LAB 2

HUMANOID COMPILER

Your Mission:

Complete and compile the given C code into LC-3 assembly language manually, and then assemble it into LC-3 object file.

Code Snippet:

```c
typedef int i16;
typedef unsigned int u16;

i16 func(i16 n, i16 a, i16 b, i16 c, i16 d, i16 e, i16 f){  //Lots of arguments, hah?
    i16 t = GETC() - '0' + a + b + c + d + e + f;
    if(n > 1){
        i16 x = func(n - 1, a, b, c, d, e, f);
        i16 y = func(n - 2, a ,b, c, d, e, f);
        return x + y + t - 1;
    }else{
        return t;
    }
}


i16 main(void){
    i16 n = GETC() - '0';
    return func(n, 0, 0, 0, 0, 0, 0);
}

_Noreturn void __start(){
    /*
        Here is where this program actually starts executing.
        Complete this function to do some initialization in your compiled assembly.
        TODO: Set up C runtime.
    */
    u16 __R0 = main(); //The return value of function main() should be moved to R0.

    HALT();
}
```

Details:

- The code must be translated AS IT IS. You might want to add some optimizations, but DO FOLLOW the C standard. In additional, optimizations must ONLY be done within basic blocks, which means you CANNOT change the control flow structure. For example, you cannot optimize the recursion into iteration, although the functionality is the same.
- Any usage of 3rd party C compiler is NOT allowed.
- __start() is actually not a function. The program starts from there, and then calls main function. At first the registers and memory are in random status. So, you should design and complete the

initialization process yourself. Notice that this process should be general, which means it must have nothing to do with the content of the main() or any other functions in this program.

- GETC() and HALT() are traps. Trap routines may alter the contents of registers and memory, thus disturbing your program. So, you may work out a standard for trap routines to follow. Notice that all callees in your program should also follow this standard.
- If another program wants to call func() in your program, it must have to follow certain rules. Design these rules as a standard. Also, all callers in your program should follow this standard.
- Your program will be RANDOMLY loaded to x3000 – xC000. Thus the .ORIG clause and the first two bytes of object file will be ignored.
- Your report should at least contain: 1. Initialization process; 2. Calling conventions; 3. Other standards you designed; 4. Error handling.

Scoring Criterion:

70% of score will depend on the correctness and design. 20% of score will depend on the report. The last 10% goes to the speed of the program.

"Speed" is calculated from the weighted average of userland instructions executed in several test cases.

Weights table:

Instruction	Weight
LDI/STI	8
LD/LDR/ST/STR	4
BR/JMP/JSR/JSRR/RET/TRAP	2
Other instructions	1

Additional Requirements:

WARNING: If you don't comply with these rules, the lab may be counted as an invalid work!

1. Save your report in pdf format and name it with your student number in uppercase (e.g. PB17001001.pdf).
2. You should have the object file named "program.obj", and the source code named "lab2.asm".
3. Put all above in a directory named after your student number in uppercase and pack it using TAR with GZIP compression (e.g. tar cvzf PB17001001_张三_Lab2.tar.gz PB17001001/).

Better Pigeon Than Cheat!!!

WARNING: If you are found cheated in this lab, Your FINAL SCORE will be affected ENORMOUSLY!

All programs will be checked by SSDEEP (a fuzzy hashing) and some will also be checked manually.