Another example: The C / C++ code

```c
#include <iostream.h>

int main ( )
{
    int i;
    int sum = 0;
    for(i=0; i <= 100; i++)
        sum = sum + i*i ;
    cout << "The answer is " << sum << endl;
}
```

Let’s write the assembly ...
Assembly Language Example 1

.text
.align 2

main:
    move $14, $0     # i = 0
    move $15, $0     # tmp = 0
    move $16, $0     # sum = 0

loop:
    mul $15, $14, $14 # tmp = i*i
    add $16, $16, $15 # sum = sum + tmp
    addi $14, $14, 1  # i++
    ble $14, 100, loop # if i < 100, goto loop

# how are we going to print the answer here?  
# and how are we going to exit the program?
System Call Instruction

- System call is used to communicate with the operating system and request services (memory allocation, I/O)
  - syscall instruction in MIPS
- SPIM supports “system-call-like”
  1. Load system call code into register $v0
    - Example: if $v0==1, then syscall will print an integer
  2. Load arguments (if any) into registers $a0, $a1, or $f12 (for floating point)
  3. syscall
    - Results returned in registers $v0 or $f0
### SPIM System Call Support

<table>
<thead>
<tr>
<th>code</th>
<th>service</th>
<th>ArgType</th>
<th>Arg/Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>print</td>
<td>int</td>
<td>$a0</td>
</tr>
<tr>
<td>2</td>
<td>print</td>
<td>float</td>
<td>$f12</td>
</tr>
<tr>
<td>3</td>
<td>print</td>
<td>double</td>
<td>$f12</td>
</tr>
<tr>
<td>4</td>
<td>print</td>
<td>string</td>
<td>$a0 (string address)</td>
</tr>
<tr>
<td>5</td>
<td>read</td>
<td>integer</td>
<td>integer in $v0</td>
</tr>
<tr>
<td>6</td>
<td>read</td>
<td>float</td>
<td>float in $f0</td>
</tr>
<tr>
<td>7</td>
<td>read</td>
<td>double</td>
<td>double in $f0 &amp; $f1</td>
</tr>
<tr>
<td>8</td>
<td>read</td>
<td>string</td>
<td>$a0=buffer, $a1=length</td>
</tr>
<tr>
<td>9</td>
<td>sbrk</td>
<td>$a0=amount</td>
<td>address in $v0</td>
</tr>
<tr>
<td>10</td>
<td>exit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Echo number and string

```
.text
main:
   li   $v0, 5      # code to read an integer
   syscall        # do the read (invokes the OS)
   move $a0, $v0   # copy result from $v0 to $a0

   li   $v0, 1      # code to print an integer
   syscall        # print the integer

   li   $v0, 4      # code to print string
   la   $a0, nln    # address of string (newline)
   syscall

# code continues on next slide ...
```
Echo Continued

```
li  $v0, 8    # code to read a string
la  $a0, name # address of buffer (name)
li  $a1, 8    # size of buffer (8 bytes)
syscall

la  $a0, name # address of string to print
li  $v0, 4    # code to print a string
syscall

jr  $31        # return
```

.data
.align 2
name:  .word 0,0
nln:   .asciiz "\n"