Summer 2015

ECE 3056

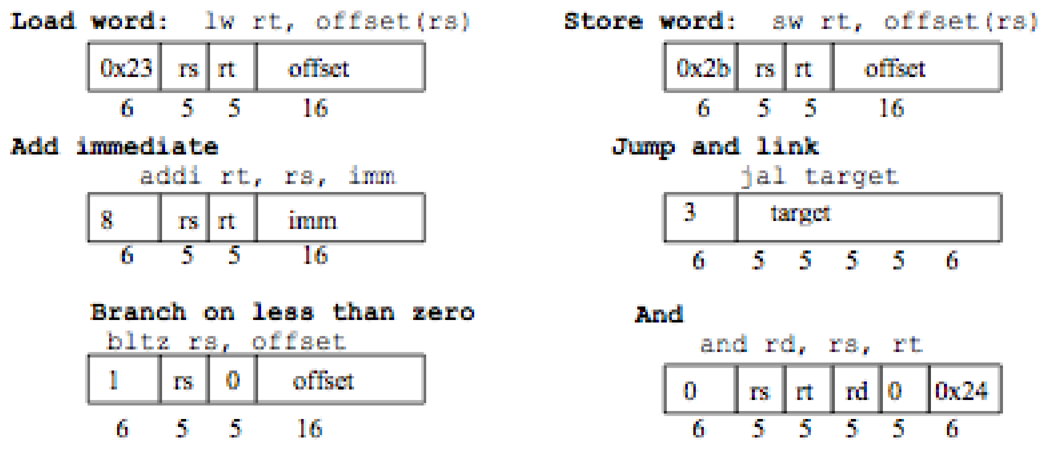
Architecture, Concurrency and Energy in Computation

Project 1

MIPS Assembly Language Disassembler

In this assignment, you will implement a simple disassembler program using MIPS assembly language for the Mini-MIPS ISA defined below. You will use the MIPS functional simulator for validating the functionality of your MIPS disassembler program. You can find useful information on SPIM here. Your program should disassemble only the six instructions defined in the mini-MIPS ISA from the binary words provided. All other words not belonging to any of the six instructions should print out "unsupported opcode !!" First, download the projskel.s skeleton file which is attached to this assignment. This skeleton file contains 14 instructions defined after the label "binary_code" in hexadecimal format. Each instruction is encoded by each .word using Little Endian byte-order. Your MIPS program should read one .word at a time, decode the .word binary to determine the instruction, and then print out each instruction on the Console window. Note that some instructions in the skeleton program are not defined by the Mini-MIPS ISA. During disassembly, your program should print out "Unsupported opcode !!" on the console window when encountering such instructions . Be sure to note that the Offset field encodes a signed number, be cautious when you print them out. For jump target, assume that the four msbs of the PC are 0000.

Mini-MIPS ISA (contains only 6 instructions, you can find the same format of these 6 instructions in Appendix A of H&P)



Printout example on the SPIM console window:

```
lw    $4  24($10)
addi  $5  $6  -7
unsupported opcode !
and   $2  $3  $4
unsupported opcode !
jal   120000
...
```

Hint. In the skeleton file, a few ASCII (.asciiz) symbols and instruction mnemonics (e.g. lw) are defined. You can use them to print out some of the needed instruction symbols in the Console window. There are many repetitive operations to carry out in your program (e.g. print dollar sign ($) before print the register number). In order to reduce the code size, you are encouraged to use procedure/function call, e.g. jal instruction for call and its corresponding jr $31 for return in MIPS, to handle these repetitive operations. If you are careful in how you allocate your registers, you will not need to use the stack to store the procedure's local variables etc. An example function might extract an unsigned 5-bit number from a register, given a bit offset into the register.

SPIM Simulator Warmup. The SPIM simulator will be used for executing your MIPS program. For those who use PCSpim on Windows, uncheck the "load trap file" under Simulator→Settings since our skeleton file already contains _start initialization code. If you've never used any editor such as vi, emacs, MS visual IDE to write programs before, a free syntax-coloring editor with MIPS syntax coloring files is available at www.crimsoneditor.com.

Signoff. You must go to the lab hours held by TA for having your assignment checked off before the due time (last scheduled hours) or the first hours scheduled after the due date. In either case, the code must be submitted on T-square on time (Friday 5/29 before 11:55pm). Also, please turn in your code on the assignment page. Contact Taimour (our TA) for alternative appointments for check-off. A rubric defining how points are allocated is attached. Also note that, Taimour might use a different input set to perform further tests.