CSE 410 - Spring 2004

Homework 1

due on Wednesday, April 7 at 9:30 AM, the beginning of class

42 points + 2 extra credit

Name	Solution
Student #	DWJ 8-Apr-04

This homework is intended to help you get comfortable dealing with instructions and data at the machine level: individual MIPS instructions and the raw bits of numeric representations. Feel free to use the SPIM simulator to try out the code that you write for these problems.

You can discuss these problems with others to get a better understanding of the material. However, do not just copy the answers from someone else – you won't learn anything and you will be cheating yourself and your classmates. Ask questions instead!

1. (2pt) An example of an R-format instruction is Shift Left Logical: `sll rd,rt,shamt`. The `shamt` field in an R-format instruction is 5 bits wide. The number that appears in the `shamt` field is used to control how far to shift (number of bit positions). Given that the value is interpreted as an unsigned integer, what is the maximum shift distance that can be specified by the `shamt` field? Why?

   31 bits, because $2^5 - 1 = 31$ is the largest number that will fit in 5 bits.

2. An example of an I-format instruction is Load Word: `lw rt,c(rx)`. This instruction loads register `rt` with the value from the word at location `c(rx)`, using the sum of the immediate `c` and the register `rx` as the byte address. The offset `c` is stored in a 16-bit field.

a. (2pt) Assuming that one bit of the offset field is a sign bit, how many bytes can be addressed for any single value of `rx`? Why?

→ $2^{15} = 32768$. $2^{16} = 65536$. If you use a sign bit and allow ± 0 then you get $32768_{+\text{addresses}} + 32768_{-\text{addresses}} - 1_{\text{dup}} = 65535$.

or

→ If you use 2's complement negation you get 65536 addresses.

b. (2pt) If the field were treated as an unsigned 16-bit integer instead of a signed integer as in part (a), how many bytes could be addressed for any single value of `rx`? Why?

$2^{16} = 65536$

3. MIPS instructions are 4 bytes long. All instructions are word-aligned in memory. Every byte in memory has a unique address. The first instruction in a program is often put at location 0x400000. ("0x" means that the following number is in base 16, or hexadecimal.)

a. (2pt) What is the address of the second instruction in the program? Give the address in binary (base 2) notation.

$0100 \quad 0000 \quad 0000 \quad 0000 \quad 0000 \quad 0100_2$

$= 4 \quad 0 \quad 0 \quad 0 \quad 0 \quad 4_{16}$

b. (2pt) What is the address of the fourth instruction in the program? Give the address in hexadecimal (base 16) notation.

0x 40 000C

0
4
8
C

c. (2pt) Assume that this is a very large program with several thousand instructions. What are the two low order binary digits of the byte address of the 2019th instruction? (You do not need to convert 2019 to binary in order to answer this question!)

00_2

4. A nibble is generally considered to be 4 bits (half of a byte). Although larger machines like desktop processors and mainframes usually don't operate on individual nibbles, it is common for small embedded systems to use them.

a. (2pt) What is the largest unsigned integer value that can be stored in a nibble? Give your answer in hexadecimal (base 16) notation.

F_{16}

b. (2pt) The value 1000_2 fits exactly into one nibble. If it is copied from there to a one byte register (ie, a register that is 8 bits wide) and is then shifted to the left by four bits, what is the resulting value in the register? Assume that there is no sign extension during the copy. Give your answer in decimal (base 10) notation.

$1000\ 0000_2 = 2^7 = 128_{10}$

or $1000_2 = 8_{10}$ and $8 * 16 = 128_{10}$

Note: The sign extension issue doesn't really matter in this problem because all the high order bits get shifted out during the shift left operation

5. Hand coding assembly language control flow is similar to programming in higher level languages like C or Java. However, the test conditions are a little more difficult to write and the logic often needs to be inverted. In the space below, write the code to implement the given expressions. You can use any of the $tn registers as required for temporary storage. Comment your code so that we can follow your logic.

a. (5pt) Assume that there is some value in register `$t0`, and some value in memory location `limit`. Show any additional labels that you use in implementing this.

Implement:
```
if ($t0 > limit)
    $t0 = limit;
```

```
        lw    $t1, limit          # get limit value
        ble   $t0, $t1, skip      # skip if t0 <= limit
        move  $t0, $t1            # t0 = limit
skip:
```

b. (5pt) Assume that there is a signed integer value greater than or equal to 1 in memory location `limit`. Calculate the sum of the numbers 1..`limit` and accumulate it in register `$t1`.

Implement:
```
$t1 = 0;
for ($t0=1; $t0<=limit; t0++)
    $t1 = $t1 + $t0;
```

```
      move   $t1, $zero          # sum = 0
      li     $t0, 1              # index = 1
      lw     $t2, limit          # upper limit

loop:
      add    $t1, $t1, $t0       # accumulate
      addi   $t0, 1              # increment index
      ble    $t0, $t2, loop      # repeat
```

6. For this problem, you will need the example program sum.s and the SPIM simulator. Start SPIM running and load the sum.s program. Run the program. You should get a console display window that shows the number of elements in the array and their sum.

a. (2pt) The first display pane in the SPIM window shows all the registers in the machine. Notice that both the name and number are shown for each register, along with the current value. What is the value shown in register $t0 after the program executes? Referring back to the sum.s source code, what is the meaning of this value?

$40_{16} = 64_{10}$ $to is the sum register

b. (2pt) What is the value shown in register $t3 after the program executes? Referring back to the sum.s source code, what is the meaning of this value?

F_{16} last value in the array

c. (2pt) The second display pane shows the program code stored in the text segment. What is the address of the first instruction in the text segment?

0x 0040 0000

d. (2pt) The first little bit of code is supplied as part of SPIM. Our program starts at location 0x00400024. You can see that the source line is number 17, and the original source code is shown over on the right hand side. Scroll down to where line 37 is shown. What is the hexadecimal value of the instruction word associated with this line?

0x 112a 0005

e. (2pt) What is the value of the least significant 4 hexadecimal digits of the instruction word in (d)? How is this value used when the instruction is executed?

0005 It is the offset in words to the skip label. If the branch is taken, then the offset is used to calculate the location of the next instruction.

Note that the actual value of an ~~the~~ offset like this is always calculated by the assembler, not the programmer.

f. (2pt) The third display pane shows the program data stored in the data segment. Scrolling through the data in this section, you should be able to identify the val array from the sum.s program. At what address does the val array start?

0x 1001 0040

g. (2pt) Given what you can see about the val array in the data display pane and referring back to the sum.s source code, what would you expect the address of label endtag to be?

0x 1001 0060

h. (2pt) Referring back to the sum.s source code, identify the two registers that should contain the value given in (g) when the program finishes executing. What are they? Verify that the register display agrees with your answer.

$t1 , $t2

7. (Extra credit, 2 points) Assume that memory location `str` is the first byte of an asciiz text string (ie, it is a string of character bytes followed by a byte with the value 0.) Assume that memory location `histo` is the beginning of a block of 256 words. Assume that the `histo` block has already been initialized to 0 in each word. Look at each character in the string and increment the appropriate word in the `histo` array to record how many times each character occurs in the text string. Note that this code records the 0 character that terminates the string. The character codes for A to Z are 65_{10} through 90_{10}, and the character codes for a to z are 97_{10} through 122_{10}.

Implement:

```
idx = 0;
do {
      c = str[idx];
      histo[c]++;
      idx++;
} while (c!=0)
```

Enter your code into the skeleton histo.s and check that it works. Submit the source file electronically.

See attached.

```
# This program does a histogram on the character string str.
# This file was written in the Context editor with smart tabs
# turned off (Options -> Environment -> Editor -> Smart Tabs).

        .data   # The following entries go in the data segment
strSpace:
        .asciiz  " "
strNewline:
        .asciiz  "\n"
str:
        .asciiz "AACCEEGGIIKKMMOOQQSSUUWWYYabcdefghijklmnopqrstuvwxyz"
        .align  2                   # ensure table is word aligned
histo:
        .space  1024                # 256 words

        .text   # The following entries go in the text (program code) segment
main:
        li      $v0,4               # load print_string code
        la      $a0,str             # load address of the string to print
        syscall                     # ask the system to print the string

        li      $v0,4               # load print_string code
        la      $a0,strNewline      # load address of the string to print
        syscall                     # ask the system to print the string

        # zero out the histogram block

        la      $t0,histo           # start address
        addi    $t1,$t0,1024        # end address
initLoop:
        sw      $zero,0($t0)        # store zero
        addi    $t0,$t0,4           # increment address pointer
        blt     $t0,$t1,initLoop        # loop if more to do

        # implement problem 7 code here

        move    $t0,$zero           # $t0 = character index = 0
        la      $t1,str             # $t1 = address(str)
        la      $t2,histo           # $t2 = address(histo)

histoLoop:
        add     $t3,$t1,$t0         # address of next character
        lbu     $t4,0($t3)          # $t4 = next character
        sll     $t4,$t4,2           # convert to word offset
        add     $t5,$t2,$t4         # address of histo entry
        lw      $t6,0($t5)          # load the histogram entry
        addi    $t6,1               # increment it
        sw      $t6,0($t5)          # store it back in memory
        addi    $t0,$t0,1           # increment the character index
        bnez    $t4,histoLoop       # loop if not at end

        # print the contents of the histogram

        la      $t0,histo           # $t0 = address of the first word of the histo
array
        addi    $t1,$t0,1024        # $t1 = address of word following the histo array
        move    $t2,$zero           # $t2 = index = 0

printLoop:
        li      $v0,1               # load print_integer code
        move    $a0,$t2             # put the index in $a0 for printing
        syscall                     # ask the system to print the integer

        li      $v0,4               # load print_string code
```

```
la      $a0,strSpace     # load address of the string to print
syscall                  # ask the system to print the string

li      $v0,1            # load print_integer code
lw      $a0,0($t0)       # put the histogram value in $a0 for printing
syscall                  # ask the system to print the integer

li      $v0,4            # load print_string code
la      $a0,strNewline   # load address of the string to print
syscall                  # ask the system to print the string

addi    $t0,$t0,4        # $t0 = address(next entry)
addi    $t2,$t2,1        # $t2 = index(next entry)
bne     $t0,$t1,printLoop # loop if more to do

jr      $ra              # return
```