Why Representation?

- Need to map numbers to binary bits
  - Need a representation anyway...
- Representation is consequential
  - Complexity of arithmetic operations depends heavily on the representation

Number representation is the heart of computer arithmetic...

What Is a Number System?

- A number is represented as an ordered n-tuple of symbols (digit vector)
  \[ \langle a_{n-1}a_{n-2}...a_2a_1a_0 \rangle \]
- Each symbol is a digit
  \[ a_{n-1}, a_{n-2}, ..., a_2, a_1, a_0 \]
- Digits usually represent integers from a given set—e.g.,
  \[ a_i \in \{0, 1\} \text{ or } a_i \in \{0, 1, ..., 9\} \text{ or } a_i \in \{-1, 0, 1\} \]
Redundancy

- Nonredundant digit vectors \( N, Z, R, \ldots \)
- Redundant digit vectors \( N, Z, R, \ldots \)

Positional Weighted Systems

- The rule of interpretation is a scalar product
  \[
  A = \left( a_{n-1}a_{n-2} \ldots a_2a_1a_0 \right) = \sum_{i=0}^{n-1} a_i w_i
  \]
  where
  \[
  W = \left( w_{n-1}, w_{n-2}, \ldots, w_2, w_1, w_0 \right)
  \]
  is the weight vector

Radix Systems

- Weights are not arbitrary but related to a radix vector
  \[
  R = \left( r_{n-1}, r_{n-2}, \ldots, r_2, r_1, r_0 \right)
  \]
  in the following way
  \[
  w_0 = 1 \quad w_i = w_{i-1} \cdot r_{i-1} = \prod_{j=0}^{i-1} r_j
  \]

Mixed-Radix Systems

- Fixed-radix if all elements of \( R \) are identical
  \[
  w_i = r^i \quad \Rightarrow \quad A = \sum_{i=0}^{n-1} a_i r^i
  \]
- A few mixed-radix systems are very common—e.g., time
  \[
  R = (24, 60, 60)
  \]
  \[
  W = (86400, 3600, 60, 1)
  \]
Common Decimal Notation

- It is **weighted**
  \[ A = \sum_{i=0}^{n-1} a_i w_i \]

- It is **positional**
  \[ w_i \text{ depends only on } i \]

- It is **fixed-radix**
  \[ w_i = r^i \]

Digit Set

- **Canonical** set
  \[ a_i \in \{0, 1, \ldots, r-1\} \]
  A canonical system is nonredundant

- Any other choice is noncanonical
  \[ a_i \in \{-1, 0, 1, 2\} \]

Why Redundant Representations?

- Remember paper & pencil addition?

  - In binary:
    - Slow—critical path contains \( n \) full-adders: naïve delay is \( O(n) \)
    - Must be performed right to left (LSB to MSB)
Redundancy May Simplify Some Operations

- Radix-2, noncanonical \{0, 1, 2, 3\}, 2 bits per digit

<table>
<thead>
<tr>
<th>Digit</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>2</td>
<td>1 1 1</td>
</tr>
<tr>
<td>3</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>

Critical path now contains only two fulladders: delay is now \(O(1)\)

Redundancy and MSB-first?

- Now that the carry chain is broken, output digit \(i\) depends exclusively on input digits \(i, i - 1, i - 2\)
- Hence I can receive the operands in serial form and, when I have seen the first three input digits, I can produce the first output digit; afterwards, every new input digit, I can produce a new output digit

Online arithmetic or MSDF digit-serial arithmetic

Very Large Choice of Weighted Positional Number Systems

Binary Representation and Negative Numbers

- Simplest signed system: Sign-and-magnitude
  - One bit represents the sign (typ. 1 \(\to\) neg.)
  - Absolute value represented as unsigned

\[
A = \left\langle s a_{n-2} \ldots a_2 a_1 a_0 \right\rangle = (-1)^s \sum_{i=0}^{n-2} a_i 2^i
\]

- Exactly as humans do in decimal

\[
A = \left\langle \pm a_{n-2} \ldots a_2 a_1 a_0 \right\rangle = \pm \sum_{i=0}^{n-2} a_i 10^i
\]

Of course, we could also write 0 for + and 9 for –...
**Sign-and-Magnitude Representation**

- Some advantages and disadvantages
  - Familiar for users
  - Simple naïve multiplication
  - Adders are not the most efficient
  - Redundant zero (+0 and –0) may cause some problems (e.g. testing for a variable =0)

**Negative Numbers As Result of Paper & Pencil Subtraction**

- Consider a normal paper-and-pencil subtraction

\[
\begin{array}{cccccc}
-1 & -1 & -1 & -1 \\
0 & 0 & 0 & 1 & 0 & 1 & 0_2 \\
-0 & 0 & 0 & 1 & 0 & 0 & 0_2 \\
\end{array}
\rightarrow
\begin{array}{c}
10_{10} \\
17_{10} \\
\end{array}
\]

\[
\begin{array}{cccccccccc}
... & 1 & 1 & 1 & 1 & 0 & 0 & 1_2 \\
-1 & 1 & 1 & 1 & 1 & 0 & 0 & 1_2 \\
-2^7 & +2^6 & +2^5 & +2^4 & +2^3 & +2^0 & \rightarrow & -7_{10} \\
\end{array}
\]

**Intuitive Derivation of the Two’s Complement System**

- Two’s complement representation
  \[A = \langle a_{n-1}a_{n-2}...a_2a_1a_0 \rangle = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i2^i\]

- Largest (positive): all ones except \(a_{n-1} = 0\)
  \[A_{\text{max}} = 2^{n-1} - 1\]

- Smallest (negative): all zeros except \(a_{n-1} = 1\)
  \[A_{\text{min}} = -2^{n-1}\]

- Asymmetric range, non-redundant
  - Standard in computers for signed integers nowadays

**Comparison of Binary Representations on 3 Bits**

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (Unsigned)</th>
<th>Sign and Magnitude</th>
<th>Two's Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>000 / 100</td>
<td>000</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td>101</td>
<td>111</td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
Two's Complement Representation of Negative Numbers

- It "shifts" the second half of the unsigned range into the negative by $2^n$
- E.g., on 8 bits:

$\begin{align*}
00000000 & \quad 01111111 & \quad 10000000 & \quad 11111111 \\
0 & \quad 127 & \quad 128 & \quad 255
\end{align*}$

that is, $-A$ is represented as the unsigned integer $2^n - A$

True-and-Complement Systems for Signed Numbers

- Transform signed numbers in unsigned and then use conventional systems

Mapping from Signed to Unsigned

- A number $A$ is represented through the unsigned number $A_R$ defined through a complementation constant

$$A_R = A \mod C$$

- This is equivalent (for $|A| < C$) to

$$A_R = \begin{cases} 
A & \text{if } A \geq 0 \\
C - |A| = C + A & \text{if } A < 0 
\end{cases}$$

- For the representation to be unambiguous, it must be

$$|A| < C / 2$$

or at most equal...

Inverse Mapping

- Inversely

$$A = \begin{cases} 
A_R & \text{if } A_R < C / 2 \\
A_R - C & \text{if } A_R > C / 2 
\end{cases}$$

- If $A_R = C / 2$ can be represented, it can be assigned either to $A = -C / 2$ or to $A = C / 2$, but the representation is then asymmetric ($\rightarrow$ system not close under sign change operation)

- If $A_R = C$ is can be represented, there are two representations for zero
Range Complement and Digit Complement Systems

- Two particular cases of true-and-complement systems
  - Range complement: \( C = r^n \)
  - Digit complement: \( C = r^n - 1 \)

In radix-2

- Two’s complement: \( C = 2^n \)
- One’s complement: \( C = 2^n - 1 \)

Two’s complement is asymmetric and nonredundant
One’s complement is symmetric with a redundant zero

Comparison of More Binary Representations on 3 Bits

<table>
<thead>
<tr>
<th></th>
<th>Decimal</th>
<th>Binary (Unsigned)</th>
<th>Sign and Magnitude</th>
<th>Two’s Complement</th>
<th>One’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td>111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>110</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>011</td>
<td>111</td>
<td>011</td>
<td>011</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>010</td>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>001</td>
<td>001</td>
<td>001</td>
<td>001</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>000 / 100</td>
<td>000 / 111</td>
<td>000 / 111</td>
<td>000 / 111</td>
</tr>
<tr>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td></td>
<td></td>
<td></td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Similar Property for One’s Complement

Using the true-and-complement definition, if \( A_R < C / 2 \)

\[
\begin{align*}
A &= A_R \\
A_{n-1} &= 1
\end{align*}
\]

Hence the property

\[
A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i
\]

Equivalence of the Two Two’s Complement Definitions

Using the true-and-complement definition, if \( A_R < C / 2 \)

\[
\begin{align*}
A &= A_R \\
a_{n-1} &= 0 \quad \Rightarrow \quad A = \sum_{i=0}^{n-1} a_i 2^i
\end{align*}
\]

If \( A_R \geq C / 2 \)

\[
\begin{align*}
A &= A_R - C \\
a_{n-1} &= 1
\end{align*}
\]

Hence the property

\[
A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i
\]
Sign

$$\text{sign}(A) = \begin{cases} 
0 & \text{if } A \geq 0 \\
1 & \text{if } A < 0 
\end{cases}$$

- For sign-and-magnitude is clearly trivial...
- For true-and-complement too, it is

$$\text{sign}(A) = \begin{cases} 
0 & \text{if } A_R < C / 2, \text{ that is if } a_{n-1} = 0 \\
1 & \text{if } A_R \geq C / 2, \text{ that is if } a_{n-1} = 1 
\end{cases}$$

- For all these representations

$$\text{sign}(A) = a_{n-1}$$

Implied Digits in the Unsigned Representation

- Signed numbers (like naturals in decimal) have infinite leading zeros in front of them:

$$1\,234_{10} \rightarrow \ldots00\,001\,234_{10}$$

$$1\,0101_{2} \rightarrow \ldots0\,0000\,0001\,0101_{2}$$

- Changing the representation from \( n \) bits to \( m \) bits \((m > n)\) is just a matter of padding the number with leading zeros

Changing the Number of Bits in One’s and Two’s Complement

- Signed numbers can be thought as having infinite replicas of the MSB/sign in front of them

$$1\,1\,0\,1_{2} \rightarrow -3_{10}$$

$$1\,1\,1\,1\,1\,1\,1\,0\,1_{2} \rightarrow -3_{10}$$

- “Minus three” on 4 bits is not represented as “minus three” on 8 bits!

Sign Extension and Shift Right

- Two’s complement representation on \( n \) bits

$$A^{(n)} = \langle a_{n-1} a_{n-2} \ldots a_{2} a_{1} a_{0} \rangle = -a_{n-1} 2^{n-1} + \sum_{i=0}^{n-2} a_{i} 2^{i}$$

- Sign extended two’s complement representation on \( m \) bits \((m > n)\)

$$A^{(m)} = \langle a_{n-1} \ldots a_{n-1} a_{n-2} \ldots a_{2} a_{1} a_{0} \rangle =$$

$$= -a_{n-1} 2^{m-1} + \sum_{j=n-1}^{m-2} a_{n-1} 2^{j} + \sum_{i=0}^{n-2} a_{i} 2^{i}$$

- To extend from \( n \) to \( m \) bits, one adds (for positive #) zero or (for negative #)

$$(-2^{m-1} + 2^{m-2} + \ldots + 2^{n-1}) - (-2^{n-1}) \equiv 0$$
Addition in True-and-Complement

- If there is no overflow (that is, we assume the result of the sum to be representable), \( S = A + B \) can be simply computed as

\[
S_R = (A_R + B_R) \mod C
\]

- Easy to prove using \((X \mod Y) \mod Y = X \mod Y\)

\[
(A_R + B_R) \mod C = (A \mod C + B \mod C) \mod C = (A + B) \mod C = S \mod C = S_R
\]

Two’s Complement Addition

- Particularly comfortable in two’s complement, because \( C = 2^n \) and performing \( \mod 2^n \) implies just to ignore the carry out bit

\[
C_{out}
\]

One’s Complement Addition

- A little more complex in one’s complement, because \( C = r^n - 1 \) and the modulo of \( W_R = A_R + B_R \) is less easy

\[
w_s = 0 \quad \text{and} \quad W_s \mod (r^n - 1) = W_s \quad \text{if} \quad A_s + B_s < r^n - 1
\]

\[
w_s = 0 \quad \text{and} \quad W_s \mod (r^n - 1) = 0 \quad \text{if} \quad A_s + B_s = r^n - 1
\]

\[
w_s = 1 \quad \text{and} \quad W_s \mod (r^n - 1) = W_s - r^n + 1 \quad \text{if} \quad r^n - 1 < A_s + B_s \leq 2(r^n - 1)
\]

- Hence

If \( w_s = 0 \) \( \Rightarrow \) \( S_s = W_s \)

If \( w_s = 1 \) \( \Rightarrow \) \( S_s = W_s + 1 \), subtracting \( r^n = 2^* \) simply by ignoring \( w_s \).
Negation

- Bitwise negation
  \[ \overline{A} = \text{not}(A) = \left\{ \overline{a}_{n-1} \ a_{n-2} \ldots \ a_2 \ a_1 \ a_0 \right\} \]

  where, as usual in binary,
  \[ \overline{a} = (r - 1) - a \]

- Example in radix-2 on 4 bits:
  \[ \overline{6}_{10} = \text{not}(6_{10}) = \langle \overline{0 \ 1 \ 1 \ 0} \rangle = \langle 1001 \rangle = 1001_2 = 9_{10} \]

Complement and Negation in True-and-Complement

- Fundamental property in \( n \)-digit \( r \)-radix systems
  \[ A + \overline{A} + 1 = r^n \]

  because
  \[ \begin{align*}
  a_{n-1} & \ a_{n-2} \ldots \ a_2 \ a_1 \ a_0 \\
  \overline{a}_{n-1} & \overline{a}_{n-2} \ldots \overline{a}_2 \overline{a}_1 \overline{a}_0 \\
  r - 1 & \ r - 1 \ldots \ r - 1 \ r - 1 \ r - 1 \\
  1 & = 1
  \end{align*} \]

  \[ \begin{align*}
  1 & 0 \ 0 \ldots \ 0 \ 0 \ 0 \ 0 \\
  r^n & \end{align*} \]

Change of Sign

- In one’s complement
  \[ -A = \overline{A} \]

  \[ -A = r^n - 1 - A = \overline{A} \quad \text{if} \ A \geq 0 \\
  \overline{A} = r^n - 1 + A = r^n - (r^n - 1 + A) - 1 = -A \quad \text{if} \ A < 0 \]

- In two’s complement
  \[ -A = \overline{A} + 1 \]

  \[ -A = r^n - A = \overline{A} + 1 \quad \text{if} \ A \geq 0 \\
  \overline{A} + 1 = r^n + A + 1 = r^n - (r^n + A) - 1 + 1 = -A \quad \text{if} \ A < 0 \]

Change of Sign in Two’s Complement

- Another way of verifying the property
  \[ A + \overline{A} = -1 \]

- Use two’s complement property/definition
  \[ \left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \right) + \left(-\overline{a}_{n-1}2^{n-1} + \sum_{i=0}^{n-2} \overline{a_i} 2^i \right) = \]

  \[ = -\left(a_{n-1} + \overline{a}_{n-1}\right)2^{n-1} + \sum_{i=0}^{n-2} \left(a_i + \overline{a_i}\right)2^i = -2^{n-1} + \sum_{i=0}^{n-2} 2^i = -1 \]
Two’s Complement Subtractor

\[ A - B = A + (-B) = A + \overline{B} + 1 \]

Two’s Complement Adder/Subtracter

Overflow?!?

Overflows for Unsigned Addition

- Limited range, thus sum of two numbers can exceed the range

\[
\begin{align*}
1 & \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1_2 \rightarrow 201_{10} \\
+ & \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0_2 \rightarrow 200_{10} \\
1 & \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1_2 \rightarrow 401_{10} (>255_{10})
\end{align*}
\]

- **Overflows if carry out is 1**—that is, if the first bit which cannot be stored is not null

Overflows for Signed Addition

- The situation is somehow different and the same simple rule does not hold anymore

\[
\begin{align*}
1 & \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1_2 \rightarrow -55_{10} \\
+ & \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0_2 \rightarrow -56_{10} \\
1 & \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1_2 \rightarrow -111_{10} \\
1 & \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1_2 \rightarrow -111_{10}
\end{align*}
\]

- **Carry out is 1**, but now it is ok because the result is negative (think of sign extension...)

© Ienne, Oklobdzija 2004
Overflows for Signed Addition

- Overflows makes sum of positives look like negative or sum of negatives look like positive

\[
\begin{align*}
90_{10} &= 0101 1010_2 \\
105_{10} &= 0110 1001_2 \\
195_{10} &= 1100 0011_2 \\
-61_{10} &= 1111 1000_2
\end{align*}
\]

Overflows Addition and Subtraction

- No overflow possible if adding operands of different sign
- Overflow only if

\[
\begin{align*}
A + B &\geq 0 \quad A - B \geq 0 \\
A + B &< 0 \quad A - B < 0
\end{align*}
\]

\[
OVF_+ = a_{n-1} b_{n-1} c_n \oplus c_{n-1}
\]

Another View on Overflow for Signed Addition

- Sign extend the operands and check if the additional result bit is equal to the previous one (sign "unextension"...)

\[
\begin{align*}
1 1 1 0 0 0 1 0 &\rightarrow -55_{10} \\
+ 1 1 1 0 0 0 1 0 &\rightarrow -56_{10}
\end{align*}
\]

\[
\begin{align*}
1 1 0 0 1 0 0 &\rightarrow -111_{10} \\
1 0 0 1 0 0 1 &\rightarrow -111_{10}
\end{align*}
\]

\[
OVF_+ = (a_n \oplus b_n \oplus c_n) \oplus (a_{n-1} \oplus b_{n-1} \oplus c_{n-1}) =
= (a_{n-1} \oplus b_{n-1} \oplus c_n) \oplus (a_{n-1} \oplus b_{n-1} \oplus c_{n-1}) =
= c_n \oplus c_{n-1}
\]
**Fixed Point**

- In real life, one does not only need integers...
- If one adds a fractional point in a fixed position, hardware for integers works just as well

<table>
<thead>
<tr>
<th>2^4</th>
<th>2^3</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1_2</td>
</tr>
</tbody>
</table>

\[0.1001_2 \rightarrow 9_{10}\]

\[0.1001_2 \rightarrow 1.125_{10}\]

+ \[0.0011_2 \rightarrow 3_{10}\]

\[0.0011_2 \rightarrow 0.375_{10}\]

\[0.1100_2 \rightarrow 12_{10}\]

\[0.1100_2 \rightarrow 1.500_{10}\]

- It’s just a matter of representation!

**Fixed Point Requires Attention from Programmers**

- Multiplication often introduces the need of arithmetic shifts to the right

\[0.1010_2 \rightarrow 0.625_{10}\]

\[0.0111_2 \rightarrow 0.375_{10}\]

\[\times\]

\[\rightarrow \quad 0.011110_2 \rightarrow 0.234375_{10}\]

\[\rightarrow \quad 0.001110_2 \rightarrow 0.1875_{10}\]

- Note the low accuracy!

**References**

- M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann, 2004