C Programming String Parsing and Translation

For this assignment, you will implement a simple C program that parses a restricted subset of MIPS assembly instructions and prints out information relevant to translating those instructions into machine code. In particular, you will be concerned with MIPS assembly instructions of one of the following forms:

```
R-format:  mnemonic   reg1, reg2, reg3
I-format:  mnemonic   reg1, reg2, immediate
           mnemonic   reg1, immediate
```

where `mnemonic` is one of the following MIPS mnemonics:

```
add  and  or  sub  addi  andi  lui  ori
```

and `reg1`, `reg2` and `reg3` are each one of the following MIPS registers:

```
$t0, $t1, $t2, $t3, $s0, $s1, $s2, $s3, $at
```

and `immediate` is an integer in the range -32768 to 32767. The elements of each instruction are separated by whitespace and commas, as shown. Whitespace can be any mixture of spaces and tab characters.

The instructions you are concerned with in this assignment can be classified as follows:

```
R-format:  add  and  or sub
I-format:  addi  andi  lui  ori
```

The `lui` instruction is special in that it follows the last form shown above, one register name and an immediate.

Given such a string if the instruction is R-format, your program must print out the correct function code for the mnemonic field, and the numeric values for the register numbers. If the instruction is I-format, your program must print out the correct opcode for the mnemonic field, the numeric values for the register numbers, and the numeric value of the immediate. In either case, you must print out an indication of the instruction's form (see output sample below).

Each of these values must be printed in *text binary* format (sequence of `'0'` and `'1'` characters). Opcodes and function codes must be printed as 6-bit values, register numbers as 5-bit values, and immediates as 16-bit 2's complement values.

For example, the input string: `add  $s3, $t1, $t0`

should yield the output: `R-format:  100000  10011  01001  01000`

and the input string: `addi  $t0, $s2, -42`

should yield the output: `I-format:  001000  01000  10010  1111111111010110`

On the other hand, the input string: `lui  $t3, 42`

should yield the output: `I-format:  001111  01011  0000000000101010`

The output values should be separated by whitespace.

It is possible that some input strings will violate the restrictions stated above, but only in the following ways;

- the mnemonic field may contain a string other than those listed above
- a register name field may contain a string other than those listed above

If that's the case, your program should print out `"Error: "` followed by the contents of the first field that violates the restrictions. In this situation, do not print out values for any of the other fields, even if those are valid.

Input and output files

Your program will read the input strings from a file named `MIPSInstructions.txt`. The input file will have one MIPS assembly instruction per line; lines will always be terminated by a newline character. Sample input files will be available on the course website; here's a short example:

```
add  $t0, $t1, $t2
or   $t3, $s1, $s1
addi $t0, $t1, 30352
ori  $t1, $s2, 8096
```

Your program will write its output to a file named `MIPSParse.txt`. You will write one line of output for each input line, as described earlier. Sample output files will be available on the course website; here's a short example corresponding to the sample input given above:

```
R-Format:  100000  01000  01001  01010
R-Format:  100101  01011  10001  10001
I-Format:  001000  01000  01001  0111011010010000
I-Format:  001101  01001  10010  0001111110100000
```

Suggested resources

Aside from a good C language reference (see the Resources page on the course website), the following sources of information should prove useful:

Computer Organization and Design: the Hardware/Software Interface
good overall description of MIPS32 architecture, register names/numbers, etc.
MIPS32 Architecture Volume 2: the MIPS32 Instruction Set (on the Resources page)
good details on specific MIPS32 assembly instructions and their representations
CS 2505 notes on C bitwise operations, especially slides 13 – 17 on printing the bits of a value

Memory management requirements

There is no particular reason for your program to use dynamic allocation for any strings, since there are reasonable, small limits on the lengths of all of the strings you need to process. That being said, if you choose to use dynamic allocation in your solution, you are required to properly deallocate each relevant block of memory as soon as possible after you are done using it. We may use valgrind to test this requirement, and penalize your score accordingly.

What to submit

You will submit an uncompressed tar file containing your complete C implementation (`.c` and `.h` files), and nothing else. This assignment will be graded automatically. Your submission will be compiled with a test driver:

```
gcc -std=c99 -m32 -Wall ...
```

It will be graded according to how many test cases your solution handles correctly. You will be allowed up to ten submissions for this assignment, so use them wisely. Test your functions thoroughly before submitting them. Make sure that your functions produce correct results for every logically valid test case you can think of.

The *Student Guide* and other pertinent information, such as the link to the proper submit page, can be found at:

http://www.cs.vt.edu/curator/

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the submitted file:

```
//    On my honor:
//
//    - I have not discussed the C language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used C language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any C language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    <Student Name>
```