Ackermann – Spring 2012 Midterm (Patterson)

The Ackermann function A is defined as follows:

\[
A(m, n) = \begin{cases} 
  n + 1 & \text{if } m = 0 \\
  A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0 \\
  A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0.
\end{cases}
\]

Fill in the following C function so that it computes \( A(m, n) \).

```c
unsigned int A(unsigned int m, unsigned int n) {
    if (m == 0) {
        return n + 1;
    } else if (n == 0) {
        return A(m - 1, 1);
    } else {
        return A(m - 1, A(m, n - 1));
    }
}
```

Now you’re going to translate that C into an equivalent MIPS function. We’ve built a skeleton once again, but you’re going to have to fill in the blanks to flesh it out.

```
A:
    addiu $sp, $sp, -12
    sw $s0, 0($sp)
    sw $s1, 4($sp)
    sw $ra, 8($sp)
    addu $s0, $a0, $0
    addu $s1, $a1, $0
    beq $s0, $0, L1
    beq $s1/$a1, $0, L2
    addu $a0, $s0, $0
    addiu $a1, $s1, -1
    jal A
    addu $a1, $v0, $0
    jal A
    j Exit

L1:
    addiu $v0, $s1/$a1, 1
    j Exit

L2:
    addiu $a0, $s0, -1
    addiu $a1, $0, 1
    jal A
    j Exit

Exit:
    lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $ra, 8($sp)
    addiu $sp, $sp, 12
    jr $ra

addiu $a0, $s0, -1
jal A
j Exit
```
MIPS Mystery Question – Fall 2010 Midterm (Katz)
What does the assembly function mystery return? Write your answer as a binary number.

Address Instruction
0x08001000 mystery:  addiu $sp, $sp, -4
0x08001004  sw $ra, 0($sp)
0x08001008  addiu $v0, $zero, 0
0x0800100c  jal inner
0x08001010  lw $ra, 0($sp)
0x08001014  addiu $sp, $sp, 4
0x08001018  jr $ra
0x0800101c inner:
  lw $v0, 4($ra)
  jr $ra

001001 11101 11101 0000 0000 0100
Binary encoding of addiu $sp, $sp, 4.

Cache Question – Fall 2011 Midterm (Garcia)
Take a look at the following C function `sum_iter` run on a 32-bit MIPS machine. On this system, these structs are aligned to two-word boundaries since `sizeof(struct Node) = 8`. Assume the total space taken up by the linked list is greater than (and a multiple of) the cache size.

```c
struct Node {
    int n;
    struct Node *next;
};

int sum_iter (struct Node *head) {
    int sum = 0;
    while (head != NULL) {
        sum += head->n;   // load from head+0
        head = head->next;   // load from head+4
    }
    return sum;
}
```

Given a direct-mapped data cache with this configuration: INDEX: 13 bits, OFFSET: 7 bits

a. How many words are in a block? 32 words
b. How many bytes of data does this cache hold? (in IEC format) $2^{20} = 1$MiB

Define A and B as your answers to (a) and (b) above, respectively.

For questions (c) and (d) below, use these variables in your answer if necessary.

Also, when we mention hit rate below, we’re talking about accessing data (not instructions).

c. What is the lowest possible cache hit rate for the while loop in `sum_iter`? 50%

d. What is the highest possible cache hit rate for the while loop in `sum_iter`? $\frac{(A-1)}{A} \times 100\%$

e. To achieve this maximum hit rate, we obviously could have every Node next to every other node, like an array. However, that’s too strict a constraint -- we can still achieve this hit rate if that’s not the case. What is the loosest constraint for how the Nodes are distributed in memory to get the best hit rate? **Nodes are laid out in memory such that when a block is kicked out, the other 15 nodes in that block will have been accessed (so a block is never kicked out without having a perfect A-1:1 hit rate). Said another way, if you draw a memory block-size wide, then all the “memory blocks” are either full or empty, and “linking” of all of them is so all nodes in a particular block has been visited before another node is visited that shares its tag (so when that block is kicked out, all nodes have been visited).**