Lecture 7

or “TGIAF”
Turn in yo homework!

I will hopefully have it back to you by next class so that we can go over it.

I am not grading Section 2’s (1:15-2:30), but I will normalize the grades with my own grading
Next Homework!
Next Homework!

Probably going to give it to you next time, some kind of MIPS assignment
Recap from last class

We learned about branch statements and jumps.

We saw how these were used to make if statements and loops
(I feel like the other example had too much in it to get the point across so...)

while(i < j) i++;
while(i < 5) i++;
Loops

while(i < j) i++;
$s1 = i
$s2 = j
while(i < 5) i++;
$s1
Loops

We have a command that we briefly discussed in class last time for doing less than stuff.

slt and slti

Which of these should be used for each loop?
Loops

Let’s deal with the first one first:
while(i < j) i++;
Loop: slt $t0, $s0, $s1
    beq $t0, $zero, Exit
    addi $s0, $s0, 1
    j Loop
Exit
Now for the next one

while(i < 5) i++;
Loop: slti $t0, $s0, 5
    beq $t0, $zero, Exit
    addi $s0, $s0, 1
    j Loop
Exit
Done with loops

Hopefully loops make sense now, if not we can do some more of this after homework review next time
Now we are going to discuss everything that goes into a MIPS assembly file to make it “assemble-able”
Assembly

Parts of your code:
Comments (denoted by “#”)
Algorithm
   Basically your code
Other essentials
   Assembler directives: .data, .text
   Labels: main, user defined
   syscall: suspends execution of program and transfers control to the OS
      The OS looks for contents in $v0
Example

.data
var1: .word 23  # declare storage for var1; initial value is 23

The .data section of your program tells the assembler “Hey, I want to store stuff into memory”

Part by Part breakdown

  var1 = address that we are storing stuff into
  .word = what type of data is coming next (other ex. .byte...)

Example cont.

.text
start:
    lw $t0, var1    # load contents $t0 = var1
    li $t1, 5      # $t1 = 5 ("load immediate")
    sw $t1, var1   # store contents into RAM: var1 = $t1

.text portion tells the assembler “Hey, the next part contains instructions”
More example

la $t0, var1 # load address of var1 to $t0
lw $t2, ($t0) # load content $t2 = var1
sw $t2, ($t0) # store content var1 = $t2
What are li and la?

Well, that’s an interesting question, one that I had to Google...
If you recall, MIPS is a RISC architecture, meaning it has a more simpler instructions set.

While this seemed to work fine for a while, programs kept getting bigger and assembly code longer.
PseudoInstructions!

Memory was no longer an issue, so why limit the instructions? Well, we don’t want to just put everything in there, but we do want to make the code more readable, and faster.
The developers of the instruction set decided that it would make more sense to benchmark instructions and decide if they belong based on their speed.
That’s where PseudoInstructions come in
PseudoInstructions!

Basically, a pseudoinstruction is a fake instruction. It is taken in by the assembler, converted to whatever assembly it corresponds to, and then the program is created from that.

This allows us to more easily read the code, but also ended up making the code faster.
Another quick example...

.data
array1: .space 12 # declare 12 bytes of storage to hold array of 3 integers
.text
start:   la $t0, array1 # load base address of array into register $t0
   li $t1, 5 # $t1 = 5 ("load immediate")
   sw $t1, ($t0) # first array element set to 5; indirect addressing
   li $t1, 13 # $t1 = 13
   sw $t1, 4($t0) # second array element set to 13
   li $t1, -7 # $t1 = -7
   sw $t1, 8($t0) # third array element set to -7
What if...

What do we do if we want to get user input? Can we do that?
Using syscall

Load the service number into register $v0
Load argument values, if any, into $a0, $a1, $a2, or $f12 (specified in chart)
Make SYSCALL instruction call
Retrieve any return values from result registers
## syscall

<table>
<thead>
<tr>
<th>Service</th>
<th>$v0</th>
<th>Arguments</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_int</td>
<td>1</td>
<td>$a0 = int to print</td>
<td></td>
</tr>
<tr>
<td>print_float</td>
<td>2</td>
<td>$f12 = float to print</td>
<td></td>
</tr>
<tr>
<td>print_double</td>
<td>3</td>
<td>$f12 = double to print</td>
<td></td>
</tr>
<tr>
<td>print_string</td>
<td>4</td>
<td>$a0 = add. of string</td>
<td></td>
</tr>
<tr>
<td>read_int</td>
<td>5</td>
<td>int returned in $v0</td>
<td></td>
</tr>
<tr>
<td>read_float</td>
<td>6</td>
<td>float returned in $v0</td>
<td></td>
</tr>
<tr>
<td>read_double</td>
<td>7</td>
<td>double returned in $v0</td>
<td></td>
</tr>
<tr>
<td>read_string</td>
<td>8</td>
<td>$a0 = add of string input</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a1 = length of buffer</td>
<td></td>
</tr>
<tr>
<td>sbrk</td>
<td>9</td>
<td>$a0 = amount</td>
<td>address in $v0</td>
</tr>
<tr>
<td>exit</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

We want to add 2 ints
- We will take input from the user
- Add the inputs
- Output the sum
- Exit
.text
main:
    li $v0, 5 # load syscall read_int into $v0.
syscall # make the syscall.
move $t0, $v0 # pseudo instruction, move the number read into $t0.
    li $v0, 5 # load syscall read_int into $v0.
syscall # make the syscall.
move $t1, $v0 # move the number read into $t1.
add $t2, $t1, $t0 # add and store into $t2
move $a0, $t2
    li $v0, 1 # prints integer
    syscall
    li $v0, 10
    syscall
Download this!

http://courses.missouristate.edu/KenVollmar/MARS/

Try some MIPS code