CS 441: Integer representation and Algorithms

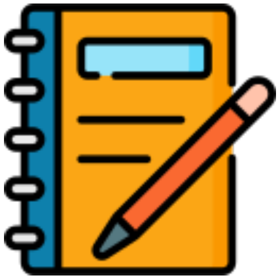
PhD. Nils Murrugarra-Llerena
nem177@pitt.edu

University of Pittsburgh

Today's topics

- Integer representations
  - Base b expansions
  - Common bases: Binary, octal, hexadecimal
  - Base conversions

- Integer algorithms
  - Addition
  - Multiplication
  - Connection to computing and pen-and-paper arithmetic



While we typically use decimal, other base systems work very similarly

Recall: Decimal expansion of integers, e.g.,

$$3528 = 3 * 10^3 + 5 * 10^2 + 2 * 10^1 + 8 * 10^0$$

Theorem: Let b be an integer greater than 1. Any $n \in \mathbf{Z}^+$ can be expressed uniquely in the form:

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b^1 + a_0 b^0$$

where $k \in \mathbf{N}$, each $a_i \in \mathbf{N}$ where $a_i < b$, and $a_k \neq 0$.

This representation is called the base b expansion of n, which we write compactly as $(a_k a_{k-1} \ldots a_1 a_0)_b$

- When $b > 10$, we write each a_i as a single symbol in an extended "alphabet" of digits
  - e.g., 0123456789ABCDEFGH…

Examples of base b expansions

Express each of these expansions in decimal:

- $(675)_8 =$
  $=$
- $(110101)_2 =$
  $=$
- $(2A4)_{12} =$
  $=$

Here, A has value 10

Common base expansions

These base systems are very common in computing:

- Base 2, binary: Expansions are bit strings
  $412 = (110011100)_2$
- Base 8, octal: Each digit a_i is $0 \leq a_i < 8$
  $412 = (634)_8$
- Base 16, hexadecimal: Each digit $a_i \in \{0, 1, \dots, 9, \mathrm{A}, \mathrm{B}, \dots, \mathrm{F}\}$
  $412 = (19\mathrm{C})_{16}$

Trends to note:

- Base b requires b digits in the "alphabet"
- Lesser b yields longer expansions, greater b yields shorter expansions

Why are these important?

- Data is stored in binary, octal represents 3 bits per digit, hexadecimal represents 4 bits per digit

Constructing base b expansions

procedure *base b expansion*(n, b: positive integers with $b > 1$)

$q := n$

$k := 0$

while $q \neq 0$

$a_k := q$ **mod** b

$q := q$ **div** b

$k := k + 1$

return $(a_{k-1}, a_{k-2} \ldots, a_1, a_0)$

$\{(a_{k-1}a_{k-2}\cdots a_1a_0)_b$ is the base b expansion of $n\}$

Digits are produced right-to-left

Repeat: Divide q by b; remainder becomes a digit, quotient replaces q

Return when q = 0

Constructing base b expansions, examples

1. Express 1501 in hex
   - 1501 divided by 16
     q = 93, r = 13 = $(D)_{16}$
   - 93 divided by 16
     q = 5, r = 13 = $(D)_{16}$
   - 5 divided by 16
     q = 0, r = 5
   - Thus, 1501 = (5DD)16

2. Express 441 in octal
   - 441 divided by 8
     q = 55, r = 1
   - 55 divided by 8
     q = 6, r = 7
   - 6 divided by 8
     q = 0, r = 6
   - Thus, 441 = $(671)_8$

3. Express 441 in base-30
   - 441 divided by 30
     q = 14, r = 21 = $(I)_{30}$
   - 14 divided by 30
     q = 0, r = 14 = $(E)_{30}$
   - Thus, 441 = $(EI)_{30}$

4. Express 441 in base-4
   - 441 divided by 4
     q = 110, r = 1
   - 110 divided by 4
     q = 27, r = 2
   - 27 divided by 4
     q = 6, r = 3
   - 6 divided by 4
     q = 1, r = 2
   - 1 divided by 4
     q = 0, r = 1
   - Thus, 441 = $(12321)_4$

When $b = 2^i$, conversion can be done on i bits at a time

Since an octal digit encodes 3 bits and a hex digit encodes 4 bits, we can use binary to help convert

$$(3720)_8$$

$$(011111010000)_2$$

$$(7D0)_{16}$$

In-class exercises

Problem 1: Find the octal expansion of 100

Problem 2: Find the octal expansion of $(100)_2$

Problem 3: Find the octal expansion of $(100)_{16}$

Problem 4: Find the octal expansion of $(100)_{36}$

Adding base b expansions

procedure *add*(x, y: positive integers, b: integer > 1)
{The base b expansions of x and y are $(x_{n-1}x_{n-2}\cdots x_1x_0)_b$ and $(y_{n-1}y_{n-2}\cdots y_1y_0)_b$, respectively}
$c := 0$
for $j := 0$ **to** $n-1$ {Move right-to-left}
$t := x_j + y_j + c$ {Add the jth digits together}
$c := \lfloor t/b \rfloor$ {Carry a digit if needed}
$s_j := t - bc$ {Remove carry and save as s_j}
$s_n := c$ {Final carry becomes s_n}
return $(s_n, s_{n-1}, \ldots, s_1, s_0)$
{The base b expansion of the sum is $(s_ns_{n-1}\cdots s_1s_0)_b$}

Does this sound familiar?
What is its complexity?

Addition examples in hexadecimal/octal

Hex | Octal

```
  B8C0              5630
+ 827F            + 3766
```

```
  13AC4F            723405
+ 3B9E00          +  27305
```

Multiplying base b expansions

```
procedure multiply(x, y: positive integers, b: integer > 1)
    {The base b expansions of x and y are (x_{n-1}x_{n-2}···x_1x_0)_b and (y_{n-1}y_{n-2}···y_1y_0)_b, respectively}
    p := 0                          {Resulting product}
    for j := 0 to n–1               {Move right-to-left in y}
        c := 0                      {Reset carry}
        for i := 0 to n-1           {Move right-to-left in x}
            t := x_i * y_j + c      {Multiply digits and add carry}
            c := ⌊t/b⌋              {Carry a digit if needed}
            r_i := t - bc           {Partial product, digit i}
        r_n := c                    {Final carry becomes r_n}
        r := r shifted j places     {Shift position to align with j}
        p := add(p, r)              {Add r to the result}
    return (p_{2n}, p_{2n-1}, …, p_1, p_0)
    {The base b expansion of the sum is (p_{2n}p_{2n-1}···p_1p_0)_b}
```

What is its complexity?

Multiplication examples in hexadecimal/octal

Hex | Octal

```
  C38        365
* 6A4      * 457
-----      -----



-----      -----
```

How are these algorithms used in practice?

In previous exercises, didn't we assume basic arithmetic operations were $\Theta(1)$?

- This is often true! Modern CPUs can compute (at least) 32-bit integer multiplication in circuitry in a few cycles
- What about numbers bigger than your CPU's MUL?
  - e.g., for cryptography
- Let $b = 2^{32}$, consider b-bit expansions where each "digit" is a 32-bit word

We can compare a CPU's MUL (etc.) circuits to the multiplication tables we memorized in grade school

- For small enough values, we know the answer very quickly
- For larger values, we learn an algorithm that utilizes many smaller multiplications

Other multiplication algorithms for even larger values

Algorithm	Complexity	Threshold example*
Grade school	$O(n^2)$	(native MUL)
Karatsuba	$O\left(n^{\log_2 3}\right) \approx O(n^{1.585})$	832 bits
Toom–Cook (Toom-3)	$O\left(n^{\log_3 5}\right) \approx O(n^{1.46})$	6208 bits
Schönhage–Strassen	$O(n \log n \log \log n)$	159744 bits
Fürer	$O\left(n \log n \, 2^{\Theta(\log^* n)}\right)$	—
Harvey–van der Hoeven	$O(n \log n)$	—

In-class exercises

Problem 5: Use the integer addition algorithm to compute $(734)_8 + (225)_8$

Problem 6: Use the integer multiplication algorithm to compute $(110110)_2 \times (100101)_2$

Problem 7: Calculate $(FF)_{16} \times (FF)_{16}$, $(77)_8 \times (77)_8$, and 99×99 and compare

Final thoughts

- Integers can be represented uniquely in any specified base
- Integer arithmetic can be computed in other bases, and even pen-and-paper algorithms can be useful in computing
  - Arithmetic isn't always constant
- Next time:
  - Primes and composites (Section 4.3)