Session 57: Arithmetic with Base 2 Expansions

- Addition
- Multiplication
- Modular Exponentiation

Adding $a = (1110)_2$ and $b = (1011)_2$ $\longleftarrow$ $\longleftarrow$

$a_0 + b_0 = 1$ $s_0 = 1$

$a_1 + b_1 = 1 + 1 = 10$ $s_1 = 0$ $c = 1$

$a_2 + b_2 + c = 1 + 0 + 1 = 10$ $s_2 = 0$ $c = 1$

$a_3 + b_3 + c = 1 + 1 + 1 = 11$ $s_3 = 1$ $c = 1$

$s_4 = 1$

$s = (s_4 s_3 s_2 s_1 s_0)_2 = (11001)_2$

$a_i + b_i + c$

possible results:

c	s
0	0
0	1
1	0
1	1

$$c_{new} = \left\lfloor \frac{a_i + b_i + c_{old}}{2} \right\rfloor \qquad s = a_i + b_i + c_{old} - 2c_{new}$$

Binary Addition of Integers

procedure *add*(a, b: positive integers)

{the binary expansions of a and b are $(a_{n-1}, a_{n-2},\ldots, a_0)_2$ and $(b_{n-1}, b_{n-2},\ldots, b_0)_2$, respectively}

$c := 0$

for $j := 0$ **to** $n - 1$

  $d := \lfloor (a_j + b_j + c)/2 \rfloor$

  $s_j := a_j + b_j + c - 2d$

  $c := d$

$s_n := c$

return$(s_0, s_1,\ldots, s_n)$

{the binary expansion of the sum is $(s_n, s_{n-1},\ldots, s_0)_2$}

The number of additions of bits used by the algorithm to add two n-bit integers is $O(n)$.

Binary Multiplication

Two observations

$a \cdot b = a \cdot (b_k 2^k + b_{k-1} 2^{k-1} + \dots + b_1 2 + b_0) = ab_k 2^k + ab_{k-1} 2^{k-1} + \dots + ab_1 2^1 + ab_0 2^0$

Multiplying a binary number by 2^j is corresponds to add j zeros at the end

Example: Multiply a = $(110)_2$ and b = $(101)_2$

$a \cdot b_0 2^0 = (110)_2 \cdot 1 \cdot 2^0 = (110)_2$

$a \cdot b_1 2^1 = (110)_2 \cdot 0 \cdot 2^1 = (0000)_2$

$a \cdot b_2 \cdot 2^2 = (110)_2 \cdot 1 \cdot 2^2 = (11000)_2$

$(110)_2 + (0000)_2 + (11000)_2 = (11110)_2$

Binary Multiplication of Integers

procedure *multiply*(a, b: positive integers)
{the binary expansions of a and b are $(a_{n-1}, a_{n-2},\ldots, a_0)_2$ and $(b_{n-1}, b_{n-2},\ldots, b_0)_2$, respectively}
for $j := 0$ **to** $n - 1$
    if $b_j = 1$ **then** $c_j = a$ with j zeros appended
    else $c_j := 0$
{$c_o, c_1,\ldots, c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ **to** $n - 1$
    $p := p + c_j$
return p

The number of additions of bits used by the algorithm to multiply two n-bit integers is $O(n^2)$.

Example Multiplying $a = (1110)_2$ and $b = (1011)_2$

$$\begin{array}{lrl}
a b_0 2^0 & 1110 & \\
a b_1 2^1 & 11100 & + \\
a b_3 2^3 & 1110000 & \\ \hline
 & 101010 & \\
 & 1110000 & + \\ \hline
 & 10011010 &
\end{array}$$

Binary Modular Exponentiation

In cryptography, it is important to be able to find b^n **mod** m efficiently, where b, n, and m are large integers.

- Use the binary expansion of n, $n = (a_{k-1}, \ldots, a_1, a_0)_2$, to compute b^n.

  Note that:

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}} \cdots b^{a_1\cdot 2}\cdot b^{a_0}.$$

- Therefore, to compute b^n, we need only compute the values of

$$b,\ b^2,\ (b^2)^2 = b^4,\ (b^4)^2 = b^8,\ \ldots,\ b^{2^{k-1}}$$

- and then multiply the terms b^{2^j} in this list, for all $a_j = 1$.

<u>Example</u>

How to compute 3^{11} fast?

Note: $11 = (1011)_2$

$$
\begin{aligned}
3^{11} &= 3^{1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0} = \\
&= 3^{1\cdot 2^3} \cdot 3^{0\cdot 2^2} \cdot 3^{1\cdot 2^1} \cdot 3^{1\cdot 2^0} \\
&= (((3^{2^0})^2)^2)^2 \cdot 1 \cdot (3^{2^0})^2 \quad 3^{2^0} \\
&= 3^8 \quad \cdot 1 \quad 3^2 \quad \cdot 3
\end{aligned}
$$

$$1 \quad 0 \quad 1 \quad 1$$

Note: $(3^{2^k})^2 = 3^{2^k \cdot 2} = 3^{2^{k+1}}$

Example

Example: Compute 3^{11} using this method.

Note that $11 = (1011)_2$ so that

$3^{11} = 3^8\ 3^2\ 3^1 =$

$((3^2)^2\ )^2\ 3^2\ 3^1\ =$

$(9^2\ )^2 \cdot 9 \cdot 3 =$

$(81)^2 \cdot 9 \cdot 3 =$

$6561 \cdot 9 \cdot 3 =$

117,147.

Binary Modular Exponentiation Algorithm

The algorithm successively finds

$$b \textbf{ mod } m,\ b^2 \textbf{ mod } m,\ b^4 \textbf{ mod } m,\ \ldots,\ b^{2^{k-1}} \textbf{ mod } m,$$

and multiplies together the terms b^{2^j} where $a_j = 1$.

procedure *modular exponentiation*(b: integer, $n = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$, m: positive integers)
 x := 1
power := b **mod** m
for i := 0 to $k - 1$
  if a_i= 1 **then** x := ($x \cdot$ *power*) **mod** m
  power := (*power*$\cdot$ *power*) **mod** m
return x

$O((\log m)^2 \log n)$ bit operations are used to find b^n **mod** m.

<u>Example</u> : compute $3^{11} \bmod 5$

Determine binary representation of 11 : $(11)_{10} = (\overset{8}{1}0\overset{2}{1}\overset{1}{1})_2$

Therefore $3^{11} = 3^8 \cdot 3^2 \cdot 3^1$

$$3^1 \bmod 5 = 3$$

$$3^2 \bmod 5 = 4$$

$$3^4 \bmod 5 = 4^2 \bmod 5 = 1$$

$$3^8 \bmod 5 = 1^2 \bmod 5 = 1$$

Therefore $3^{11} \bmod 5 = 3^8 \bmod 5 \cdot 3^2 \bmod 5 \cdot 3^1 \bmod 5 = 1 \cdot 4 \cdot 3 =$

$$= 12 \bmod 5 = 2$$

Summary

- Binary addition, multiplication, modular exponentiation