Polynomials

A polynomial \( p = d_n x^n + \cdots + d_0 x^0 \) with variable \( x \) and integer coefficients \( d_i \) may be used to represent the integer \( p(b) \); the integer \( b \) is the base and the sequence \( d_n \ldots d_0 \) is comprised of digits.

For example, the digits 1011 determine the polynomial \( p = 1x^3 + 0x^2 + 1x^1 + 1x^0 \) which represents the integer \( 2^3 + 0^2 + 1^1 + 1^0 = 11 \) when the base is 2. This is commonly denoted by 1011 = 11 base 2. An alternative notation is: 1011₂ = 11.

Keeping digits 1011 but changing the base to 16 produces \( 16^3 + 16^1 + 16^0 = 4113 \).
Hence 1011₁₆ = 4113. An alternative notation is: 0x1011 = 4113.

The base \( b \) is typically restricted to an integer greater than 1, and the digits are restricted to elements of \( \{0, \ldots, b - 1\} \). We follow that convention.

Modular arithmetic

Loosely speaking, modular arithmetic refers to arithmetic on integers where one is allowed to pretend all multiples of some positive integer \( m \) are zero; \( m \) is called the modulus.

For example, if the modulus is 3, then

\[
54 = 3 + 51 = 0 + 51 = 51 = 17 \cdot 3 = 0
\]

We would say: 54, 51, and 0 are all equal mod 3. This is typically denoted by \( 54 \equiv 51 \equiv 0 \). To identify a single answer from among infinitely many possibilities, it is customary to choose an element from \( \{0, \ldots, m - 1\} \). Thus we would say 54 is 0 mod 3, since 0 \( \in \{0, 1, 2\} \). This is commonly denoted by \( 54 = 0 \text{ mod } 3 \).

If the modulus is 7, then

\[
54 = 7^2 + 5 \equiv 0 + 5 = 5
\]

Thus \( 54 = 5 \text{ mod } 7 \), since 5 \( \in \{0, \ldots, 6\} \). An alternative notation is: \( 54 \equiv 5 \mod 7 \).

If \( n = qm + r \), where \( q \) is the quotient from dividing \( n \) by \( m \), and \( r \in \{0, \ldots, m - 1\} \) is the remainder, then

\[
n = r \mod m
\]

For example, if the modulus is 3, then \( 54 = 0 \mod 3 \) because 54 divided by 3 has remainder 0.

If the modulus is 7, then \( 54 = 5 \text{ modulo } 7 \), because 54 divided by 7 has remainder 5.
Modular arithmetic is useful for computing the digits of \( p(b) \):

\[
\begin{align*}
p(b) & = d_n b^n + \cdots + d_0 b^0 \\
p(b) \mod b & = d_0 \\
(p(b) - d_0)/b & = d_n b^{n-1} + \cdots + d_1 b^{1-1} \\
((p(b) - d_0)/b - d_1)/b & = d_n b^{n-2} + \cdots + d_2 b^{2-2} \\
& \vdots
\end{align*}
\]

The computation above is expressed by the following flowchart:

Implement the above algorithm in MIPS.

For example, to express 4113 in base 16,

\[
\begin{align*}
p & = 4113 \\
d_0 & = 4113 \mod 16 = 1 \\
p & = (4113 - 1)/16 = 257 \\
d_1 & = 257 \mod 16 = 1 \\
p & = (257 - 1)/16 = 16 \\
d_2 & = 16 \mod 16 = 0 \\
p & = (16 - 0)/16 = 1 \\
d_3 & = 1 \mod 16 = 1 \\
p & = (1 - 1)/16 = 0
\end{align*}
\]

Note that the digits were printed in reverse order! Hence \( 1011_{16} = 4113 \).
Two’s complement

One need not necessarily regard the answer to \( n \mod m \) as an element of \( \{0, \ldots, m - 1\} \). For example, if \( m \) is even and if we are free to pretend \( m \) is zero, then we could:

\[
\begin{align*}
\text{interpret } & 0 & \text{as } & 0 \\
\vdots & & \vdots & \vdots \\
\text{interpret } & m/2 - 1 & \text{as } & m/2 - 1 \\
\text{interpret } & m/2 & \text{as } & m/2 - m = -m/2 \\
\vdots & & \vdots & \vdots \\
\text{interpret } & m - 1 & \text{as } & m - 1 - m = -1
\end{align*}
\]

Suppose \( m = 2^n \) and elements \( x \in \{0, \ldots, 2^n - 1\} \) are represented in base 2; then \( n \) bits suffice,

\[
x = (b_{n-1} \cdots b_0)_2 = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_02^0 \tag{1}
\]

**Unsigned arithmetic** refers to interpreting \( x \) using equation (1). The **binary complement** of \( b_i \) is \( b'_i = 1 - b_i \). The **unsigned complement** \( x' \in \{0, \ldots, 2^n - 1\} \) of \( x \) is

\[
x' = (b'_{n-1} \cdots b'_0)_2 \in \{0, \ldots, 2^n - 1\}
\]

Note that

\[
x + x' = \sum_{0 \leq i < n} b_i 2^i + \sum_{0 \leq i < n} b'_i 2^i = \sum_{0 \leq i < n} (b_i + b'_i) 2^i = \sum_{0 \leq i < n} 2^i = 2^n - 1 \tag{2}
\]

Solving equation (2) for \(-x\) yields

\[
x = -2^n + 1 + x' \equiv 1 + x' \tag{3}
\]

The operation \( x \mapsto 1 + x' \) is called **two’s complement**; according to (3), it maps \( x \) to its negative (modulo \( 2^n \)). Solving equation (2) for \(-1 - x'\) yields

\[
-1 - x' = x - 2^n = -2^n + b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_02^0
\]

\[
= (b_{n-1} - 2)2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_02^0 \tag{4}
\]

Bit \( b_{n-1} \) of \( x \) may be regarded as a **sign bit** indicating whether \( x \) is to be interpreted as negative (if \( b_{n-1} = 1 \) use equation (4)) or non-negative (if \( b_{n-1} = 0 \) use equation (1)). In this scheme, commonly called **signed (two’s complement) arithmetic**, (1) and (4) merge into the single equation

\[
x = (b_{n-1} \cdots b_0)_2 = -b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_02^0 \tag{5}
\]

**Sign extension**

Let \( x \) be the signed \( n \)-bit integer in equation (5), and let \( m > n \). Consider the signed \( m \)-bit integer

\[
x^* = (b_{m-1} \cdots b_{n-1} \cdots b_0)_2 \text{ where } b_i = b_{n-1} \text{ for } n \leq i < m
\]

**Case 1.** \( b_{n-1} = b_n = \cdots = b_{m-1} = 0 \). Applying (5) — with \( n \) replaced by \( m \) — leads to

\[
x^* = 0 + \cdots -b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_02^0 = x
\]
Case 2. $b_{n-1} = b_n = \cdots = b_{m-1} = 1$. Applying (5) — with $n$ replaced by $m$ — leads to

$$x^* = -2^{m-1} + \left( \sum_{i=n-1}^{m-2} 2^i \right) + b_{n-2}2^{n-2} + \cdots + b_0 2^0$$

$$= -2^{m-1} + \left( 2^{n-1}(2^{m-n} - 1) \right) + 2^{n-1} + b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \cdots + b_0 2^0$$

$$= -2^{m-1} + 2^{m-1} - 2^{n-1} + x$$

$$= x$$

Therefore, replicating the high order bit when increasing the number of bits — typically called *sign extension* — will preserve the value of signed integers. The corresponding operation which preserves the value of unsigned integers is *zero padding* (the additional bits are $b_n = \cdots = b_{m-1} = 0$).

**Shifting**

Understanding the result of a left shift — assuming $n$ bits — depends upon whether arithmetic is signed or unsigned. In either case, the effect on bits is

$$b_{n-1} \cdots b_0 \mapsto b_{n-2} \cdots b_0 0$$

Note that 0 fills the vacated bit position on the left, and $b_{n-1}$ is lost. In the unsigned case,

$$x = \sum_{0 \leq i < n} 2^i b_i \mapsto \sum_{0 < i < n} 2^i b_{i-1} = 2x - 2^n b_{n-1} \equiv 2x$$

In the signed case,

$$x = -b_{n-1}2^{n-1} + \sum_{0 \leq i < n-1} 2^i b_i \mapsto -b_{n-2}2^{n-1} + \sum_{0 < i < n-1} 2^i b_{i-1}$$

$$= 2x + 2^n b_{n-1} - 2b_{n-2}2^{n-1}$$

$$\equiv 2x$$

Unlike the unsigned case — where all numbers are non-negative — the negativity of the result depends upon $b_{n-2}$. Consequently, the shift will produce a change in sign whenever $b_{n-1} \neq b_{n-2}$.

For example, assume $n = 4$ bits (arithmetic is mod $2^4 = 16$) and *signed* arithmetic.

$$x = b_3 b_2 b_1 b_0 = 1011_2 = -5 \mapsto b_2 b_1 b_0 0 = 0110_2 = 6$$

The result changed sign because $b_3 = 1 \neq b_2 = 0$. It is nevertheless true that $2x = -10 \equiv 6$.

A right shift comes in two flavors: either *logical* or *arithmetic*. The effect on bits of a logical right shift is

$$b_{n-1} \cdots b_0 \mapsto 0b_{n-1} \cdots b_1$$

Note that 0 fills the vacated bit position on the left, and $b_0$ is lost. In particular, the result is always
non-negative for both signed and unsigned arithmetic. In the unsigned case,

\[ x = \sum_{0 \leq i < n} 2^i b_i \quad \mapsto \quad \sum_{0 \leq i < n-1} 2^i b_{i+1} = \frac{x - b_0}{2} \]

In the signed case,

\[ x = -b_{n-1} 2^{n-1} + \sum_{0 \leq i < n-1} 2^i b_i \quad \mapsto \quad \sum_{0 \leq i < n-1} 2^i b_{i+1} \]

\[ = \frac{x + 2b_{n-1} 2^{n-1} - b_0}{2} \]

\[ = b_{n-1} 2^{n-1} + \frac{x - b_0}{2} \]

The effect on bits of an arithmetic right shift is

\[ b_{n-1} \cdots b_0 \mapsto b_{n-1} b_{n-1} \cdots b_1 \]

Note that \( b_{n-1} \) fills the vacated bit position on the left, and \( b_0 \) is lost. In the unsigned case,

\[ x = \sum_{0 \leq i < n} 2^i b_i \quad \mapsto \quad b_{n-1} 2^{n-1} + \sum_{0 \leq i < n-1} 2^i b_{i+1} = b_{n-1} 2^{n-1} + \frac{x - b_0}{2} \]

In the signed case,

\[ x = -b_{n-1} 2^{n-1} + \sum_{0 \leq i < n-1} 2^i b_i \quad \mapsto \quad -b_{n-1} 2^{n-1} + \sum_{0 \leq i < n-1} 2^i b_{i+1} \]

\[ = -b_{n-1} 2^{n-1} + \frac{x + 2b_{n-1} 2^{n-1} - b_0}{2} \]

\[ = \frac{x - b_0}{2} \]

Unlike the left shift, an arithmetic right shift cannot produce a change in sign. Note that the result — for both unsigned and signed arithmetic — is equivalent to integer division with rounding toward \(-\infty\).

**Overflow**

*Overflow* refers to the situation where a result requires more than the limited number of bits available to represent it. Operations that use modular arithmetic — where the modulus is \( m = 2^n \) and \( n \) is the number of bits available for operands and results — cannot overflow!

However, using modular arithmetic as a surrogate for real arithmetic can yield an incorrect result; that situation is also commonly referred to as “overflow”. *Signed Overflow* refers to any instance of this type of “overflow” where, in particular, the sign of the result is incorrect. Otherwise (when the sign of the incorrect result happens to be correct), the “overflow” is called *Unsigned Overflow*.

**LAB 1**

1. Implement the program whose flowchart is given on Page 2 of this handout.
3. Do exercises 4.1, 4.2