Digital Logic Design: a rigorous approach ©

Chapter 5: Binary Representation

Guy Even   Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

March 22, 2020

Book Homepage:
http://www.eng.tau.ac.il/~guy/Even-Medina

Division and Modulo

Definition

Given $a \in \mathbb{Z}$ and $b \in \mathbb{Z}^+$ $(b > 0)$ define:

$$(a \div b) \triangleq \max\{q \in \mathbb{Z} \mid q \cdot b \leq a\}$$
$$\mathrm{mod}(a, b) \triangleq a - b \cdot (a \div b).$$

- $(a \div b)$ is called the quotient and $\mathrm{mod}(a, b)$ is called the remainder.
- if $\mathrm{mod}(a, b) = 0$, then a is a multiple of b (a is divisible by b).
- $(a \div b) = \left\lfloor \frac{a}{b} \right\rfloor$.
- $(a \bmod b), \mathrm{mod}(a, b), a \pmod{b}$ denote the same thing.

Examples

1. $3 \bmod 5 = 3$ and $5 \bmod 3 = 2$.
2. $999 \bmod 10 = 9$ and $123 \bmod 10 = 3$.
3. $a \bmod 2$ equals 1 if a is odd, and 0 if a is even.
4. $a \bmod b \geq 0$.
5. $a \bmod b \leq b - 1$.

Division & Mod are Well Defined

Claim

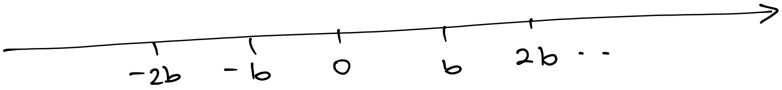
$\mathrm{mod}(a, b) \in \{0, 1 \ldots, b-1\}$.

Claim

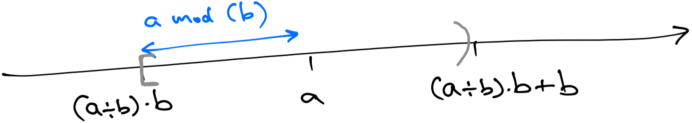
If $a = q \cdot b + r$ *and* $0 \leq r \leq b - 1$, *then*

$$q = a \div b$$
$$r = a \pmod{b} .$$

Claim: $a \pmod{b}$ $(b>0)$



$$a \div b = \max\{q \in \mathbb{Z} : q \cdot b \le a\}$$



$\Rightarrow$ 1) $a \pmod{b} \geq 0$

2) $a \pmod{b} \leq b-1$ $\boxtimes$

Q: $-11 \bmod 6 = (-12+1) \bmod 6 = 1$

Claim: if $a = q \cdot b + r \quad 0 \le r \le b-1$

$$\Rightarrow \quad q = a \div b$$
$$r = a \pmod{b}$$

proof: we want to prove that if

$a = q_1 \cdot b + r_1 = q_2 \cdot b + r_2$ where $r_1, r_2 \in [0, b)$,

then $(q_1, r_1) = (q_2, r_2)$.

WLOG, $r_2 \ge r_1$. subtraction $\Rightarrow$

$$0 = (q_2 - q_1) \cdot b + (r_2 - r_1) \qquad r_2 - r_1 \in [0, b).$$

now: 0 & $(q_2 - q_1) \cdot b$ are divisible by b.

$\Rightarrow (r_2 - r_1)$ is divisible by b

$\Rightarrow \quad r_2 - r_1 = 0 \quad \Rightarrow \quad r_1 = r_2 \quad \Rightarrow \quad q_1 = q_2. \quad \boxtimes$

Modular Addition

Lemma

For every $z \in \mathbb{Z}$,

$$x \bmod b \quad = \quad (x + z \cdot b) \bmod b$$

Lemma

$$((x \bmod b) + (y \bmod b)) \bmod b \quad = \quad (x + y) \bmod b$$

$\forall z \in \mathbb{Z} \quad a \pmod b = (a + z \cdot b) \pmod b$

<u>proof</u>: let $q \triangleq a \div b$, $r \triangleq a \pmod b$

hence: $a = q \cdot b + r \quad 0 \leq r \leq b-1$

consider $a + z \cdot b$:

$$a + z \cdot b = q \cdot b + r + z \cdot b = (q+z)b + r$$

prev. claim $\Longrightarrow$
$$\begin{cases} (a+zb) \div b = q+z \\ (a+z \cdot b) \pmod b = r \end{cases}$$

$\boxtimes$

claim: $\left(\underbrace{(x \bmod b)}_{r_x} + \underbrace{(y \bmod b)}_{r_y}\right) \pmod{b}$

$= (x+y) \pmod{b}$

proof: divide x & y by b:

$$\begin{cases} x = q_x \cdot b + r_x \\ y = q_y \cdot b + r_y \end{cases} \qquad r_x, r_y \in [0, b-1]$$

$$x+y = (q_x+q_y)\cdot b + r_x + r_y$$

prev. claim $\Longrightarrow$ $z = q_x + q_y$

$$(x+y) \bmod b = (r_x + r_y) \bmod b$$

$\boxtimes$

Definition

A binary string is a finite sequence of bits.

Ways to denote strings:

1. sequence $\{A_i\}_{i=0}^{n-1}$,
2. vector $A[0:n-1]$, or
3. $\vec{A}$ if the indexes are known.

We often use $A[i]$ to denote A_i.

Example

- $A[0:3] = 1100$ means $A_0 = 1$, $A_1 = 1$, $A_2 = 0$, $A_3 = 0$.
- The notation $A[0:5]$ is zero based, i.e., the first bit in $\vec{A}$ is $A[0]$. Therefore, the third bit of $\vec{A}$ is $A[2]$ (which equals 0).

$A[3:0]$

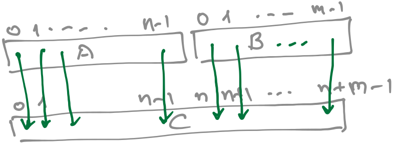
3	2	1	0
1	1	0	0

concatenation

A basic operation that is applied to strings is called concatenation. Given two strings $A[0 : n-1]$ and $B[0 : m-1]$, the concatenated string is a string $C[0 : n+m-1]$ defined by

$$C[i] \triangleq \begin{cases} A[i] & \text{if } 0 \leq i < n, \\ B[i-n] & \text{if } n \leq i \leq n+m-1. \end{cases}$$

We denote the operation of concatenating string by $\circ$, e.g., $\vec{C} = \vec{A} \circ \vec{B}$.



Example

Examples of concatenation of strings. Let $A[0:2] = 111$, $B[0:1] = 01$, $C[0:1] = 10$, then:

$$\vec{A} \circ \vec{B} = 111 \circ 01 = 11101\,,$$
$$\vec{A} \circ \vec{C} = 111 \circ 10 = 11110\,,$$
$$\vec{B} \circ \vec{C} = 01 \circ 10 = 0110\,,$$
$$\vec{B} \circ \vec{B} = 01 \circ 01 = 0101\,.$$

bidirectionality (MSB first / LSB first)

Let $i \leq j$. Both $A[i:j]$ and $A[j:i]$ denote the same sequence $\{A_k\}_{k=i}^{j}$. However, when we write $A[i:j]$ as a string, the leftmost bit is $A[i]$ and the rightmost bit is $A[j]$. On the other hand, when we write $A[j:i]$ as a string, the leftmost bit is $A[j]$ and the rightmost bit is $A[i]$.

Example

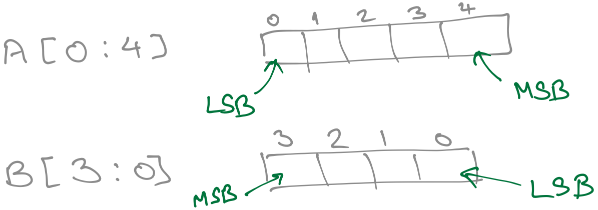
The string $A[3:0]$ and the string $A[0:3]$ denote the same 4-bit string. However, when we write $A[3:0] = 1100$ it means that $A[3] = A[2] = 1$ and $A[1] = A[0] = 0$. When we write $A[0:3] = 1100$ it means that $A[3] = A[2] = 0$ and $A[1] = A[0] = 1$.

big endian vs. little endian...

Definition

The least significant bit of the string $A[i:j]$ is the bit $A[k]$, where $k \triangleq \min\{i,j\}$. The most significant bit of the string $A[i:j]$ is the bit $A[\ell]$, where $\ell \triangleq \max\{i,j\}$.

The abbreviations LSB and MSB are used to abbreviate the least significant bit and the most significant bit, respectively.



LSB/MSB - examples

1. The least significant bit (LSB) of $A[0:3] = 1100$ is $A[0] = 1$. The most significant bit (MSB) of $\vec{A}$ is $A[3] = 0$.
2. The LSB of $A[3:0] = 1100$ is $A[0] = 0$. The MSB of $\vec{A}$ is $A[3] = 1$.
3. The least significant and most significant bits are determined by the indexes. In our convention, it is not the case that the LSB is always the leftmost bit. Namely, if $i \leq j$, then LSB in $A[i:j]$ is the leftmost bit, whereas in $A[j:i]$, the leftmost bit is the MSB.

Binary Representation

We are now ready to define the binary number represented by a string $A[n-1:0]$.

Definition

The natural number, a, represented in binary representation by the binary string $A[n-1:0]$ is defined by

$$a \triangleq \sum_{i=0}^{n-1} A[i] \cdot 2^i.$$

In binary representation, each bit has a weight associated with it. The weight of the bit $A[i]$ is 2^i.

$A[0], A[1], \ldots, A[n-1]$: coefficients of power series $2^0, 2^1, \ldots, 2^{n-1}$

Notation

Consider a binary string $A[n-1:0]$. We introduce the following notation:

$$\langle A[n-1:0] \rangle \triangleq \sum_{i=0}^{n-1} A[i] \cdot 2^i.$$

To simplify notation, we often denote strings by capital letters (e.g., A, B, S) and we denote the number represented by a string by a lowercase letter (e.g., a, b, and s).

Consider the strings: $A[2:0] \triangleq 000, B[3:0] \triangleq 0001$, and $C[3:0] \triangleq 1000$. The natural numbers represented by the binary strings A, B and C are as follows.

$$\begin{aligned}
\langle A[2:0]\rangle &= A[0]\cdot 2^0 + A[1]\cdot 2^1 + A[2]\cdot 2^2\\
&= 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 = 0\,,\\
\langle B[3:0]\rangle &= B[0]\cdot 2^0 + B[1]\cdot 2^1 + B[2]\cdot 2^2 + B[3]\cdot 2^3\\
&= 1\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 0\cdot 2^3 = 1\,,\\
\langle C[3:0]\rangle &= C[0]\cdot 2^0 + C[1]\cdot 2^1 + C[2]\cdot 2^2 + C[3]\cdot 2^3\\
&= 0\cdot 2^0 + 0\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3 = 8\,.
\end{aligned}$$

Leading Zeros

Consider a binary string $A[n-1:0]$. Extending $\vec{A}$ by leading zeros means concatenating zeros in indexes higher than $n-1$. Namely,

1. extending the length of $A[n-1:0]$ to $A[m-1:0]$, for $m > n$, and
2. defining $A[i] = 0$, for every $i \in [m-1:n]$.

Example

$$A[2:0] = 111$$
$$B[1:0] = 00$$
$$C[4:0] = B[1:0] \circ A[2:0] = 00 \circ 111 = 00111.$$

Leading Zeros

The following lemma states that extending a binary string by leading zeros does not change the number it represents in binary representation.

Lemma

Let $m > n$. If $A[m-1:n]$ is all zeros, then $\langle A[m-1:0]\rangle = \langle A[n-1:0]\rangle$.

Example

Consider $C[6:0] = 0001100$ and $D[3:0] = 1100$. Note that $\langle \vec{C} \rangle = \langle \vec{D} \rangle = 12$. Since the leading zeros do not affect the value represented by a string, a natural number has infinitely many binary representations.

claim: $A[m-1:n] = 0^{m-n} \Rightarrow \langle A[m-1:0]\rangle = \langle A[n-1:0]\rangle$

proof:

$$\langle A[m-1:0]\rangle \overset{def}{=} \sum_{i=0}^{m-1} A[i]\cdot 2^i$$

$$= \sum_{i=0}^{n-1} A[i]\cdot 2^i + \sum_{i=n}^{m-1} A[i]\cdot 2^i$$

$$= \langle A[n-1:0]\rangle + 0$$

$\boxtimes$

Representable Ranges

The following lemma bounds the value of a number represented by a k-bit binary string.

Lemma

Let $A[k-1:0]$ denote a k-bit binary string. Then,

$$0 \leq \langle A[k-1:0] \rangle \leq 2^k - 1 \ .$$

What is the largest number representable by the following number of bits: (i) 8 bits, (ii) 10 bits, (iii) 16 bits, (iv) 32 bits, and (v) 64 bits?

$2^8 = 256$ $\quad 2^{16} = 65,536$ $\quad 2^{64} \cong 1.8 \cdot 10^{19}$

$2^{10} = 1024$ $\quad 2^{32} = 4$ gigabit $\approx 4.29 \cdot 10^9$

claim: $0 \le \langle A[K-1:0] \rangle \le 2^K - 1$

proof: $\langle \vec{A} \rangle \ge 0$ easy.

$$\langle A[K-1:0] \rangle \overset{\text{def}}{=} \sum_{i=0}^{K-1} A[i] \cdot 2^i$$

$$\underset{A[i] \le 1}{\le} \sum_{i=0}^{K-1} 2^i \underset{\substack{\text{geom.} \\ \text{series}}}{=} \frac{2^K - 1}{2 - 1}$$

$\boxtimes$

Computing a Binary Representation

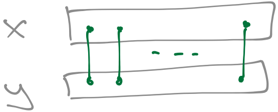
Fix k the number of bits (i.e., length of binary string).
Goals:

1. show how to compute a binary representation of a natural number using k bits.
2. prove that every natural number in $[0, 2^k - 1]$ has a unique binary representation that uses k bits.

1) $x := 13$

$\langle \boxed{0\,0\,0\,0\,1\,1\,0\,1} \rangle = 13$

2) if $x == y$ then



binary representation algorithm: specification

Algorithm $BR(x, k)$ for computing a binary representation is specified as follows:

Inputs: $x \in \mathbb{N}$ and $k \in \mathbb{N}^+$, where x is a natural number for which a binary representation is sought, and k is the length of the binary string that the algorithm should output.

Output: The algorithm outputs "fail" or a k-bit binary string $A[k-1:0]$.

Functionality: The relation between the inputs and the output is as follows:

1. If $0 \leq x < 2^k$, then the algorithm outputs a k-bit string $A[k-1:0]$ that satisfies $x = \langle A[k-1:0] \rangle$.
2. If $x \geq 2^k$, then the algorithm outputs "fail".

binary representation algorithm

Algorithm 1 $BR(x, k)$ - An algorithm for computing a binary representation of a natural number a using k bits.

1. Base Cases:
   1. If $x \geq 2^k$ then return (fail).
   2. If $k = 1$ then return (x).
2. Reduction Rule:
   1. If $x \geq 2^{k-1}$ then return $(1 \circ BR(x - 2^{k-1}, k - 1))$.
   2. If $x \leq 2^{k-1} - 1$ then return $(0 \circ BR(x, k - 1))$.

example: execution of $BR(2, 1)$ and $BR(7, 3)$

Theorem

If $x \in \mathbb{N}$, $k \in \mathbb{N}^+$, and $x < 2^k$, then algorithm $BR(x, k)$ returns a k-bit binary string $A[k-1:0]$ such that $\langle A[k-1:0] \rangle = x$.

$BR(2, 1)$ $\left[\begin{array}{l} x = 2 \\ k = 1 \end{array}\right.$

$2 \geq 2^1$: fail !

Indeed : $2 > 2^1 - 1 = 1$ (out of range)

$BR(7,3)$

$x=7$

$k=3$

$7 \geq 2^3$ No!

$3=1$ No!

$7 \geq 2^{3-1} = 4$ yes;

$A[2]=1$

$BR(7-2^2, 3-1)$

$BR(3,2)$

$3 \geq 2^2$ No! $2=1$ No!

$3 \geq 2^{2-1} = 2$ yes!

$A[1]=1$

$BR(3-2^1, 1)$

$BR(1,1)$ $k=1$ return $A[0]=1$

claim: $k \geq 1$ & $0 \leq x < 2^k \Rightarrow \langle BR(x,k) \rangle = x$

proof: ind. on k.

<u>basis</u>: $k = 1$: $0 \leq x < 2$.

$BR(x,k) = x \in \{0,1\}$

indeed, $\langle x \rangle = x$

<u>hyp</u>: $\langle BR(x,k) \rangle = x$

<u>step</u>: prove that if $0 \leq x < 2^{k+1}$

then $\langle BR(x, k+1) \rangle = x$

$$\langle BR(x, k+1)\rangle = x$$

case 1 : $x < 2^k$

$$BR(x, k+1) = 0 \circ BR(x, k)$$

ind. hyp. $\langle BR(x, k)\rangle = x$

So

$$\langle BR(x, k+1)\rangle = \langle 0 \circ BR(x, k)\rangle$$

$$= \langle BR(x, k)\rangle$$

$$= x$$

$$\langle BR(x, k+1) \rangle = x$$

case 2: $2^{k+1} > x \geq 2^k$

$$BR(x, k+1) = 1 \circ BR(x - 2^k, k)$$

ind. hyp. $\langle BR(x-2^k, k) \rangle \overset{?}{=} x - 2^k$

$x - 2^k < 2^k$

so

$$\langle BR(x,k+1) \rangle = \langle 1 \circ BR(x-2^k, k) \rangle$$

$$= 2^k + \langle BR(x-2^k, k) \rangle$$

$$= 2^k + x - 2^k = x$$

$\boxtimes$

alternative view of $BR(x, k)$

Notation: $B_k \triangleq \{0, 1, \ldots, 2^k - 1\} \leftarrow$ set of numbers repr. using k-bits

claim: let $x \in B_k$ and $A[k-1:0]$ s.t. $\langle \vec{A} \rangle = x$.

then $A[k-1] = 1 \iff x \geq 2^{k-1}$

proof: $x = \langle \vec{A} \rangle = A[k-1] \cdot 2^{k-1} + \langle A[k-2:0] \rangle$

if $A[k-1] = 0$, then $x = \langle A[k-2:0] \rangle \leq 2^{k-1} - 1$

if $A[k-1] = 1$, then $x \geq 2^{k-1}$

$\boxtimes$

suppose $x \in B_k$ and we want to compute $A[k-1:0]$ s.t. $\langle \vec{A} \rangle = x$.

* if $x < 2^{k-1}$, then $A[k-1] = 0$.

  But: $x \in B_{k-1}$, so

$$A[k-2:0] \leftarrow BR(x, k-1)$$

* if $x \geq 2^{k-1}$, then $A[k-1] = 1$.

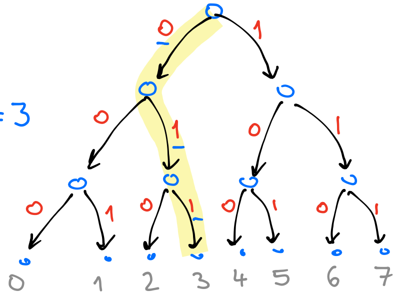
  now $x \leq 2^k - 1 \Rightarrow x - 2^{k-1} \leq 2^{k-1} - 1 \Rightarrow x - 2^{k-1} \in B_{k-1}$

  hence

$$A[k-2:0] \leftarrow BR(x - 2^{k-1}, k-1).$$

Trie

$\langle 011 \rangle = 3$



find bin. repr. of x

* find path from root to leaf x
* edge labels along path root $\rightsquigarrow$ leaf represent x.

1) decision: follow edge labelled 1

$\iff \quad x \geq 2^{k-1}$

2) what about finding path from leaf to root? (extract bits LSB $\to$ MSB).
(see book for such an alg.)

How many bits do we need to represent x?

Corollary

Every positive integer x has a binary representation by a k-bit binary string if $k > \log_2(x)$.

Proof.

$BR(x, k)$ succeeds if $x < 2^k$. Take a log:

$$\log_2(x) < k.$$

$\square$

Theorem (unique binary representation)

The binary representation function

$$\langle\rangle_k : \{0,1\}^k \to \{0,\ldots,2^k-1\}$$

defined by

$$\langle A[k-1:0]\rangle_k \triangleq \sum_{i=0}^{k-1} A[i]\cdot 2^i$$

is a bijection (i.e., one-to-one and onto).

ex:
1) $f : A \xrightarrow{\text{onto}} B$
2) $|A| = |B|$
$\Rightarrow f$ in 1-1

Proof.

1. $\langle\rangle_k$ is onto because $\langle BR(x,k)\rangle_k = x$.
2. $|\text{Domain}| = |\text{Range}|$ implies that $\langle\rangle_k$ is one-to-one.

□

We claim that when a natural number is multiplied by two, its binary representation is "shifted left" while a single zero bit is padded from the right. That property is summarized in the following lemma.

Lemma

Let $a \in \mathbb{N}$. Let $A[k-1:0]$ be a k-bit string such that $a = \langle A[k-1:0] \rangle$. Let $B[k:0] \triangleq A[k-1:0] \circ 0$, then

$$2 \cdot a = \langle B[k:0] \rangle.$$

Example

$\langle 1000 \rangle = 2 \cdot \langle 100 \rangle = 2^2 \cdot \langle 10 \rangle = 2^3 \cdot \langle 1 \rangle = 8.$

$\langle 10 \circ 0 \rangle = 4 \qquad \langle 1 0 \rangle = 2$

claim: $B[k:0] = A[k-1:0] \circ 0$

$\Rightarrow \langle \vec{B} \rangle = 2 \cdot \langle \vec{A} \rangle$

proof

$$\langle \vec{B} \rangle = \sum_{i=0}^{k} B[i] \cdot 2^i$$

$$= \sum_{i=1}^{k} A[i-1] \cdot 2^i + 0 \cdot 2^0$$

$$= 2 \cdot \sum_{j=0}^{k-1} A[j] \cdot 2^j = 2 \cdot \langle \vec{A} \rangle$$

$\boxtimes$