COMPSCI 210 Assignment 2

Due date: 09:59 21st October 2014
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 machine code that is different from the examples given in the specifications when testing your programs.

Disassembler is a very useful reverse engineering tool[1]. It converts machine instructions to human-readable assembly language equivalent. Writing a disassembler helps you know how a computer understands the meaning of a machine instruction. In this assignment, you are required to implement a disassembler for a subset of the LC-3 assembly language.

Part 1 (40 marks)

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

1. Each machine instruction is represented as a four digits hexadecimal number. All the instructions are stored in a file.
2. Each line of the file stores exactly one instruction. For example:
   ```
   5105
   0ffd
   ```
3. For this part, it should be assumed that the operands only use the "register" addressing mode. That is, the values of all the operands are stored in registers.

[1] Have you ever wondered how a hacker figures out the ways to circumvent the anti-piracy defence measure of software? Disassembler is one of the tools that helps a hacker to understand how a software works.

4. Your program should read each of the instructions in the file and convert each instruction to an LC-3 assembly language instruction. The results should be displayed on the screen.
5. The name of the file that contains the machine code instructions must be given as a command line argument.
6. Name this program as part1.c

An example is given below. In this example, the name of the file that contains the machine code instructions is “obj”. For this example, the contents of “obj” are:

```
1283
5105
```

The execution of the program is as below. The outputs of the program are marked in blue.

```
$ ./part1 obj
add r1,r2,r3
and r0,r4,r5
```

Part 2 (16 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. In the result, the value operand should be displayed as a decimal number.
Name this program as part2.c

An example is given below. In this example, the name of the file that contains the machine code instructions is “obj”. For this example, the contents of “obj” are:

```
1283
5105
1df7
506f
```

The execution of the program is as below. The outputs of the program are marked in blue.

```
$ ./part2 obj
add r1,r2,r3
and r0,r4,r5
add r6,r7,-9
and r0,r1,15
```

Part 3 (14 marks)

Expand your program in Part 2 to include instruction JMP.

Name this program as part3.c

An example is given below. In this example, the name of the file that contains the machine code instructions is “obj”. For this example, the contents of “obj” are:

```
1283
5105
1df7
506f
c080
```

The execution of the program is as below. The outputs of the program are marked in blue.

```
$ ./part3 obj
add r1,r2,r3
and r0,r4,r5
add r6,r7,-9
and r0,r1,15
jmp r2
```

Part 4 (25 marks)

1. Expand your program in Part 3 to include instruction BR.
2. The starting address of the program should be given as another command line argument that follows the name of the file containing the machine code. It should be assumed that the address is a 4-digit hexadecimal number.
3. The address of the instruction to be branched to should be displayed as a 4-digit hexadecimal number with prefix “0x”.
4. Name this program as part4.c

An example is given below. In this example, the name of the file that contains the machine code instructions is “obj”. For this example, the contents of “obj” are:

```
1283
5105
0ffd
0802
1df7
506f
c080
```

The execution of the program is as below. The outputs of the program are marked in blue.

```
$ ./part4 obj 3000
add r1,r2,r3
and r0,r4,r5
brnzp 0x3000
brn 0x3006
```

```
add r6,r7,-9
and r0,r1,15
jmp r2
```

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 A2.tar.gz part1.c part2.c part3.c part4.c" to pack the four C programs to file A2.tar.gz.
3. Submit A2.tar.gz using the assignment drop box.
4. **NO** email submission will be accepted.

Suggestions

- You should create more test cases to test your program. The easiest way is to use LC-3 assembler and simulator.
- 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 (no marks)

Implement a disassembler that is capable of decoding the full set of LC-3's instructions.