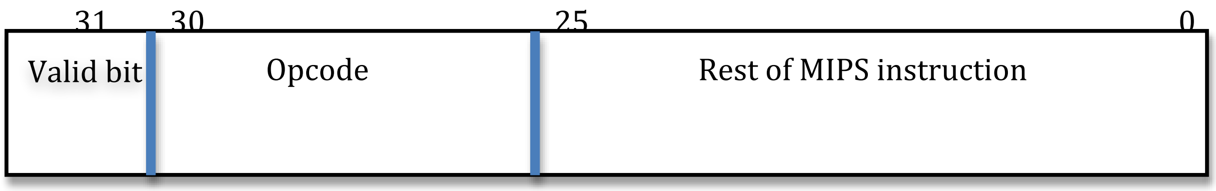
Project 1

Description: In this project, you will create a simple MIPS simulator. Your simulator will read a binary file containing a MIPS program and execute that program. This will occur in two steps. First, your program will generate the assembly code for the given MIPS program (dissassembler). Second, your program will create an instruction-by-instruction simulation of the MIPS program. This simulation will execute instructions sequentially (non-pipelined) and output the contents of all registers and memory (the state of the processor and memory) after each instruction. You will not have to implement exception/interrupt handling.

Implementation: You may implement this project in any programming language of your choosing. You MUST include instructions in a README file that indicate how to compile (if necessary) and run your program.

Details: Refer to the MIPS instruction set architecture PDF that is posted along with the assignment in TRACS. It provides the details for all MIPS instructions. NOTE that we are making the following changes to the instruction set architecture:

Instead of a 6 bit opcode, we will use a 5 bit opcode that is preceded by a valid bit . The valid bit will be set to 1 if the instruction is valid and should be executed. If the valid bit is set to 0, then the instruction has no effect. The opcodes will be the same as those in the MIPS instruction set, just ignore the most significant bit (the first bit). We will not change the functionality of any instruction, simply we use this convention for the opcode. The figure below illustrates this change



You will be given an input file containing a sequence of 32 bit instruction words. Assume that the first instruction is at memory address 96. The final instruction in an instruction sequence is ALWAYS a "BREAK" instruction. Following the break instruction is a sequence of 32 bit 2's compliment signed integers for the program data. These continue until the end of file.

Your simulator/dissassembler must support the following MIPS instructions:

J, JR, BEQ, BLTZ
ADD, ADDI, SUB
SW, LW
SLL, SRL
MUL,
AND, OR,
MOVZ
NOP

Your program must accept command line arguments for execution. The following arguments must be supported (Executable named “SIM”):

```
SIM –i INPUTFILENAME –o OUTPUTFILENAME
```

Your program will produce 2 output files. One named OUTPUTFILENAME_sim.txt, which contains the simulation output, and one named OUTPUTFILENAME_dis.txt, which contains the disassembled program code for the input MIPS program.

Your program will be graded both with the sample input and output provided to you, and with input and output that is not provided to you. It is recommended you construct your own input programs for testing.

Output: The dissembled output file should contain one line per word in the input file. It should be separated into 4 columns, each separated by tab character. The columns contain the following information:

1) The binary representation of the instruction word. If the word is an instruction (as opposed to memory data after the BREAK instruction), the instruction should be split into seven groups of digits: the valid bit, the opcode bits, four groups of 5 bits, and a final group of 6 bits.
2) The address of the memory location (in decimal)
3) The disassembled instruction opcode
4) If it is an instruction, print the operation, followed by a tab character, then print each argument separated by a comma and a space (“, “).

The simulation file have the following format:

- 20 equal signs and a newline
- cycle: [cycle number] [tab] [instruction address] [tab] [instruction string (same as step 4 above)]

- [blank lane]
- registers:
- r00: [tab] [integer value of R00][tab] [integer value of R01][tab] …[integer value of R07]
- r08: [tab] [integer value of R08][tab] [integer value of R09][tab] …[integer value of R15]
- r16: [tab] [integer value of R16][tab] [integer value of R17][tab] …[integer value of R23]
- r24: [tab] [integer value of R24][tab] [integer value of R25][tab] …[integer value of R31]
- [blank line]
- [data address]: [tab] [show 8 data words, with tabs in between]
- …[continue until last data word]

Instructions and arguments should be in capital letters. All integer values should be in decimal. Immediate values should be preceded by a # sign. Be careful and consider which instructions take signed values and which take unsigned values. Be sure to use the correct format depending on the context.

You output will be graded with the DIFF command. Test your output against the provided sample outputs! Any differences reported by the DIFF command are assumed to be incorrect output!

Sample files will be provided with the following extensions:

- .c – C code
- .mips – the compiled version of the C code
- .bin – the binary version of the .mips file
- sample output files
- a file named similar to sample_bin.txt which is a text version of the .bin file for your reference. Note that your program must accept the .bin file, not the text version.

What to turn in: Your source files. A README.txt file that contains the names and email addresses of your group members. It must also contain instructions for compiling and running your program. You may use any programming language and environment you like. If your program is runnable on Linux, I will just grade it, otherwise, you must demo it to me in my office. The executable must be named "mipssim" once the program is compiled. If your programming language uses an interpreter to execute the program, indicate that in the README.txt file. DO NOT turn in any sample input files or any previously generated ouput files.