1 Pointers and Structures

```
void increment(node_t * head, int value)
{
  for(node_t * trav = head; trav != NULL; trav = trav->next)
  {
    *(trav->data) += value;
  }
}
```

Write increment as a MIPS funtcion.

```
increment:
# $a0 = head (of type node_t *), $a1 = value (of type int)
increment_loop:
beq $a0, $zero, increment_done      # if( a0 == NULL ) exit
lw $t0, 0($a0)                      # t0 = (*a0).data , i.e. t0 = a0->data
lw $t1, 0($t0)                      # t1 = *(t0) , i.e. t1 = *(a0->data)
add $t1, $t1, $a1                   # t1 += value
sw $t1, 0($t0)                      # *(a0->data) = t1
lw $a0, 4($a0)                      # a0 = (*a0).next , i.e. a0 = a0->next
j loop
increment_done:
jr $ra
```

2 Endian-ness

2.1 Endian Detection

Write a MIPS function that returns the value 1 on a big endian machine and 0 on a little endian machine.

Solution

```
.text
Endian:
        li      $t0, 0x01000000
        sub     $sp, $sp, 4
        sw      $t0, 0($sp)             # save word on stack

        lb      $v0, 0($sp)             # load top most byte
        add     $sp, $sp, 4             # fixup stack
        jr      $ra                     # return
```

There are other ways to do it also. The key is that you need to store and load different sizes (a word and a byte, in this case) to observe endianness. Play around running this function on the Solaris EWS machines (SPARC) and the EWS Linux machines in DCL!.

2.2 memset

In lecture, we discussed pointers and how they are just memory addresses at the machine level. Below is C code for a function that writes an integer value repeatedly `num_words` times starting from address `dst`.

```
void memset(int *dst, int value, int num_words) {
    for (int i = 0; i < num_words; ++i) {
       *dst = value;
        dst++;
    }
}
```

Write it as a MIPS function:

Solution

```
.text
memset:
        li      $t1, 0          # i: $t1, dst: $a0, value: $a1, num_words: $a2
loop:   bge     $t1, $a2, exit  # If i >= num_words, exit loop
        sw      $a1, 0($a0)     # *dst = value;
        addi    $a0, $a0, 4     # Note address increment by 4!
        addi    $t1, $t1, 1     # i++;
        j       loop

exit:
        jr      $ra
```