. For each field indicate the line number (of the data in input log file), the Word Number, the Field Name, the value of the field, and if available the meaning of the value. Examples:
        i. Line 707: Word 0: Rec_Ctrl = 3 (processing & recording)
        ii. Line 709: Word 4: Rec_Raw = 0 (disable)
    c. An example of one actual parsed command:
        i. Line 196: Write S-to-D command: 6 words
        ii. Line 197: Word 0: Rec_Ctrl = 0 (no recording)
        iii. Line 197: Word 1: Cmd_Type = 1 (unknown)
        iv. Line 199: Word 4: Rec_Raw = 0 (disable)
        v. Line 199: Word 5: Cmd_ID = 6
    d. There shall be one blank line between each command and no blank line within a command
    e. Print the words in a command in ascending order, i.e. word 0 before word 1 before word 2, etc.
    f. When command data in input log file is in reverse order, print the words in a command in descending order, i.e. word 41 before word 40 before word 39, etc.
    g. An example of one actual parsed command where the words are in reverse order:
        i. Line 695: Write S-to-D command: 12 words
        ii. Line 696: Word 10: Num_Responses = 0
        iii. Line 699: Word 5: Cmd_ID = 6

iv. Line 699: Word 4: Rec_Raw = 0 (disable)
v. Line 701: Word 1: Cmd_Type = 1 (unknown)
vi. Line 701: Word 0: Rec_Ctrl = 0 (no recording)

4. You shall calculate the data rate for each type of "Read S-to-D", "Read D-to-S", "Write S-to-D", "Write D-to-S". The data rates should be accurate to 2 decimal places and use Kilobits/sec or Megabits/sec or Gigabits/sec or Terabits/sec as appropriate. Print the data rates at the end of the output file. Example:
    a. Read S-to-D: 25.73 Kilobits/sec
    b. Read D-to-S: 53.01 Megabits/sec
    c. Write S-to-D: 154.24 Gigabits/sec
    d. Write S-to-D: 1.22 Terabits/sec
    e. Note that RelTime column indicates the amount of time the read/write of the PREVIOUS row took (note: ns = nanoseconds, us = microseconds, ms = milliseconds)
5. Run your program 3 times on the test_data.log input file during presentation in class to capture the 3 wall clock run times. The average wall clock run time must be 2.0 seconds or less.
    a. Your runs must complete successfully (i.e. did not crashed or otherwise aborted midway).

Additional Requirements:

1. You must work in teams of 4 persons/team. See team assignment sheet.
2. You must use Git for source control
    a. The finalized code must be on the master branch in Git
    b. I want to see multiple commits in Git made by everyone on the team.
    c. Everyone must contribute his/her fair share. If I see someone contributing significantly less than his/her team members I will reduce the score of that person at my discretion.
3. You must use GNU Make to build your source code
    a. Makefile must reside at the root directory of your source code
    b. The Makefile must have a default target of building the entire program
    c. The Makefile must have a clean target to clean out all intermediate build files in order to rebuild from scratch
4. You must use either C++ or Java to implement the software and you must use object-oriented programming (i.e. have classes and objects)
    a. If you use C++ you must use GNU g++ in make to compile
    b. If you use Java you must use Oracle javac in make to compile
5. You must use ONLY the functions and libraries in GCC 7.2 or Java SE 8. This is so that I don't have to hunt down and install extra libraries just to build your code.
6. Each team is required to present a 15 minute presentation of their work. Your presentation must include:
    a. Your technical design approach (i.e. how are you planning to solve this problem),
    b. Your management plan (i.e. who does what),
    c. Any obstacles (whether technical or procedural) that you must overcome,

d. Any interesting/unique features of your software, and
e. A demo run of your software.

Submission Requirements:

1. Submit the Git repository.
    a. Zip up the .git directory (and only that) and submit it on Titanium.
    b. Note: only one submission for the entire group is needed. If there are multiple submissions the last one will override the previous ones.
2. Submit a link to your GitHub repository.

Grading & Deadline

1. This project is worth 200 points.
    a. Using Git and Make and C++ or Java, and OOP gets 75 points
    b. Implementing all of the software requirements correctly gets 100 points
    c. The presentation is worth 25 points
    d. Note: if I cannot build or run your software, the maximum you will get is only 100 points!
2. I will do the following to build and run your code. Your code must conform to this:
    a. I will unzip the submitted .git directory
    b. I will check out the source code from the master branch in your Git repo
    c. I will run "make" (using default target) to build your source code
    d. I will run your program using the following command: "./<program name> <input file>"
    e. I will open your output file and compare it (using diff) with the expected results
3. Note: My machine is different from yours! To prevent unpleasant surprises send me your work-in-progress to make sure I can compile and run your code. (I just need to test that I can compile and run your code without crashing.)
4. Everyone on the team receives the same score except anyone I feel contributing significantly less than his/her teammates.
5. Deadline is Thursday Nov 30th, 2017 @6pm. Late submissions will not be accepted.
6. Your teams will give presentations on Thursday Nov 30th, 2017.
7. Presentation order will be randomly determined and will not be broadcast until day of presentation. Therefore you must be prepared to be the first team to present.

ANY QUESTIONS JUST ASK, DON'T ASSUME!