Programming Assignment 2

Shreya Bajpai

Rutgers, The State University of New Jersey

Computer Architecture (01:198:211) - Summer 2017

Section F6 - Professor Brian M. Russell

Overview

In this directory, you will find x86 assembly code which contains nCr (nCr.c and nCr.h) along with a C library, formula, that implements the assembly code.

Formula Library

Formula

In formula, we take **one** argument from our command line: power. We use power to determine the distributed form of (1 + x)^n, where *n* is the inputted power.

```
$ ./formula 5 (1 + x)^5 = 1 + 5*x^1 + 10*x^2 + 10*x^3 + 5*x^4 + 1*x^5 Time
Required = 50 microsecond
```

In my implementation, we assume the following things, as stated by Professor Russell:

1. We **do not** have to account for overflow (despite what it says in the .pdf of the instructions on the Sakai assignment page)
2. We **assume** that the user will enter a non-negative integer for the power input
3. We **assume** that we will only be working with 32-bit registers, and will not print numbers over the power of 12, as stated by Professor Russell

Implementation

Formula.c - implements nCr.s, since the latter is simply assembly code.

- **Runtime** - *O(n^2)* - the main method takes an input *power* and computes *n* values contingent on the input. To calculate each value takes *O(n)* since it requires nCr to run. As a result, the runtime is n*O(n) = O(n^2).
- **Functions**

- **main()**
  - We perform an argument check
  - We have to convert the input from ASCII to integer so nCr can manipulate the integer values. We declare a variable *int power* and store the int value in it
  - We use gettimeofday() to calculate runtime, as suggested by Professor Russell
  - We print *long* finVal of the polynomial when we call nCr from nCr.s
  - We ask the computer to compute the current time
  - We subtract and print the recorded times that represent the main method's runtime

nCr

In nCr, we are given the header file (nCr.h), which contains the two functions we must implement in nCr.s, written completely in x86 assembly, which will be implemented by formula.c. Both functions highlighted in nCr implement local variables and manipulate them from register to register in the respective stack frames of nCr and Factorial.

I present the runtime of both functions below:

- **nCr runtime**
  - *O(n)* - this function calls the code Factorial thrice to compute factorial and executes a division at the end (3 * O(n) = O(3n) → ignore the constant → O(n))
- **Factorial runtime**
  - *O(n)* - the function only has one iteration, when it determines that the power inputted is greater than 1 and goes on to compute factorial (e.g., n! = 1*2*3*...*n) for x-1 times

Implementation

nCr.s - We use this library, in x86 assembly code, to perform the necessary functions to compute possible combinations that can be obtained by taking a sample of items from a larger set. We use nCr to perform checks on *n* (set/population) and *r* (subset of *n*/sample set).

- **Functions**
  - **nCr**
    - Push base pointer (rbp)
    - Copy stack pointer into base pointer
    - Creates new variable (result) and makes room to store its value in %r12
    - Stores the input values in edi and esi respectively
    - Puts the value of the result in edi
    - Calls factorial on eax which holds (facotiral(n)/(factorial(r) * facotrial (n-r)))
    - Sees if the result given back from factorial calls is less than 0, if so, it returns 0

        - If not, then it returns result to formula.c which is the calle function
    - **Factorial - recursive**
        - pushes base pointer
        - copies stack pointer into base pointer
        - copies the first parameter into the return register eax
        - Sees if the num passed in is less than or equal to 1, if it is less than it puts 1 in the return register and leaves the function
        - Makes room on stack and recursively calls factorial result = (factorial * factorial(n-1)

```
int nCr (int n, int r) {
  int result;
  result = Factorial(n)/(Factorial(r)*Factorial(n-r));
  if (result < 0) {
      return 0;
  }
  else {
      return result;
  }
}

int Factorial(int n) {
      if (n >= 1) {
        return n * Factorial(n-1);
      }
    else {
        return 1;
    }
}
```