1 Assembler

The assembler makes use of the library string.h for string manipulation. In the first pass, the assembler records all the labels and .word instructions, each in their respective linked list.

In the second pass, it goes over every line of the assembly code, and translates it into machine code, differentiating between R-type and I-type instructions, and using the label addresses stored in the linked list of labels. The assembler generates a file memin.txt, into which the lines of machine code are written.

Once all of the R-type and I-type instructions have been executed, the assembler goes over the list of .word instructions, and executes them, meaning it writes the data from the instructions into their required locations in memin.txt.

The assembler can detect a few different errors, such as a missing input file or an invalid instruction format. In order to point out the line in the assembly file where an error occurred, the assembler stores an integer variable called "line_num", which gets updated with every new line in the assembly file.

The following files are included in the submitted folder for testing purposes:

- swap.asm - a simple reference code without errors.
- imm_error.asm - contains a line without the immediate value in an I-type instruction.
- label_error.asm - contains a typo in a label name.
- opcode_error.asm - contains a typo in an opcode.
- register_error.asm - contains a typo (missing '$') in a register name.

2 Simulator

In the simulator, the file memin.txt is parsed and a full scan gives us the instructions. The instructions are then executed one by one according to the program counter.

During execution, we output traces onto the corresponding file, and constantly check for errors, or the HALT instruction. Namely, several overflow and indexing errors.

After execution, we output all the files, memout.txt, regout.txt, etc. in the HEX format (as requested).

It is worthy to note that the registers and the memory are allocated on the stack (of the compiler), while the stack pointer memory ($sp) is allocated both on the stack and the heap, for the purpose of increasing performance.

The following libraries are used:

- stack.h - custom library used to define the struct of stack in order to simulate stack memory.
- fileParser.h - used to include helper functions for bit/string manipulation, and reading of files.
- math.h library, string.h library - used power (pow), strtol, and strtok to parse the files.

3 Fibo.asm

We manually load the values 0, 1, 1 into the memory. Then, we store the last 2 values (1, 1) onto the stack ($sp). Consequently, we call the fib label. There, we add up the values (using $t0, $t1 into $s0) on the stack and store them into the next memory address (using the incrementing $s1).

Then, the new value and the previous value (for $s0) are added onto the stack ($sp). And then, fib label is called again. It is called recursively until 1 of 2 things happen - we have a memory overflow error, or we have a number overflow error: the number exceeds 20 bits of length in the signed int representation. The latter will happen in this case, as the rate of growth of the Fibonacci sequence is very high.