5DV008 Computer Architecture Fall 2010
Obligatory Exercise 1
Due date: November 29, 2010 at 0800 (8am)

1 Overall Task And Goal

In the MIPS architecture, all machine instructions are represented as 32-bit numbers. For example, consider the 32-bit number `0x71014802`. To see whether it represents a valid instruction, first retrieve the `op` field, consisting of the leftmost six bits, by dividing by 2^{26}. The number obtained is `0x1c`, indicating that it is in a family of instructions which includes `madd`, `maddu`, `mul`, `msub`, `msubbu`, `clo`, and `clz`. Those instructions are in *R-format*, which means that the remaining 26 bits decompose into the fields `rs`, `rt`, `id`, `shamt`, and `funct`, of size 5, 5, 5, 5, and 6 bits, respectively. Computing these values yields `rs=8`, `rt=1`, `rd=9`, `shamt=0`, and `id=2`. (See page 97 of the textbook for the meaning of these abbreviations.) The *decomposed representation* of this instruction in hexadecimal format is thus `[0x1c 8 1 9 0 2]`. The `id` field indicates that it is a `mul` instruction; the *mnemonic representation* is `mul $t1, $t0, $at`.

Similarly, the 32-bit number `0x8c2a0000` represents the instruction `lw $t2, 0($at)`, which is an I-format instruction with `op=0x23`, `rs=1`, `rt=0xa`, and `imm=0`. The decomposed representation of this instruction is thus `[0x23 1 0xa 0]`.

Of course, not all 32-bit number represent instructions. For example, the number `0xffffffff` does not represent any instruction.

Figure B.10.2 on page B-50 of the textbook provides a concise representation of all such codes for the MIPS instruction set.

The overall task of this assignment is to write a program which determines the MIPS machine instruction which is associated with a given number.

2 Design and Implementation Requirements

2.1 The Base Assignment

The basic program should read a file with one 32-bit number per line, and generates a MIPS representation for that number. The following details apply.

b1. Each number in the input file may be in decimal or hexadecimal format. Both representations may occur in the same file.

b2. For each number in the input file, the disassembler produces one line of output, which contains:

b2.1 The number from the input file

b2.2 The format of the instruction (R, I, or J).

b2.3 The decomposed representation in decimal.

b2.4 The decomposed representation in hexadecimal.

b2.5 The representation of the instruction in mnemonic format, using register abbreviations wherever possible (*e.g.*, `$t0` instead of `$8`) and using decimal numbers whenever actual numbers are necessary.

If the number does not identify an instruction which is known to the program, a message to the effect `Instruction not known` should be printed after the number.

b3. For the basic assignment, all instructions of format R, I, and J which are found on pages B-51 to B-80 of the textbook. **except floating-point instructions**, must be supported. For the purposes of this assignment, the floating-point instructions are precisely those with the value `16`, `17`, or `18` (decimal) in the `op` field. An instruction is in R-format if its decomposition is in $(6,5,5,5,5,6)$-bit format, I-format if its decomposition is in $(6,5,5,16)$-bit format, and J-format if its decomposition is in $(6,26)$-bit format in its description on those pages of the textbook. Regard instructions with decomposition in $(6,5,5,10,6)$-bit format with the 10-bit entry equal to zero as being in R-format. The instructions `mult` and `div` are examples. In the output of the disassembler, such instructions should be represented in $(6,5,5,5,5,6)$-bit format. Note that there is a short list of *Exception and Interrupt Instructions* on page B-80, after the long list of floating-point instructions, and that of these, `nop` must be supported by the program.

b4. The output must be formatted in an æsthetically pleasing way, using tabs and column headers.

2.2 Development requirements

d1. The software may be written in any language, as long as the final product compiles, loads, and runs on the departmental Linux systems. Submissions will be evaluated on the local Linux systems, and submissions which require other systems for any of these steps will not be accepted. It is highly recommended that the software be written in *C*, *C++*, or *Java*, because it is unlikely that the course staff will be able to help you with problems in other languages. For *C* and *C++*, the *gcc* compiler must be used.

2.3 Grading and Extra Credit

The basic assignment is worth up to 50 points. Up to 25 extra points may be obtained for adding the following additional features.

xc1. 10 additional points may be obtained by supporting the remaining instruction, exclusive of pseudoinstructions and those with `16`, `17`, or `18` (decimal) in the `op` field. Examples which must be supported include `jr`, `mfhi`, `mthi`, and `syscall`. To obtain this credit, these instructions must be classified succinctly into new formats and each format group must be documented completely in the user manual. An implementation alone, without full supporting documentation, will not receive extra credit.

xc2. 15 additional points may be obtained by further classifying input numbers which "partially" but not totally correspond to valid instructions. There are at least two levels of "partial legality". First, the `op` field may be legal but the `funct` field does not provide a legal subvalue. For example, a number with `op=0x1c` but `funct=3` is not legal. Second, a field which must have a given constant value may have some other value. For example, a number with `op=0` and `funct=0x20` must have `shamt=0` to be legal. To obtain credit, the information displayed about the partial legality must be succinct yet informative. Furthermore, the class of instructions involved must be documented in the user manual.

xc3. No additional points will be awarded for supporting the floating-point instructions, as that would not add anything new conceptually to this project.

An effective way to check your solution is to feed the mnemonic representations to the SPIM assembler and see whether it produces the original numbers.

3 Submission Rules

The submission must also include a user manual which describes how to use the program and how to interpret the results. This manual will be worth 15 out of the 50 points for the base task.

A printed copy of the solution must be placed in the appropriate course mailbox on the fourth floor of MIT-huset. The user-id of each group member for the submission must be indicated clearly on a cover page of the printed submission. In addition, an electronic copy must be submitted to `labs-5dv008@cs.umu.se`. with the user-id of each group member given in the subject line of the message. A submission is not considered to be complete until both paper and electronic copies have been delivered.

4 Further Guidelines

g1. Solutions may be developed and submitted by groups of up to three individuals.

g2. Late solutions will receive $p\%$ of the quality points determined by the grader, where $p = 10 -$ number of working days or partial working days late.

g3. Students who have already completed the "laboratory" part of the course are permitted to submit this exercise for points only. Please inform the grader if you submit something for points only.