IV.4. Fixed-point Representations

Numerical Representations: Problems

- sign representation
  - no special symbol available, only digit symbols
- decimal point
  - must know its position at each moment
- arithmetic operations
  - implementation - as efficient as possible
  - not possible for all operations at the same time
  - we must decide which operations to optimize

Fixed-point Encodings

- sign - use one of the bits
- decimal point
  - always the same position in the bit string
    - no need to explicitly memorize the position
- operations with efficient implementation
  - addition, subtraction
- encodings - on **n**+**m** bits (**n**≥1, **m**≥0)
  - **m**=0 - integer numbers
  - **n**=1 - subunit numbers

Redundant Encodings

- redundant encoding
  - there is at least one number with two distinct representations
  - problems with arithmetic operations
- encodings used in practice
  - positive number representation - same as for unsigned numbers; different only for negative numbers
  - some have two distinct representations for 0

Sign-magnitude Representation

- notation: A+S

$$val_{A+S}^{n,m}(a_{n-1}a_{n-2}\ldots a_1 a_0 a_{-1}\ldots a_{-m}) =$$

$$= \begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{if } a_{n-1}=0 \\ -(a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m}) & \text{if } a_{n-1}=1 \end{cases}$$

- similar to base 2 writing
  - the leftmost bit encodes the sign
  - decimal point - implicit

Sign-magnitude - Limits

- on n+m bits - 2^{n+m} distinct representations
  - but only $2^{n+m}-1$ distinct numbers
  - redundant: $val_{A+S}^{n,m}(00\ldots0)=val_{A+S}^{n,m}(10\ldots0)=0$
- extreme values that can be represented

  $max_{A+S}^{n,m}=val_{A+S}^{n,m}(01\ldots1)=2^{n-1}-2^{-m}$

  $min_{A+S}^{n,m}=val_{A+S}^{n,m}(11\ldots1)=-(2^{n-1}-2^{-m})$
  - so one can represent numbers within the interval $[-(2^{n-1}-2^{-m}); +(2^{n-1}-2^{-m})]$

Sign-magnitude - Precision

- numbers that can be represented exactly start from min=$-(2^{n-1}-2^{-m})$
  - and continue with step 2^{-m}
- the other numbers within the interval
  - approximation
  - error - at most 2^{-m}
  - so precision is 2^{-m}
- for fixed n+m
  - bigger numbers = poorer precision

Examples (1)

$$val_{A+S}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$$

$$val_{A+S}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12.75$$

or

$$val_{A+S}^{6,2}(00110011)=val_{A+S}^{8,0}(00110011):2^2=51:4=12.75$$

$$val_{A+S}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3.1875$$

or

$$val_{A+S}^{4,4}(00110011)=val_{A+S}^{8,0}(00110011):2^4=51:16=3.1875$$

Examples (2)

$$val_{A+S}^{8,0}(10110011)=-\left(2^5+2^4+2^1+2^0\right)=-51$$

$$val_{A+S}^{4,4}(10110011)=-(2^1+2^0+2^{-3}+2^{-4})=-3.1875$$

or

$$val_{A+S}^{4,4}(10110011)=val_{A+S}^{8,0}(10110011):2^4=-51:16=-3.1875$$

$$min_{A+S}^{8,0}=val_{A+S}^{8,0}(11111111)=-127$$

$$min_{A+S}^{4,4}=val_{A+S}^{4,4}(11111111)=-7.9375$$

or

$$min_{A+S}^{4,4}=min_{A+S}^{8,0}:2^4=-127:16=-7.9375$$

Examples (3)

$$max^{8,0}_{A+S}=val^{8,0}_{A+S}(01111111)=127$$

$$max^{4,4}_{A+S}=val^{4,4}_{A+S}(01111111)=7.9375$$

or

$$max^{4,4}_{A+S}=max^{8,0}_{A+S}:2^4=127:16=7.9375$$

- intervals for representation
  - $A+S^{8,0}$: [-127; 127] $\rightarrow$ 255 numbers, step 1
  - $A+S^{4,4}$: [-7.9375; 7.9375] $\rightarrow$ 255 numbers, step 0.0625 (=1:16)

Operations in A+S

- addition/subtraction
  - determine the sign of the result (comparison)
  - apply classic algorithms
- multiplication/division
  - similar to classic algorithms
- more complex than we wish
  - we cannot simply use a "classic" adder for computing the sum

One's Complement Representation

- notation: C_1

$$val_{C_1}^{n,m}(a_{n-1}a_{n-2}\ldots a_1a_0a_{-1}\ldots a_{-m})=$$

$$=\begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{if } a_{n-1}=0 \\ (a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m})-(2^{n-1}-2^{-m}) & \text{if } a_{n-1}=1 \end{cases}$$

- homework: prove that the value is negative for $a_{n-1} = 1$
  - so a_{n-1} stands for the sign

One's Complement - Limits

- on n+m bits - 2^{n+m} distinct representations
  - but only $2^{n+m}-1$ distinct numbers
  - redundant: $val_{C_1}^{n,m}(00...0)=val_{C_1}^{n,m}(11...1)=0$
- extreme values that can be represented

$max_{C_1}^{n,m}=val_{C_1}^{n,m}(01...1)=2^{n-1}-2^{-m}$

$min_{C_1}^{n,m}=val_{C_1}^{n,m}(10...0)=-(2^{n-1}-2^{-m})$

  - so one can represent numbers within the interval $[-(2^{n-1}-2^{-m}); +(2^{n-1}-2^{-m})]$

One's Complement - Precision

- numbers that can be represented exactly start from min=$-(2^{n-1}-2^{-m})$
  – and continue with step 2^{-m}
- the other numbers within the interval
  – approximation
  – error - at most 2^{-m}
  – so precision is 2^{-m}
- for fixed n+m
  – bigger numbers = poorer precision

Complementing

- representations of positive numbers - easy to determine
- harder for negative numbers
- is there a relation between the representations of numbers q and $-q$?
- yes: representation of $-q$ is achieved by negating all bits in the representation of q
  - commutative operation - also holds for $q < 0$

Examples (1)

$$\mathrm{val}_{C_1}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$$

$$\mathrm{val}_{C_1}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12.75$$

or

$$\mathrm{val}_{C_1}^{6,2}(00110011)=\mathrm{val}_{C_1}^{8,0}(00110011):2^2=51:4=12.75$$

$$\mathrm{val}_{C_1}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3.1875$$

or

$$\mathrm{val}_{C_1}^{4,4}(00110011)=\mathrm{val}_{C_1}^{8,0}(00110011):2^4=51:16=3.1875$$

Examples (2)

$$val_{C_1}^{8,0}(11001100)=(2^6+2^3+2^2)-(2^7-2^0)=-51$$

$$val_{C_1}^{4,4}(11001100)=(2^2+2^{-1}+2^{-2})-(2^3-2^{-4})=-3.1875$$

or

$$val_{C_1}^{4,4}(11001100)=val_{C_1}^{8,0}(11001100):2^4=-51:16=-3.1875$$

$$min_{C_1}^{8,0}=val_{C_1}^{8,0}(10000000)=0-(2^7-2^0)=-127$$

$$min_{C_1}^{4,4}=val_{C_1}^{4,4}(10000000)=0-(2^3-2^{-4})=-7.9375$$

or

$$min_{C_1}^{4,4}=min_{C_1}^{8,0}:2^4=-127:16=-7.9375$$

Examples (3)

$$\max_{C_1}^{8,0} = \mathrm{val}_{C_1}^{8,0}(01111111) = 127$$
$$\max_{C_1}^{4,4} = \mathrm{val}_{C_1}^{4,4}(01111111) = 7.9375$$

or

$$\max_{C_1}^{4,4} = \max_{C_1}^{8,0} : 2^4 = 127:16 = 7.9375$$

- intervals for representation
  - $C_1^{8,0}$: [-127; 127] $\rightarrow$ 255 numbers, step 1
  - $C_1^{4,4}$: [-7.9375; 7.9375] $\rightarrow$ 255 numbers, step 0.0625 (=1:16)

Operations in C_1

- can we add two numbers in C_1 with a "classic" adder?
- yes, but in two steps
  – in the second step, add the carry out to the result (from the first step)
  – so two adders are needed for addition
- subtraction: add the first operand to the symmetric of the second operand

Two's Complement Representation

- requirements
  - non-redundant representation
    - a single representation for 0
  - the sum of two numbers can be computed with a single adder
    - just as for unsigned numbers
    - gain - a single addition operation implemented in the processor for both signed and unsigned data types

Two's Complement

- notation: C_2

$$val_{C_2}^{n,m}(a_{n-1}a_{n-2}\ldots a_1 a_0 a_{-1}\ldots a_{-m})=$$

$$=\begin{cases} a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m} & \text{if } a_{n-1}=0 \\ (a_{n-2}\times 2^{n-2}+\ldots+a_{-m}\times 2^{-m})-2^{n-1} & \text{if } a_{n-1}=1 \end{cases}$$

- homework: prove that the value is negative for $a_{n-1} = 1$
  - so a_{n-1} stands for the sign

Two's Complement - Limits

- on n+m bits - 2^{n+m} distinct representations
  - and 2^{n+m} distinct numbers
  - 00...0 - the only representation for 0
- extreme values that can be represented

$$\max_{C_2}^{n,m} = val_{C_2}^{n,m}(01...1) = 2^{n-1} - 2^{-m}$$

$$\min_{C_2}^{n,m} = val_{C_2}^{n,m}(10...0) = -2^{n-1}$$

  - so one can represent numbers within the interval $[-2^{n-1}; +(2^{n-1}-2^{-m})]$ - asymmetrical

Two's Complement - Precision

- numbers that can be represented exactly start from min=-2^{n-1}
  - and continue with step 2^{-m}
- the other numbers within the interval
  - approximation
  - error - at most 2^{-m}
  - so precision is 2^{-m}
- for fixed n+m
  - bigger numbers = poorer precision

Complementing (1)

- is there a relation between the representations of numbers q and $-q$?
- yes: representation of $-q$ is the two's complement of the representation of q
  - negate all bits and add 0...01
  - just as for C_1, the operation is commutative - can be applied regardless of the sign of q

Complementing (2)

- example

  q = 77 is represented 01001101 in $C_2^{8,0}$

  -q = -77 is represented 10110010 + 00000001 = 10110011

- homework
  - the C_2 N-bit representation of the negative integer q is in fact the N-bit representation of the number $q + 2^N = 2^N - |q|$

Examples (1)

$$val_{C_2}^{8,0}(00110011)=2^5+2^4+2^1+2^0=51$$

$$val_{C_2}^{6,2}(00110011)=2^3+2^2+2^{-1}+2^{-2}=12.75$$

or

$$val_{C_2}^{6,2}(00110011)=val_{C_2}^{8,0}(00110011):2^2=51:4=12.75$$

$$val_{C_2}^{4,4}(00110011)=2^1+2^0+2^{-3}+2^{-4}=3.1875$$

or

$$val_{C_2}^{4,4}(00110011)=val_{C_2}^{8,0}(00110011):2^4=51:16=3.1875$$

Examples (2)

$val_{C_2}^{8,0}(11001101)=(2^6+2^3+2^2)-(2^7-2^0)=-51$

$val_{C_2}^{4,4}(11001101)=(2^2+2^{-1}+2^{-2})-(2^3-2^{-4})=-3.1875$

or

$val_{C_2}^{4,4}(11001101)=val_{C_2}^{8,0}(11001101):2^4=-51:16=-3.1875$

$min_{C_2}^{8,0}=val_{C_2}^{8,0}(10000000)=0-2^7=-128$

$min_{C_2}^{4,4}=val_{C_2}^{4,4}(10000000)=0-2^3=8$

or

$min_{C_2}^{4,4}=min_{C_2}^{8,0}:2^4=-128:16=-8$

Examples (3)

$\max_{C_2}^{8,0} = \mathrm{val}_{C_2}^{8,0}(01111111) = 127$

$\max_{C_2}^{4,4} = \mathrm{val}_{C_2}^{4,4}(01111111) = 7.9375$

or

$\max_{C_2}^{4,4} = \max_{C_2}^{8,0} : 2^4 = 127:16 = 7.9375$

- intervals for representation
  - $C_2^{8,0}$: [-128; 127] $\rightarrow$ 256 numbers, step 1
  - $C_2^{4,4}$: [-8; 7.9375] $\rightarrow$ 256 numbers, step 0.0625 (=1:16)

Conclusions

- C_2 is the most widely used representation
  - non-redundant
  - addition/subtraction - same implementation as for unsigned numbers
- in practice - integer data types from the programming languages
  - special case (m=0)
  - for real (actually rational) numbers, floating-point representations are used

IV.5. Overflows in Operations on Fixed-point Representations

Overflows

- not enough bits, on the integer part, for the number to be represented
  - the number is outside the representable interval
- problem
  - given two numbers within the representable interval, the result of an operation performed on them may fall outside the interval - *overflow*
  - when does this occur and how do we detect it?

Moving to Longer Representations

- given the *n*-bit representation of a number, what is its $n+k$-bit representation?
  - append non-significant digits to the integer part; the fractional part is left unchanged
- A+S: append *k* digits of value 0 to the right of the sign digit
- C_1, C_2: repeat *k* times the sign digit to its right

Examples

	number	
encoding	51	-51
$A+S^{8,0}$	00110011	10110011
$A+S^{16,0}$	0000000000110011	1000000000110011
$C_1^{8,0}$	00110011	11001100
$C_1^{16,0}$	0000000000110011	1111111111001100
$C_2^{8,0}$	00110011	11001101
$C_2^{16,0}$	0000000000110011	1111111111001101

Moving to Shorter Representations

- we use these results to answer the inverse question
- given the *n*-bit representation of a number, can in be represented on *n*-*k* bits?
  - yes, if and only if the first *k* bits to the right of the sign bit have the values as shown before
  - in that case, those *k* bits can be eliminated from the representation

Operations in C_2

- in the sequel we will only discuss C_2
  - the most widely used representation
- restrictions introduced by the computer on representation-based operations
  - addition: the operands and the result are represented on the same number of bits
  - multiplication: the operands are represented on the same number of bits, and the result is represented on double that number of bits

Overflow - Definition

- let there be a given representation and an operation *op* on numbers
- on n+m bits, numbers can be represented within an interval [min; max]
- let there be two numbers a, b $\in$ [min; max]
- operation *op* performed on a and b causes overflow if

$$a \; op \; b \notin [min; max]$$

Examples (1)

- in the sequel we will use the C_2 representation with n=4, m=0

$1111 + 1111 = 11110 \rightarrow 1110$

  - the "additional" bit is ignored (only 4 bits)
  - that is in fact the carry out bit

$\mathrm{val}_{C_2}^{4,0}(1111)=-1$

$\mathrm{val}_{C_2}^{4,0}(1110)=-2$

  - the result is correct - no overflow

Examples (2)

$0111 + 0111 = 1110 \rightarrow 1110$

– no "additional" bit (carry out is 0)

$val_{C_2}^{4,0}(0111)=7$

$val_{C_2}^{4,0}(1110)=-2$

– incorrect result - overflow occurs

- conclusion - the carry out bit does not provide information about overflow
  – we must find another detection method

Overflow Condition

- we cannot use directly the definition
  - the numbers (operands) are not available
- observation
  - overflow may occur on addition only when both operands have the same sign
    - and representation of the result has the opposite sign
- homework: overflow may not occur when adding two numbers of opposite signs

Algebraic Sum in C_2 (1)

- Theorem 1

  if numbers a and b can be represented in $C_2^{n,m}$, then $a \pm b$ can be represented in $C_2^{n+1,m}$

- Lemma

if $a=val_{C_2}^{n+1,m}(\alpha_n\alpha_{n-1}\ldots\alpha_1\alpha_0\alpha_{-1}\ldots\alpha_{-m})$ and $\alpha_n=\alpha_{n-1}$

then $a=val_{C_2}^{n,m}(\alpha_{n-1}\ldots\alpha_1\alpha_0\alpha_{-1}\ldots\alpha_{-m})$

Algebraic Sum in C_2 (2)

- consider the representations

$$\alpha = \alpha_{n-1}\alpha_{n-2}\ldots\alpha_1\alpha_0\alpha_{-1}\ldots\alpha_{-m}$$

$$\beta = \beta_{n-1}\beta_{n-2}\ldots\beta_1\beta_0\beta_{-1}\ldots\beta_{-m}$$

- we define their formal sum $\gamma = \alpha + \beta$ as

$$\gamma = \gamma_n\gamma_{n-1}\gamma_{n-2}\ldots\gamma_1\gamma_0\gamma_{-1}\ldots\gamma_{-m}$$

that is

$$\sum_{i=-m}^{n}(\gamma_i \times 2^i) = \sum_{i=-m}^{n-1}\left((\alpha_i + \beta_i) \times 2^i\right)$$

Algebraic Sum in C_2 (3)

- Theorem 2

  if the algebraic sum of numbers represented by α and β does not cause overflow, then the representation of the result is

  $$\gamma_{n-1}\gamma_{n-2}\cdots\gamma_1\gamma_0\gamma_{-1}\cdots\gamma_{-m}$$

- Theorem 3

  the algebraic sum of numbers represented by α and β does not cause overflow if carry digits C_{n-1} and C_n of the result are equal

Consequences

- addition in C_2 can be performed by using a "classic" adder
  – the sigs bits are added just as any other bits
- overflow in C_2 can be tested by attaching an NXOR gate to the adder
  – whose inputs are the carry digits C_{n-1} and C_n
  – it is thus not necessary to know the operands

IV.4. Floating-point Representations

Problems with Fixed-point Representations

- total length $n+m$ is fixed by hardware
- but, in fixed-point representations, both n and m are also fixed
  - so magnitude and precision are predetermined and cannot be changed
  - what if we need better precision and are willing to decrease magnitude for that?
    - or the opposite

Scientific Notation

- useful for numbers with many digits

  $745194.2623 \times 10^{-32}$ instead of

  $0.000000000000000000000000007451942623$

- same number - many ways of writing it

  $571.42 \times 10^{2} = 5.7142 \times 10^{4} = 571420 \times 10^{-1} = \ldots$

- normalized writing
  - exactly one significant digit to the left of the decimal point - unique for each number

  5.7142×10^{4}

Scientific Notation in Base 2

- the significant digit before the decimal point can only be 1
  - no need to memorize it in practice
- exception - representation of number 0
  - only digits with value 0
- normalized writing (non-zero number)

  $1.xx...x \times 2^y$

  - base 2 is implicit - also no need to memorize it

Floating-point Representations

- components
  - sign (S): 0 or 1 (+ or -)
  - mantissa (M): 1.*xx*...*x*
    - usually, only the fractional part (f) is memorized

    M = 1 + f; f = 0.*xx*...*x*
  - characteristic (scale)
    - excess representation of the exponent

$$N = (-1)^{S} \times 1.f \times 2^{C\text{ - excess}}$$

Limits

- the length of the characteristic is fixed
  - so there are a minimal and a maximal value of the exponent
- overflow - exponent too big
  - the number is considered $\pm\infty$
- underflow - exponent too small
  - the number is considered 0
- the error type does not depend on the sign

Standardization

- essential for portability
- standard IEEE 754/1985
  - established between 1977 and 1985
  - first commercial implementation: Intel 8087
- 2 main variants
  - single precision (32 bits)
  - double precision (64 bits)
  - some extensions have also been designed

Single Precision

31	30 … 23	22 … 0
S	C = exponent + 127	f = fractional part of the mantissa

- corresponds to data type *float* in C/C++
- limits in base 10
  - minimum: $\approx 1.2 \times 10^{-38}$
    - any number with smaller module is considered 0
  - maximum: $\approx 3.4 \times 10^{38}$
    - any number with bigger module is considered $\pm\infty$

Double Precision

63	62 … 52	51 … 0
S	C = exponent + 1023	f = fractional part of the mantissa

- corresponds to data type *double* in C/C++
- limits in base 10
  - minimum: $\approx 1.7 \times 10^{-308}$
  - maximum: $\approx 1.7 \times 10^{308}$
- higher magnitude
- higher precision

Structure

- in fact, the floating-point representation is made of two fixed-point representations
  – sign and mantissa: sign-magnitude
  – characteristic: excess
- why are fields memorized in order (S,C,f)?
  – to compare two representations, fields must be considered in this order

Representations in IEEE 754/1985

	single precision	double precision
Bits sign+mantissa	24	53
Maximal exponent	128	1024
• finite numbers	127	1023
Minimal exponent	-127	-1023
• normalized numbers	-126	-1022
Characteristic: excess	127	1023

Example 1

- consider number -23.25
  - what is its single precision representation?
- sign: 1 (negative)
- writing in base 2: $-23.25_{(10)} = -10111.01_{(2)}$
- normalization: $10111.01 = 1.011101 \times 2^4$
- characteristic: $4 + 127 = 131 = 10000011_{(2)}$
- representation

  $1\ 10000011\ 0111010...0_{(2)} = \mathrm{C1BA0000}_{(16)}$

Example 2

- which number has the single precision representation $42D80000_{(16)}$?

$42D80000_{(16)} = 0\ 10000101\ 10110000...0_{(2)}$

$S = 0 \rightarrow$ positive number

$C = 10000101_{(2)} = 133_{(10)} \Rightarrow e = 133 - 127 = 6$

$M = 1 + 0.1011 = 1.1011$

- number: $+1.1011 \times 2^6 = 1101100_{(2)} = 108_{(10)}$

Extended Arithmetic

- common real-number arithmetic, plus
  - representation for ∞ and elementary computation rules with it
    - x / ∞, $x \times \infty$, $\infty \pm \infty$
  - representations for the result of undefined operations (NaN - Not a Number) and propagation rules
    - NaN *op* x = NaN, $\forall$*op*
- usage - mathematical libraries

Example

- computing function arccos with the formula

$$\arccos(x)=2\cdot\arctan\sqrt{(1-x)/(1+x)}$$

- what is the value of arccos(-1)?

$$x=-1 \Rightarrow (1-x)/(1+x)=2/0=\infty \Rightarrow$$

$$\Rightarrow \arctan\sqrt{(1-x)/(1+x)}=\pi/2$$

- answer: arccos (-1) = π
  - impossible to reach it without extended arithmetic

Types of Floating-point Values

value type	exponent (e)	f	value
normalized	$e_{min} < e < e_{max}$	any value	$(-1)^S \times 1.f \times 2^e$
denormalized	$e = e_{min}$	$f \neq 0$	$(-1)^S \times 0.f \times 2^e$
zero	$e = e_{min}$	$f = 0$	S 0 (= 0)
infinity	$e = e_{max}$	$f = 0$	S ∞ ($\pm\infty$)
NaN	$e = e_{max}$	$f \neq 0$	NaN