COE 308 – Computer Architecture

Assignment 2: MIPS Instructions and Assembly Language

Solution

1. (2 pts) Bits have no inherent meaning. Given the 32-bit pattern:

`1010 1101 0001 0000 0000 0000 0000 0010`

What does it represent, assuming it is …

a) A 2's complement signed integer?
b) A MIPS instruction?

Solution:

a) **`-1,391,460,350`**

b) **Op = 101011_2 = 0x2b = sw - store word (I-Type format)**

rs = 01000_2 = r8 = $t0

rt = 10000_2 = r16 = $s0

immediate16 = $0000\ 0000\ 0000\ 0010_2$ = 2

MIPS instruction = sw $s0, 2($t0)

2. (2 pts) Find the shortest sequence of MIPS instructions to:

a) Determine if there is a carry out from the addition of two registers **$t3** and **$t4**. Place the carry out (**0** or **1**) in register **$t2**. It can be done in two instructions.

b) Determine the absolute value of a signed integer. Show the implementation of the following pseudo-instruction using three real instructions:

```
abs  $t1, $t2
```

Solution:

a)
```
addu $t5, $t3, $t4
sltu $t2, $t5, $t3  # there is carry if sum < any operand
```

b)
```
addu $t1, $t2, $zero
bgez $t2, next
subu $t1, $zero, $t2
next:
```

3. (4 pts) For each pseudo-instruction in the following table, produce a minimal sequence of actual MIPS instructions to accomplish the same thing. You may use the **`$at`** for some of the sequences. In the following table, **`imm32`** refers to a 32-bit constant.

Pseudo-instruction	Solution
`move  $t1, $t2`	**`addu  $t1, $t2, $zero`**
`clear $t5`	**`addu  $t5, $zero, $zero`**
`li    $t5, imm32`	**`lui  $t5, upper16`** **`ori  $t5, $t5, lower16`**
`addi  $t5, $t3, imm32`	**`lui  $at, upper16`** **`ori  $at, $at, lower16`** **`add  $t5, $t3, $at`**
`beq   $t5, imm32, Label`	**`lui  $at, upper16`** **`ori  $at, $at, lower16`** **`beq  $t5, $at, Label`**
`ble   $t5, $t3, Label`	**`slt  $at, $t3, $t5`** **`beq  $at, $zero, Label`**
`bgt   $t5, $t3, Label`	**`slt  $at, $t3, $t5`** **`bne  $at, $zero, Label`**
`bge   $t5, $t3, Label`	**`slt  $at, $t5, $t3`** **`beq  $at, $zero, Label`**

4. (2 pts) Translate the following statements into MIPS assembly language. Assume that *a*, *b*, *c*, and *d* are allocated in \$s0, \$s1, \$s2, and \$s3. All values are signed 32-bit integers.

a) **`if ((a > b) || (b > c)) {d = 1;}`**

Solution:

```
   bgt  $s0, $s1, L1
   ble  $s1, $s2, next
L1:
   ori  $s3, $zero, 1
next:
```

b) **`if ((a <= b) && (b > c)) {d = 1;}`**

Solution:

```
   bgt  $s0, $s1, next
   ble  $s1, $s2, next
   ori  $s3, $zero, 1
next:
```

5. (3 pts) Consider the following fragment of C code:

```
for (i=0; i<=100; i=i+1) { a[i] = b[i] + c; }
```

Assume that `a` and `b` are arrays of words and the base address of `a` is in `$a0` and the base address of `b` is in `$a1`. Register `$t0` is associated with variable `i` and register `$s0` with `c`. Write the code in MIPS.

Solution:

```
        addu  $t0, $zero, $zero       # i = 0
        addu  $t1, $a0, $zero         # $t1  = address a[i]
        addu  $t2, $a1, $zero         # $t2  = address b[i]
        addiu $t3, $zero, 101         # $t3  = 101 (max i)
loop:   lw    $t4, 0($t2)             # $t4  = b[i]
        addu  $t5, $t4, $s0           # $t5  = b[i] + c
        sw    $t5, 0($t1)             # a[i] = b[i] + c
        addiu $t0, $t0, 1             # i++
        addiu $t1, $t1, 4             # address of next a[i]
        addiu $t2, $t2, 4             # address of next b[i]
        bne   $t0, $t3, loop          # exit if (i == 101)
```

6. (3 pts) Add comments to the following MIPS code and describe in one sentence what it computes. Assume that `$a0` is used for the input and initially contains `n`, a positive integer. Assume that `$v0` is used for the output.

```
begin:    addi $t0, $zero, 0       # $t0 = sum = 0
          addi $t1, $zero, 1       # $t1 = i = 1
loop:     slt  $t2, $a0, $t1       # (n<i)? or (i>n)?
          bne  $t2, $zero, finish  # exit loop if (i>n)
          add  $t0, $t0, $t1       # sum = sum + i
          addi $t1, $t1, 2         # i = i + 2
          j    loop                # repeat loop
finish:   add  $v0, $t0, $zero     # result = sum
```

Result `$v0` is the sum of the odd positive integers 1 + 3 + 5 + … which are less than or equal to `n`.

7. (4 pts) The following code fragment processes an array and produces two important values in registers $v0 and $v1. Assume that the array consists of 5000 words indexed 0 through 4999, and its base address is stored in $a0 and its size (5000) in $a1. Describe in one sentence what this code does. Specifically, what will be returned in $v0 and $v1?

```
        add  $a1, $a1, $a1       # $a1 = 5000 * 2
        add  $a1, $a1, $a1       # $a1 = 5000 * 4
        add  $v0, $zero, $zero   # $v0 = 0
        add  $t0, $zero, $zero   # $t0 = 0
outer:  add  $t4, $a0, $t0       # $t4 = address A[i]
        lw   $t4, 0($t4)         # $t4 = A[i]
        add  $t5, $zero, $zero   # $t5 = count = 0
        add  $t1, $zero, $zero   # $t1 = 0
inner:  add  $t3, $a0, $t1       # $t3 = address A[j]
        lw   $t3, 0($t3)         # $t3 = A[j]
        bne  $t3, $t4, skip      # if (A[i]!=A[j]) skip
        addi $t5, $t5, 1         # count++
skip:   addi $t1, $t1, 4         # j = j+4
        bne  $t1, $a1, inner     # inner loop = 5000
        slt  $t2, $t5, $v0       # if (count < $v0)
        bne  $t2, $zero, next    # then goto next
        add  $v0, $t5, $zero     # $v0 = count
        add  $v1, $t4, $zero     # $v1 = A[i]
next:   addi $t0, $t0, 4         # i = i+4
        bne  $t0, $a1, outer     # outer loop = 5000
```

This code compares every element in the array against all elements for identical matches. It counts the frequency of occurrence of each value in the array. The *count* of the most frequently used value is returned in `$v0` and the *value* itself is returned in `$v1`.