University of Virginia Computer Science *http://www.cs.virginia.edu/cs216/ps/ps5/*
CS216: Program and Data Representation, Spring 2006 22 March 2006

[PDF version for printing]

Problem Set 5
Representing Numbers

Out: 22 March
Due: Wednesday (beginning of class), 29 March

Collaboration Policy - Read Carefully (similar to PS4)

For this assignment, you may work on your own or with any one other person of your choice **except for anyone you worked with on PS3 or PS4**. If you work with a partner, you should turn in one assignment with both of your names on it. **If you would prefer to be assigned a partner, send email to *evans@cs.virginia.edu* before 5pm on Friday, 24 March** (include any constraints or preferences you have on your assigned partner). If a suitable match requests a partner, you will receive a partner assignment. Partners will be assigned using a greedy algorithm based on when requests arrive, so you are more likely to receive a suitable partner assignment if you send in your request early.

You may consult any outside resources including books, papers, web sites and people you wish, except you may not copy code from other big number representation implementations. There are many implementations of big numbers in C available on the web (for example, there is one in the Python interpreter and GMP), and it would certainly defeat the purpose of this assignment if you copied one of them instead of thinking on your own. You may look at design documents for big integer implementations is you wish, but we recommend thinking about your own design first, and only looking for other designs if you are really stuck.

You are also encouraged to discuss these problems with students in the class. You must acknowledge any people and outside resources you work with on your assignment. If you discuss the assignment with people other than your partner, you may not take any written materials out of your discussion. It is fine to bounce ideas off other people, but the answers you turn in must be your own.

You are **strongly encouraged** to take advantage of the staffed lab hours posted on the CS216 web site.

Purpose

- Learn how numbers are represented.
- Discover experimentally properties of your C implementation
- Implement a variable-length integer encoding in C that is better than Microsoft's (Excel)

> **Submission:** For this assignment, you should submit all your answers (including code) on paper in class on March 29. In addition, you should submit your code for your `bignum` implementation (questions 7-9) electronically using the form at *http://www.cs.virginia.edu/cs216/ps/ps5/submit.html*

Power Analysis

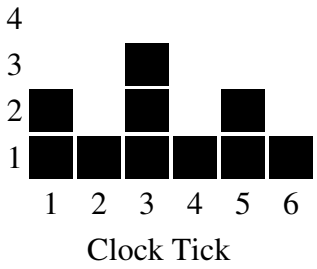
Differential Power Analysis is a technique for analyzing the power consumption a device uses to determine something about the cryptographic keys is contains. The amount of current a processor uses depends on what operations it is performing. In particular, flipping the value of a bit uses more current than when the same value is maintained. Although the current differences is small, they can be measured (especially if the processor is repeatedly performing operations).

Here, we will consider a simple hypothetical smart card with one 8-bit register using little-endian encoding. The current used for each operation scales linearly with the number of bits in the register that flip (either a 0 becomes a 1, or a 1 becomes a 0). The card implements a simple counter: the value in the register increases

by one each clock tick. So, if the register initially contains `0000 0011` after the next clock ticks the resuting value in the register will be `0000 0100` and 3 units of current will have been consumed.

Your goal is to determine the number stored in the register, just by observing the current consumed by the card.

1. Suppose you measure the current consumed as shown below. What possible values could be stored on the card after clock tick 6?



2. What is the fewest number of additional clock tick current readings that could be enough to uniquely determine the exact value stored on the card?

3. Suggest a number representation the smart card designers could use that would be less vulnerable to power analysis attacks. If necessary, you may add extra bits to the register, although this is undesirable as it increases the cost of the smart card. Analyze the feasibility of a power analysis attack to determine the number stored in your card.

Exploring Numbers

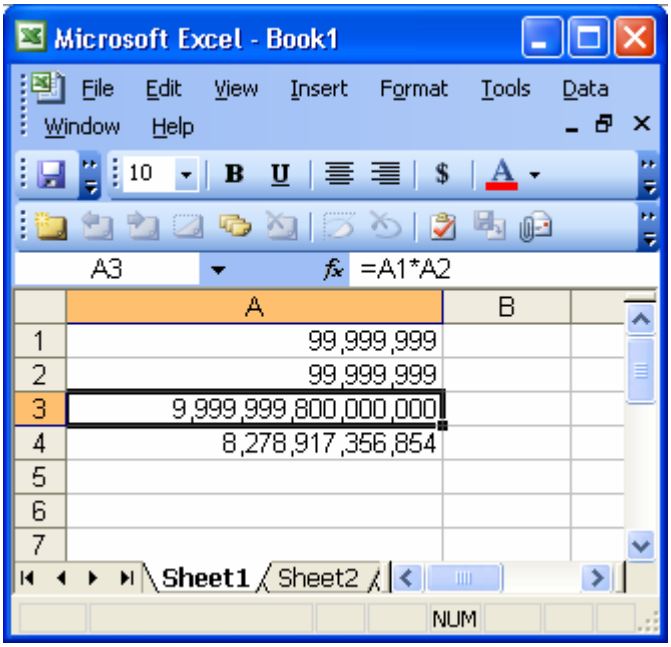
As we saw in class, the C specification leaves many properties of integer number representation up to the implementation. It does not specify the number of bits in an integer (other than the minimum number for the `int` type being 16 bits).

4. Write a simple program that determines how many bits the C implementation provides for the `int` and `long` datatypes. (Your answer should include both the program you wrote and the result. You may not use any of the constants defined in `limits.h`.)

5. Write a simple program that determines whether your C implementation uses a big-endian or little-endian for the `int` datatype.

Implementing Big Numbers

C provides fixed length integer representations, which means that all arithmetic is broken when high enough values are used. Programs that attempt to implement arithmetic correctly need to develop variable-length integer representations. Doing this correctly and efficiently is difficult. For example, here is what Excel does:



(Note that the product of any two numbers ending in 9 should end in a 1, so Excel must be wrong. The number on the bottom line is the current US national debt to give a sense that numbers of this scale could meaningfully occur in spreadsheets.)

For the next questions your goal is to design and implement a variable length integer implementation that is more accurate than Excel's.

6. Design your number representation. Your design should explain clearly how to map numbers to and from your bit representation. In developing your design, think carefully about how your design decisions will affect the difficulty of implementing addition and multiplication using your number representation, as well as the amount of memory consumed. Your number representation should support positive integers from 0 to some high bound (ideally, you would support numbers up to any bound, limited only by available machine memory).

The desired properties are:

1. Use one word (or less) for small numbers
2. Be able to store arbitrarily large numbers
3. Be able to perform addition and multiplication efficiently

For the programming questions 7, 8, and 9, your implementation must use exactly the type signatures shown here, and your code should be in a single C file, `bignum.c`, with a corresponding header file `bignum.h` that defines your datatype. When you submit your code, it must match the provided signatures. You are free (and encouraged when appropriate) to add additional functions to your code as you see fit.

7. Implement your number representation as a `bignum` C data abstraction. Provide these functions:

```
bignum bignum_create (char *p_x);
   /* post: returns a bignum object representing the value
            p_x where p_x is a decimal number represented
            as a string */

char *bignum_unparse (bignum p_b);
  /* post: returns a string representation of p_b that
           shows its decimal value */
```

8. Implement the `bignum_add` function:

```
bignum bignum_add (bignum p_b1, bignum p_b2);
   /* post: returns a bignum representing the value
              p_b1 + p_b2 */
```

9. Implement the `bignum_mult` function:

```
bignum bignum_mult (bignum p_b1, bignum p_b2);
   /* post: returns a bignum representing the value
               p_b1 * p_b2 */
```

If your implementation is correct, you should be able to evaluate

```
printf ("%s\n", bignum_unparse
                  (bignum_mult (bignum_create ("99999999"),
                                bignum_create ("99999999"))));
```

to demonstrate that your number representation is more accurate than Excel's.

10. Describe the asymptotic running time and memory use of your `bignum` implementation.

Remember to submit your `bignum.c` and `bignum.h` files here: *http://www.cs.virginia.edu/cs216/ps/ps5/submit.html*.

The submission program will run your `bignum` implementation on several test cases and check it produces the correct answers.