COMPSCI 210 Assignment 3

Due date: 21:00 23rd October 2017
Total marks: 100

This assignment aims to give you some experience with C programming.

Important Notes:

- There are subtle differences between various C compilers. We will use the GNU compiler gcc on login.cs.auckland.ac.nz for marking. Therefore, you MUST ensure that your submissions compile and run on login.cs.auckland.ac.nz. Submissions that fail to compile or run on login.cs.auckland.ac.nz will attract NO marks.
- Markers will compile your program using command "gcc –o name name.c" where name.c is the name of the source code of your program, e.g. part1.c. That is, the markers will NOT use any compiler switches to supress the warning messages.
- Markers will use instructions that are different from the examples given in the specifications when testing your programs.
- The files containing the examples can be downloaded from Canvas and unpacked on server with the command below:
    - `tar xvf A3examplefiles.tar.gz`
- As we need to return the assignment marks before the exam of this course, there is NO possibility to extend the deadline for this assignment.

Academic Honesty

Do NOT copy other people's code (this includes the code that you find on the Internet).

We will use Stanford's MOSS tool to check all submissions. The tool is very "smart". Changing the names of the variables and shuffling the statements around will not fool the tool. Last year, quite a few students were caught by the tool; and, they were dealt with according to the university's rules at https://www.auckland.ac.nz/en/about/learning-and-teaching/policies-guidelines-and-procedures/academic-integrity-info-for-students.html

In this assignment, you are required to write C programs to implement some of the functions of the LC-3 assembler. That is, the programs covert LC-3 assembly instructions to machine code.

Part 1 (24 marks)

In this part of the assignment, you are required to write a C program to translate LC-3's AND, ADD and HALT assembly language instructions into machine code. The detailed requirements are as below:

1. The assembly instructions are stored in a file. Each line of the file stores exactly one instruction. **The name of the file must be provided to the program as a command line argument**. The number of instructions in the file is **NOT** limited.
2. For this part, it should be assumed that the operands of the instructions only use the "register" addressing mode. That is, the values of all the operands are stored in registers.
3. It should be assumed that
    a. the file containing the assembly instructions starts with an ".orig" directive and ends with an ".end" directive
    b. the instructions are valid AND, ADD or HALT instructions
    c. there is exactly one space separating the opcode and the operands of the instruction
    d. the operands are separated by exactly one comma ","
    e. all the characters in the instruction are lower case letters
    f. there are no leading or trailing empty spaces in each line
    g. each line ends with the invisible "\n" character
4. The machine code should be displayed as a 4-digit hexadecimal number.
5. Name this program as part1.c

Here is an example of the execution of the program. In this example, the name of the file containing the instructions is t1.asm (NOTE: "t1.asm" is the exact name of the file). Markers might use a file with a different name when testing your program. The contents of t1.asm are:

```
.orig x3020
add r1,r2,r3
and r0,r4,r5
halt
.end
```

The execution of the program is shown below. The command line argument is marked in red.

```
$ ./part1 t1.asm
1283
5105
f025
```

Part 2 (21 marks)

Expand your program in Part 1 to allow the operand use the "immediate" addressing mode. That is, the value of an operand is stored in the instruction. It should be assumed that the value operand is given as a decimal number.
Name this program as part2.c

Here is an example of the execution of the program. In this example, the name of the file containing the instructions is t2.asm (NOTE: "t2.asm" is the exact name of the file). Markers might use a file with a different name when testing your program. The contents of t2.asm are:

```
.orig x3020
add r1,r2,r3
and r0,r4,r5
add r6,r7,#-9
and r0,r1,#15
halt
.end
```

The execution of the program is shown below. The command line argument is marked in red.

```
$ ./part2 t2.asm
1283
5105
1df7
506f
f025
```

Part 3 (14 marks)

Expand your program in Part 2 to allow instruction JMP to be converted.
Name this program as part3.c

Here is an example of the execution of the program. In this example, the name of the file containing the instructions is t3.asm (NOTE: "t3.asm" is the exact name of the file). Markers might use a file with a different name when testing your program. The contents of t3.asm are:

```
.orig x3020
add r1,r2,r3
and r0,r4,r5
jmp r5
add r6,r7,#-9
and r0,r1,#15
halt
.end
```

The execution of the program is shown below. The command line argument is marked in red.

```
$ ./part3 t3.asm
1283
5105
c140
1df7
506f
f025
```

Part 4 (18 marks)

Expand your program in Part 3 to allow the assembly program to contain directives for reserving memory locations and labels for marking memory locations.

1. Your program should generate a symbol table containing each label in the assembly program and the label's address.
2. The symbol table should be stored in a file named SymbolTable (**NOTE**: "SymbolTable" **is the exact name of the file. It does NOT have any suffix.**).
3. Each line in file SymbolTable contains exactly one label and its address.
4. A label and its address should be separated by exactly one empty space.
5. The order in which the labels appear in file SymbolTable is not important as long as all the labels are in the file.
6. Apart from creating file SymbolTable, your program does NOT need to display any output.
7. Name this program as part4.c

It should be assumed that:

- only ".fill" directive will be used for reserving memory locations
- a ".fill" directive is always preceded with a label
- each label consists of at most three characters
- there is exactly one space between a label and the ".fill" directive
- the ".fill" directives appear between the "HALT" instruction and the ".end" directive

Here is an example of the execution of the program. In this example, the name of the file containing the instructions is t4.asm (NOTE: "t4.asm" is the exact name of the file). Markers might use a file with a different name when testing your program. **In the file used by the markers, the value given in the ".orig" directive might be a value different from x3020.** The contents of t4.asm are:

```
.orig x3020
add r1,r2,r3
and r0,r4,r5
jmp r5
add r6,r7,#-9
and r0,r1,#15
halt
abc .fill #12
def .fill #-13
.end
```

The execution of the program is shown below. The command line argument is marked in red.

```
$ ./part4 t4.asm
```

After the execution, file `SymbolTable` is created and the contents of `SymbolTable` are:

```
abc 3026
def 3027
```

Part 5 (18 marks)

Expand your program in Part 4 to allow instruction LD to be converted.

1. Your program should display the machine code of the converted instructions.
2. Your program does NOT need to show the contents of the reserved memory locations.
3. Name this program as part5.c

Here is an example of the execution of the program. In this example, the name of the file containing the instructions is t5.asm (NOTE: "t5.asm" is the exact name of the file). Markers might use a file with a different name when testing your program. **In the file used by the markers, the value given in the ".orig" directive might be a value different from x3020.** The contents of t5.asm are:

```
.orig x3020
ld r2,abc
add r1,r2,r3
and r0,r4,r5
jmp r5
ld r6,def
add r6,r7,#-9
and r0,r1,#15
halt
abc .fill #12
def .fill #-13
.end
```

The execution of the program is shown below. The command line arguments are marked in red.

```
$ ./part5 t5.asm
2407
1283
5105
c140
2c04
1df7
506f
f025
```

5 marks for no compile warning messages when your programs are compiled.

Submission

1. You **MUST** thoroughly test your program on login.cs.auckland.ac.nz before submission. Programs that cannot be compiled or run on login.cs.auckland.ac.nz will **NOT** get any mark.
2. Use command "tar cvzf A3.tar.gz part1.c part2.c part3.c part4.c part5.c" to pack the five C programs to file A3.tar.gz.
    a. You **MUST** use the tar command on login.cs.auckland.ac.nz to pack the files as files packed using tools on PC cannot be unpacked on login.cs.auckland.ac.nz. You will NOT get any mark if your file cannot be unpacked on login.cs.auckland.ac.nz.
    b. You should submit your source code (i.e. the text files with ".c" suffix).
3. Submit A3.tar.gz using Canvas.
4. **NO** email submission will be accepted.

Debugging Tips

1. Debugging is a skill that you are expected to acquire. Once you start working, you are paid to write and debug programs. Nobody is going to help you with debugging. So, you should acquire the skill now. **You can only acquire it by practicing.**
2. If you get a "segmentation faults" while running a program, the best way to locate the statement that causes the bug is to insert "printf" into your program.
3. If you can see the output of the "printf" statement, it means the bug is caused by a statement that appears somewhere after the "printf" statement. In this case, you should move the "printf" statement forward. Repeat this process until you cannot see the output of the "printf" statement.
4. If you cannot see the output of the "printf" statement, it means the bug is caused by a statement that appears somewhere before the "printf" statement.
5. Combining step 3 and 4, you should be able to identify the statement that causes the "segmentation faults".
6. Once you identify the statement that causes the "segmentation faults", you can analyse the cause of bug, e.g. whether the variables have the expected values.

Suggestions

- You should create more test cases to test your program. The easiest way is to use LC-3 assembler and simulator.
    - If you use the LC3Edit program to create assembly programs on a PC, you need to be aware that each line ends with the invisible "\r\n" character sequence on a PC.
    - On a Unix machine, each line ends with an invisible "\n" character. The example assembly programs and the assembly programs used by the markers are created on a Unix machine.
- You can use a lot of C library functions. You might want to consider using the functions defined in "string.h" as it has a lot of functions for manipulating strings.

Further Work

For those who seek challenge, you can think about how to implement a full-fledged LC-3 assembler.