Lab 2 – It's all in the Timing

Handed out: September 14, 2011, 11am
Due: September 21, 2011, by 10am

Introduction

The purpose of this assignment is to start learning about assembly language programming and to better understand the execution time of a program.

Logistics

You are to **code and write individually** to solve this assignment – you may discuss the problem and the writeup with others, but you may not look at their code or their text. Your submission (via moodle) will consist of **a single .pdf file**, including brief reports for each part, along **with the associated program code**.

Part: 1

Intel-compatible machines provide a special 64-bit register that is incremented every clock cycle. This register can be read using the rdtsc assembly instruction. This register can be used to calculate a processing time for a particular piece of code.

For this part of the lab, use the code provided below that uses GCC's inline assembly mechanism to a timing table similar to the one shown below. The table illustrates the number of clock cycles required for function `sum(int n)` to execute with different values of *n*. The function `sum(int n)` needs to be implemented by you and should sum the integers from 1 to *n*.

Below is the primary code that can be used to generate a single entry in the processing time table.

```
double t;
start_counter();
sum(n)
t = get_counter();
printf("P required %f clock cycles\n", t);
```

The following function will insert the given assembly code into your compiled program.

```
static unsigned cyc_hi = 0;
static unsigned cyc_lo = 0;

/* Set *hi and *lo to the high and low order bits
   of the cycle counter.
*/
void access_counter(unsigned *hi, unsigned *lo)
{
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
       : "=r" (*hi), "=r" (*lo)
       :
       : "%edx", "%eax");
}
```

These are the final functions provided to you for the program.

```
/* Record the current value of the cycle counter. */
void start_counter()
{
    access_counter(&cyc_hi, &cyc_lo);
}

/* Number of cycles since the last call to start_counter. */
double get_counter()
{
    unsigned ncyc_hi, ncyc_lo;
    unsigned hi, lo, borrow;

    /* Get cycle counter */
    access_counter(&ncyc_hi, &ncyc_lo);

    /* Do double precision subtraction */
    lo = ncyc_lo - cyc_lo;
    borrow = lo > ncyc_lo;

    hi = ncyc_hi - cyc_hi - borrow;
    return (double) hi * (1 << 30) * 4 + lo;
}
```

To compile and run your program, use:

```
gcc program.c -o program
./program
```

If you'd like, you can copy an example .c file and Makefile from:

/usr23/taylorm/public_html/2011_cs203/lab1 /*

Once your code compiles, run it on one of the compute machines and on one of the jazz machines to generate different tables. When creating this table, you should have at least 5 different values for n and you should run the computation `sum(int n)` for each value twice

N	**Cycles**	**Cycles**
100	961	9.61
1,000	8,407	8.41
1,000	8,426	8.43
10,000	82,861	8.29
10,000	82,876	8.29
1,000,000	8,419,907	8.42
1,000,000	8,425,181	8.43
1,000,000,000	8,371,2305,591	8.37

To look at the assembly code, use:

`gcc program.c –S` (note: S is capitalized)

Write a report that contains these tables and discusses the differences in the results. This report should also include how the assembly code differed for function `access_counter` on each machine.

Part: 2

Create two functions: `factorial(int n)` and `factorial_double(double n)`. Each function will find the factorial of *n* (i.e., `factorial(5)` = 5! = 5 × 4 × 3 × 2 × 1) but `factorial` will take an integer as input and return an integer as output, while `factorial_double` will take a double as input and return a double as output.

As before, run your code on both one of the compute machines and one of the jazz machines. How do the running times of the two functions differ? Are you surprised by the (large/small/no) differences between the different machines? How does the assembly code differ between the two functions and between the two machines?

Note: your lab will be graded based on the quality of both your report and results.