MIPS style guidelines

Syntax highlighting

Syntax highlighting greatly enhances the readability of your code and helps catch simple mistakes like typing $t0 vs $t0. We have instructions for setting up highlighting in your editor, so please get that done.

Indentation

Labels should never be indented, and instructions should always be indented one tab. You should set your tab width to 8 spaces and prevent your editor from expanding tabs (Google is your friend if you’re not sure how to).

Commenting

Comments are essential to understanding what your code is doing and figuring out why it might be breaking, particularly since assembly lacks such niceties as variable names, so be liberal with commenting. A helpful guideline is to make a note of which C statement (or part of a statement) each instruction corresponds to, although this isn’t always straightforward, since some statements involve a lot of intermediate steps and conditions are usually inverted.

Instruction formatting

The first operand of every instruction should be aligned; the easiest way to do this is to set your tab width to 8 spaces and then tab after the instruction name. Comments should also be aligned, and once again, tabs are the best way to achieve this (although you might need more than one).

Label names

Label names should be all lower case with underscores separating words. Try to make your label names meaningful but not excessively long; a good tip is to base label names on the C control structures they correspond to. Some label names are pretty generic, so to disambiguate, you can prefix them with an abbreviation of the function name. For example, for the function called count_odd_elements below, I’ve referred to the start of the for loop as coe_for.

Examples

Here’s some C code we’ll be translating – it counts the number of odd elements in an array.

```c
int count_odd_elements(int *array, int size) {
    int num_odd_elements = 0;
    for (int i = 0; i < size; ++i) {
        if (array[i] & 1) {
            ++num_odd_elements;
        }
    }
    return num_odd_elements;
}
```

This is a functional but unreadable MIPS translation. Please don't do this.
count_odd_elements:
  li $v0, 0
  li $t0, 0

foo:
bge $t0, $a1, baz
sll $t1, $t0, 2
add $t1, $a0, $t1
lw $t1, 0($t1)
and $t1, $t1, 1
beqz $t1, bar
add $v0, $v0, 1

bar:
add $t0, $t0, 1
j foo

baz:
jr $ra

This is significantly better - the alignment is off, but there are comments and label names make sense.

count_odd_elements:
    li $v0, 0 # num_odd_elements = 0
    li $t0, 0 # i = 0

coe_for:
    bge $t0, $a1, coe_end # branch if !(i < size)
sll $t1, $t0, 2
add $t1, $a0, $t1 # calculate &array[i]
lw $t1, 0($t1) # load array[i]
and $t1, $t1, 1 # calculate array[i] & 1
beqz $t1, coe_for_next # branch if !(array[i] & 1)
add $v0, $v0, 1 # ++num_odd_elements

coe_for_next:
    add $t0, $t0, 1 # ++i
    j coe_for

coe_end:
    jr $ra # return num_odd_elements

This is the above with proper alignment. This is what we want your MIPS code to look like.
count_odd_elements:
    li      $v0, 0                  # num_odd_elements = 0
    li      $t0, 0                  # i = 0

coe_for:
    bge     $t0, $a1, coe_end       # branch if !(i < size)
    sll     $t1, $t0, 2
    add     $t1, $a0, $t1           # calculate &array[i]
    lw      $t1, 0($t1)             # load array[i]
    and     $t1, $t1, 1             # calculate array[i] & 1
    beqz    $t1, coe_for_next       # branch if !(array[i] & 1)
    add     $v0, $v0, 1             # ++num_odd_elements

coe_for_next:
    add     $t0, $t0, 1             # ++i
    j       coe_for

coe_end:
    jr      $ra                     # return num_odd_elements