Project 2: Big Integers

Due: Tuesday October 23, 2001

Objective

The objective of this programming assignment are 1) learn how to write assembly language routines that can be called from C/C++, 2) gain further insight into two s complement arithmetic.

Background

When you think about it, integer variables in C/C++ are really horrible. If you add two large positive numbers together, you can get an overflow and end up with a negative number. What s worse, the programmer has very little control over the situation. Unlike assembly language programming, no flags are set when an overflow occurs.

In C++, you can make your data types. For this project, you are provided with a predesigned class called BigInt. When you add two BigInts together, you never get an undetected overflow. The data structure is automatically expanded to fit a larger number. However, as we mentioned previously, overflows are hard to detect in C/C++, so part of the implementation of the BigInt class must be done in assembly language programming.

Assignment

For this project, you will write the assembly language code that will finish the implementation of the BigInt class. The C++ source for the BigInt class can be found in the directory:

```
/afs/umbc.edu/users/c/h/chang/pub/cs313/proj2
```

The header file `bimath.h` has the function prototypes of the functions that need to be implemented. The file `bimath.asm` has the stubs for these functions. Note that for assembly language programs to work with C++, the labels for the entry points of the functions have funny looking names. This is because C++ allows function names to be overloaded. However, C++ functions use the same parameter passing convention as far as the stack is concerned.

These files as they are will compile. For example, you can type in the commands:

```
nasm -f elf bimath.asm
g++ bigint.cpp main1.cpp bimath.o
```

You will get an a.out file that runs. The main program main1.cpp only exercises the input and output routines supplied with the BigInt class. The arithmetic functions are not used. The functions that need to be implemented are: addition, subtraction, multiplication, division and comparison.

For 92% of the project grade, you must implement the addition, subtraction and comparison functions. If you implement multiplication, you get the remaining 8%. Division is 10% extra credit.

Implementation Issues:

1. The BigInt class uses a dynamically allocated array to store arbitrarily large numbers. Internally, you should treat the numbers as little endian two s complement numbers.
2. When you add and subtract numbers of different lengths, the shorter one must be *sign extended*.
3. Multiplying signed and unsigned numbers are different.
4. The BigInt class is designed to take care of memory allocation issues in C++ rather than assembly language. For example, the `sum` parameter passed to the addition function already

has enough memory to hold the sum of the two addends. You should set the `len` field appropriately to indicate the length in bytes of the sum, since the sum doesn t necessarily need as much memory as is provided.

5. Don t change the `bufsize` field in your assembly language program.
6. You shouldn t need to allocate memory from assembly language program.
7. When a variable is passed as a reference, the address of the variable is pushed onto the stack.
8. The memory for the data members of an object is allocated in order. For the BigInt class, there are 3 data members: ptr, len and bufsize. They are stored in 4 bytes each with ptr taking the lowest numbered address (the same address as the object) and bufsize taking the highest numbered address.
9. Your program should compile and run without any modifications to `bigint.h`, `bigint.cpp` and `bimath.h`.
10. You should look up some assembly language instructions that will be useful in this project. These include xchg (exchange the contents of two registers), adc (add with carry), sbc (subtract with carry), jo (jump on overflow), jno (jump on no overflow), jc (jump on carry), jnc (jump on no carry), ...
11. You should pay special attention to which instructions affect which flags. For example, jump instructions do not affect the flags. Similarly, move instructions leave the flags alone. Increment and decrement instructions do not alter the carry flag, but do alter the overflow flag. When in doubt, consult the Intel manuals.

Turning in your program

Before you submit your program, record some sample runs of your program using the UNIX script command. You should select sample runs that demonstrate the features supported by your program. Picking good test cases is **your responsibility**.

Create a file called README. In the file, describe the features that you have implemented and those you have not. Include instructions on how to compile your project. If certain parameters cause your program to crash, describe those. And of course, describe the cases where your program functions correctly.

Use the UNIX `submit` command on the GL system to turn in your project. You should submit the README file and all the files needed to compile and run your program. The class name for submit is `cs313` and the project name is `proj2`.

Addendum: make a typescript file of your sample runs and submit that file as well.