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

Comp 411 Computer Organization
Fall 2012

Problem Set #2
Issued Mon. 10/1/12; Due Mon. 10/8/12 (hardcopy, beginning of class)

Homework Information: Late homework will not be accepted. Feel free to get help from others, but the work you hand in should be your own. You may **not** use solutions to a previous year's homework to aid you in completing this assignment. Please **enter your answers in the space provided.**

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.*

___ addi $v0,$zero,0x8009	**A:** 0x000A4A40
___ slti $t2,$t1,-1	**B:** 0x21490009
___ addi $t1,$t2,9	**C:** 0x014B4820
___ loop: bne $t1,$t2,loop	**D:** 0x014B4824
___ loop2: beq $t2,$t1,loop2	**E:** 0x000A4A43
___ sll $t1,$t2,9	**G:** 0x3C094809
___ and $9,$10,$11	**H:** 0x152AFFFF
___ ori $t1,$t2,9	**J:** 0x292AFFFF
___ lw $a1,0x09($v0)	**K:** 0x312A4809
___ add $t1,$t2,$t3	**L:** 0x3C03FFFF
___ andi $t2,$t1,0x4809	**M:** 0x20028009
___ lui $t1,0x4809	**N:** 0x020A4820
___ lui $v1,0xFFFF	**O:** 0x8C450009
___ sra $t1,$t2,9	**P:** 0x1149FFFF
___ add $t1,$s0,$t2	**R:** 0x35490009

Problem 1: Converting Instructions to Assembly Language (continued)

b) The "no-op" instruction consists of all zeroes: 0x00000000. Write the actual instruction that this corresponds to (including registers).

Answer:

Problem 2: Converting pseudo-instructions (35 points)

MIPS assembly language provides opcode mnemonics for instructions that are not part of the instruction set architecture. For the most part, 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:

b) `not rA, rB`
Reg[rA] ← ~Reg[rB]
Put the bitwise complement of rB into rA

Answer:

c) `neg rA, rB`
Reg[rA] ← -Reg[rB]
Put the 2's complement of rB into rA

Answer:

d) `dec rA`
Reg[rA] ← Reg[rA] - 1
Decrement rA by 1 and place result in rA

Answer:

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

f) Suppose we wanted to fill a register rA with the value 255 (0x000000FF). Would the instruction `ori rA,$0,255` perform that operation? If not, what would be the value in rA?

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?

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).

It is possible to “directly” address a limited range of 32-bit memory locations by encoding the `rs` field as $0.

Answer:

a) How many distinct memory locations can be addressed if `rs` is $0? (*Note:* Addresses generated by lw/sw instructions *must* be multiples of 4, i.e., they must be *word* addresses.)

b) MIPS assemblers often provide a pseudoinstruction (see problem 2) for loading an effective address 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)
```

Give one true instruction (with operands) that can be used to implement this pseudoinstruction:

Answer:

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 array’s base address (i.e., the location of its 0th 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. Indicate your answer in the table below.

(Hint: it may be helpful to come up with your own solution, then fill in the code below)

?	**$t3**	**$t1**	**2**
ADD	**$t4**	**$t3**	?
LW	**$t2**	?(?)	