The UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Comp 411 Computer Organization
Spring 2014

Problem Set #3
Issued Thursday, 2/13/14; Due Thursday, 2/20/14 (hardcopy, beginning of lab)

Feel free to get help from others, but the work you hand in must be your own. Please **enter your answers in the space provided.** Also, note that **once solutions have been handed out, late work may not be accepted.**

Problem 1: Converting Instructions to Assembly Language (35 points)
The conversion of a mnemonic instruction to its binary representation is called assembly. This tedious process is generally delegated to a computer program for a variety of reasons. In the following exercises, you will get a taste of what the task of translating from assembly to machine language is like.

a) Match the instructions below with their hexadecimal counterparts. Enter your answer as the letter 'A'-'R' in the blank space to the left of each instruction. NOTE: *Do not use the MARS assembler to do this exercise. Using MARS will not only deprive you of some hand-coding practice, but may also give you unexpected answers because it has a tendency to modify/rewrite some of these instructions with its preferred coding templates. However, once you have completed this exercise by hand, feel free to experiment with MARS.*

J	`addi $s0, $zero, 0x5B03`	**A:** 0x8C49000A
E	`slti $a1, $a2, -1`	**B:** 0x3C16FFFF
L	`addi $t2, $s0, 8`	**C:** 0x000C59C0
H	`loop: beq $s1, $s2, loop`	**D:** 0x34E60009
K	`loop2: bne $s2, $s1, loop2`	**E:** 0x28C5FFFF
M	`sll $t3, $t4, 7`	**G:** 0x30A64569
O	`and $17, $18, $19` think this should start with 023	**H:** 0x1230FFFF
D	`ori $a2, $a3, 9`	**J:** 0x20105B03
A	`lw $t1, 0xA($v0)`	**K:** 0x1651FFFF
N	`add $s1, $s2, $s3`	**L:** 0x220A0008
J	`andi $a2, $a1, 0x4569`	**M:** 0x01232020
P	`lui $t1, 0x4569`	**N:** 0x02538820
B	`lui $s6, 0xFFFF`	**O:** 0x02538824
C	`add $a0, $t1, $v1`	**P:** 0x3C094569

Problem 1: Converting Instructions to Assembly Language (continued)

b) The "no-op" pseudoinstruction (it is not a real instruction) consists of all zeroes: 0x00000000. Write the actual instruction that this corresponds to (including register operands).

Answer:

ssl $r0, $r0, 0

Problem 2: Converting pseudo-instructions (35 points)

MIPS assembly language provides opcode mnemonics for instructions that are not part of the instruction set architecture. These pseudoinstructions can be generated using a sequence of one or more "true" MIPS instructions.

Find a "true-instruction" equivalent for each of the following pseudo-instructions (some are official MIPS pseudoinstructions, others are made up). Each of these can be implemented using only one real MIPS instruction.

a) `move rA, rB`

Reg[rA] ← Reg[rB]
Move register rB to rA

Answer:

add $rA, $rB, $zero

b) `not rA, rB`

Reg[rA] ← ~Reg[rB]
Put the bitwise complement of rB into rA

Answer:

nor $rA, $rB, $zero

c) `neg rA, rB`

unsure

Reg[rA] ← -Reg[rB]
Put the 2's complement of rB into rA

Answer:

sub $rA, $rB, $zero

d) `dec rA`

unsure

Reg[rA] ← Reg[rA] - 1
Decrement rA by 1 and place result in rA

Answer:

addi $rA, $rA, 1111

e) Suppose we wanted to fill a register rA with the value 65535 (0x0000FFFF). Would the instruction `ori rA,$0,0xFFFF` perform that action? If not, what would be the value in rA?

unsure

Yes, it would perform that action.

f) Suppose we wanted to fill a register rA with the value 4095 (0x00000FFF). Would the instruction `addi rA,$0,4095` perform that operation? If not, what would be the value in rA?

Yes, it would perform that action.

g) Suppose we wanted to fill a register rA with the value -1 (0xFFFFFFFF). Would the instruction `ori rA,$0,-1` perform that operation? If not, what would be the value in rA?

Yes, it would perform that action.

Problem 3. "Loading up at the Store" (30 points)

The MIPS ISA provides access to memory exclusively through load (lw) and store (sw) instructions. Both instructions are encoded using the I-format, thus providing three operands: two registers and a 16-bit sign- extended constant. The memory address is computed by adding the contents of the register specified in the rs register field to the *sign-extended 16-bit constant*. Then either the contents of the specified memory location are loaded in the register specified in rt instruction field (lw), or that register's contents are stored in the indicated memory location (sw).

a) How many distinct memory locations can be accessed by changing the immediate field if the value in register `rs` is fixed at, say, 2^{15}? (*Note*: addresses generated by `lw`/`sw` instructions *must* be multiples of 4, i.e., they must be *word* addresses.)

Answer:

unsure

(2^(16))/4 = 16384

b) MIPS assemblers provide an instruction for loading an effective address from memory into a register, called "`la`" for load address. The syntax of this pseudoinstruction matches the `lw` instruction, and an example is shown below:

```
la $t0, 100($t1)
```

The result of the above instruction is to store the result of the address calculation into register $t0. The memory is actually not read. Give one true instruction (with operands) that can be used to implement this pseudoinstruction.

Answer:

unsure

addi $t1, $zero 0x76543210
lb $t0,

c) MIPS does not provide any instruction for specifying a memory address with a variable offset from `rs` (i.e., allows only an immediate constant to be specified as the offset). Fill in the multiple-instruction sequence below to accomplish this type of memory access using available MIPS instructions. Assume the base array's base address (i.e., the location of its 0^{th} member) is in register $t0, the ***word*** index is located in $t1, and the value in memory is being loaded into $t2.

Thus, we effectively want to execute an instruction that would look like (but does not exist):

```
lw $t2, $t0(4*$t1)
```

However, the MIPS instruction set does not provide any such instruction. Your task is to use a sequence of actual MIPS instructions to implement the same behavior.

unsure