CSCE 557
Notes following Trappe and Washington, Introduction to Cryptography with Coding Theory

D. A. Buell

Department of Computer Science and Engineering
University of South Carolina

Fall 2005
Outline

1. RSA
Basic Principles of Arithmetic

Arithmetic is basically a set of array operations

\[
\begin{array}{cccccc}
1 & & & & & (\text{carries}) \\
9 & 7 & 5 & 3 & 1 & 2 & 4 & 6 & 8 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
1 & 0 & 9 & 8 & 7 & 6 & 9 & 2 & 5 & 7
\end{array}
\]

and the radix doesn’t have to be the usual base ten radix

\[
\begin{array}{cccccc}
1 & & & & & (\text{carries}) \\
9 & 75 & 31 & 24 & 68 \\
1 & 23 & 45 & 67 & 89 \\
\hline
10 & 98 & 76 & 92 & 57
\end{array}
\]
Choice of Radix

- Radix is governed by wordsize and multiply characteristics
- Assume that the machine wordlength is $k$ bits
- Make sure you use unsigned arithmetic to avoid the complexity of twos-complement issues with the leftmost bit
- If the product of two single length operands of $k$ bits can be obtained as a double length product of $2k$ bits, then use radix $2^k$ and extract left and right halves of the product
- On some machines this will involve one machine instruction for the left half and one more machine instruction for the right half of the product
- If the product of two single length operands of $k$ bits is only a single length product of $k$ bits (with the chance of overflow), then use radix $2^{k/2}$
Complexity of Multiplication

\[
\begin{array}{c|c}
A & B \\
C & D \\
\hline
A*D & B*D \\
C*A & C*B \\
\hline
\end{array}
\quad
\begin{array}{c|c}
AB & CD \\
\hline
AB * CD \\
\end{array}
\]

\(k/2\)-bit radix versus \(k\) bit radix costs a factor of FOUR in the number of multiplication instructions
Signed-Magnitude Arithmetic Versus Twos-Complement

With twos-complement arithmetic, “addition” and “subtraction” are just array adds regardless of the “sign” of the operands, but a short operand needs to be padded to full length, costing both time and space.

\[
\begin{array}{cccc}
4 & 7 & 3 & 2 \\
\hline
-1 \\
4 & 7 & 3 & 1 \\
\end{array}
\]

4-bit 2’s comp

\[
\begin{array}{cccc}
4 & 7 & 3 & 2 \\
F & F & F & F \\
1 & 4 & 7 & 3 & 1 \\
\end{array}
\]

(We ignore carries left past the bit length.)
Signed-Magnitude Arithmetic Versus Twos-Complement

- Signed-magnitude requires a “full-length” operand for anything negative—there is no chance to save space (or time)
- Multiplication and division are equally difficult in either signed or 2’s complement arithmetic.
- Under some circumstances, such as modular arithmetic, the naive algorithms only use positive numbers, and we can completely ignore the issue of signs.
Memory Space

- Operand $a$ of $m$ digits base $2^k$, i.e., $a \leq mk$ bits
- Operand $b$ of $n < m$ digits base $2^k$, i.e., $b \leq nk$ bits

- $a + b$ could be from 0 to $m + 1$ digits depending on signs, magnitudes, carry

- $a \times b$ could be of $m + n$ or $m + n - 1$ digits

- $a/b$ could be of $m - n$ or $m - n + 1$ digits

- How do we deal with the digits gained or lost?
Memory Space

- Addition runs right to left (store array backwards)

- Multiplication runs right to left (store array backwards)

- Division runs left to right (store array forwards)

- There’s no guarantee we won’t have to do a shift at the end

- How do we handle temporary space for $a \leftarrow a + b$?
Memory Space

- Don’t malloc/free space exactly as needed

- A little wasted space is ok? Malloc for the upper bound, free only if used space is less than (e.g.) half allocated? (Needs a number-of-digits value as well as a malloc-ed space value.)

- “General purpose” but slow code versus “narrow use” but fast code?

- **gmp** as an example of how to do this