Assembly Language Programming

Modeling Motion Week 5 Computer Lab

This week we will investigate how programs written in high-level languages like Python are translated into machine language that can be executed on real microprocessors. We will try to translate simple Python programs into the assembly language for a simulated microprocessor, the MM1 (Modeling Motion I).

1) Write an MM1 assembly language program to compute 8! (eight factorial) leaving the result in memory at address 63. First write a Python program that computes factorial and then "compile" it into assembly language.

2) Write an MM1 assembly language program to compute the y position of a body under uniform acceleration. Start the body out at y=30 with a velocity of zero and a constant downward acceleration of 1 unit per step. Simulate bouncing by setting y to zero whenever the body drops below y<0 and making its velocity positive.

3) The following assembly language program assumes there exists a function that computes factorial defined begining at the label, `factorial:`. Write this function to complete the program leaving the final result in register *r0*. Note that this program assumes the factorial function will compute the factorial of the number in *r0* when it is called and overwrite r0 with the result. The program below computes the factorial of 3 followed by the factorial of that result. (3!=6, 6!=720). A working program will halt with a value of 720 in register *r0*

```
    LD N r0
    CALL factorial
    CALL factorial
    HALT

factorial:
    #replace this comment with your code

# Number to take factorial of
N:  DATA 3
```

See next page for examples in Python....

Below are two Python programs (also on the website in the week 5 Python Examples, factorial.py) showing two versions of the factorial function: 1) an iterative one with a loop and 2) a recursive one that calls itself. You may choose to implement either or both in assembly language.

```
#==========================================#
# Iterative version of factorial function  #
#    (using a loop)                        #
#==========================================#
def factorial(N):
    factor = 1

    while N>0:
            factor *= N
    N -= 1
    return factor

print "Iterative version:"

a=factorial(3)
print "fact(3)=%d" % a

b=factorial(a)
print "fact(%d)=%d" % (a,b)

#================================#
# Recursive version of factorial #
#  (function that calls itself)  #
#================================#
#
# recursive definition of factorial works like this:
#  1! = 1
#  N! = N*(N-1)!
#
#  Factorial is defined in terms of itself for all integers
#  except 1. This recursive definition leads to a function
#  which calls itself in Python.
#
#  example:
#  4! = 4*(4-1)!=4*3! but 3!=3*2! and 2!=2!*1! and 1!=1 (by definition)
#
#  in other words:
#  4!=4*(3*(2*(1)))=4*3*2*1

# This recursive definition encoded in Python looks like so:
def rfactorial(N):
    if N==1:
       return 1
    else:
       return N*rfactorial(N-1)

print "\nRecursive version:"
a=rfactorial(3)
print "fact(3)=%d" % a

b=rfactorial(a)
print "fact(%d)=%d" % (a,b)
```