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

$val_{C_2}^{4,0}(1111)=-1$

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

  – the result is correct - no overflow

Examples (2)

$0111 + 0111 = 1110 \rightarrow 1110$

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

$\mathrm{val}_{C_2}^{4,0}(0111)=7$

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

– 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

  0.00000000000000000000000007451942623
- 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

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

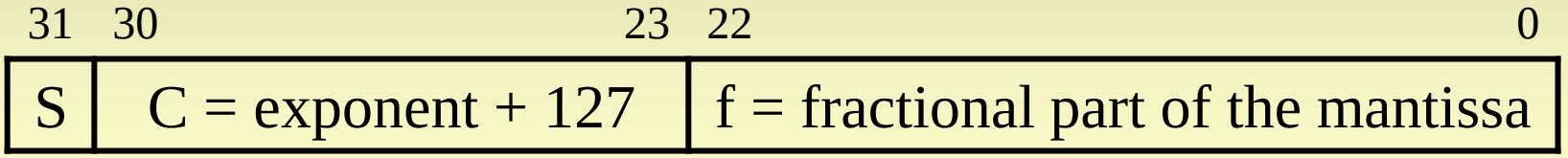
Limits

- 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

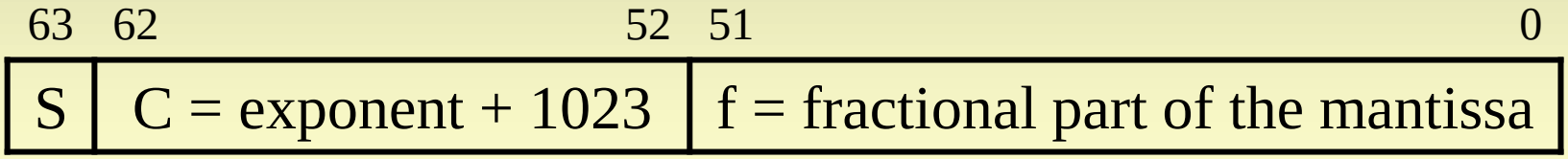
- essential for portability
- standard IEEE 754/1985
  - established between 1977 and 1985
  - first commercial implementation: Intel 8087
- 2 main variants
  - single precision (32 bits)
  - double precision (64 bits)
  - some extensions have also been designed

Single Precision



- 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



- 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

  $1\ 10000011\ 0111010...0_{(2)} = C1BA0000_{(16)}$

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

Exceeding the Limits

- underflow
  - in the normalized form of the number, the negative exponent cannot be represented by the characteristic
    - the number is considered 0
- overflow
  - in the normalized form of the number, the positive exponent is too large to be represented by the characteristic
    - the number is considered $\pm\infty$, depending on the sign

De-normalized Representations

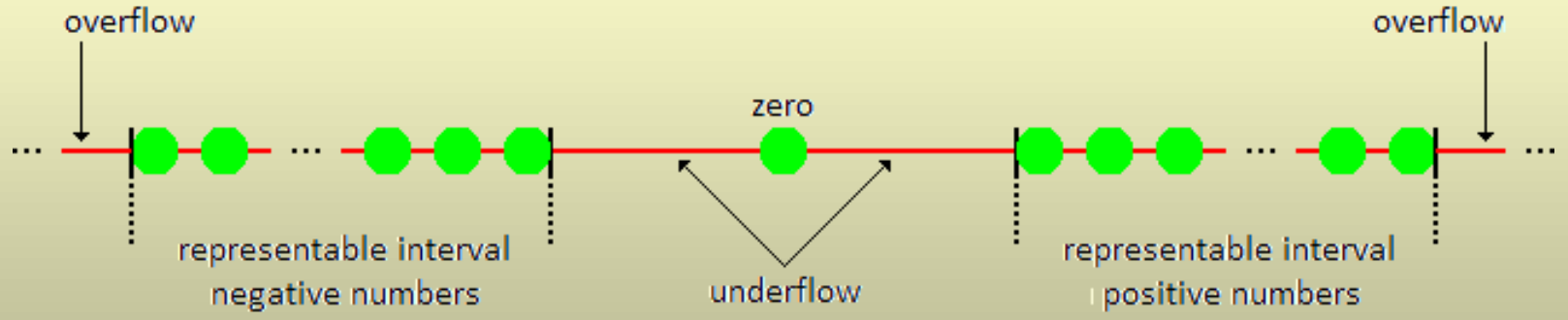
- a number that is smaller in module than the smallest (non-zero) normalized representation
  - the mantissa is no longer normalized
  - so it is going to be 0.f, instead of 1.f
  - while the exponent will have the smallest value
    - single precision: -127
    - double precision: -1023

Approximations (1)

- underflow is actually an approximation
  - a very small non-zero number is considered 0
- what is the precision of floating-point representation?
  - depends on the exponent
  - single precision: 2^{e-23}
  - double precision: 2^{e-52}
- very big exponent - very poor precision

Approximations (2)

- example: e = 123
  - distance between two consecutive numbers that can be represented exactly: $2^{123-23} = 2^{100} \approx 10^{30}$
- what can be represented exactly?
  - rational (not real) numbers - only part of them



Floating-point Arithmetic

- given two numbers

  $x = m_x \cdot 2^{e_x}$

  $y = m_y \cdot 2^{e_y}$

- elementary arithmetic operations

  $x+y = (m_x \cdot 2^{e_x - e_y} + m_y) \cdot 2^{e_y}$, if $e_x \leq e_y$

  $x-y = (m_x \cdot 2^{e_x - e_y} - m_y) \cdot 2^{e_y}$, if $e_x \leq e_y$

  $x \cdot y = (m_x \cdot m_y) \cdot 2^{e_x + e_y}$

  $x : y = (m_x : m_y) \cdot 2^{e_x - e_y}$

Floating-point Addition

- compare the exponents
  - shift one mantissa until they become equal
- add the mantissas
  - in two's complement
- normalize the sum (if necessary)
  - if overflow/underflow occurs - stop
- round the sum's mantissa to the number of bits available

Floating-point Multiplication

- add the exponents
- multiply the mantissas
- normalize the product
  - if overflow/underflow occurs - stop
- round the product's mantissa to the number of bits available
- determine the sign of the result

Homework

- follow the steps of the floating-point addition of the representations of the numbers written in base 10 as -0.75 and 0.375
- the same for the multiplication of the two numbers above
- IEEE 754, single precision representation will be considered

V. Computer Architecture and Organization

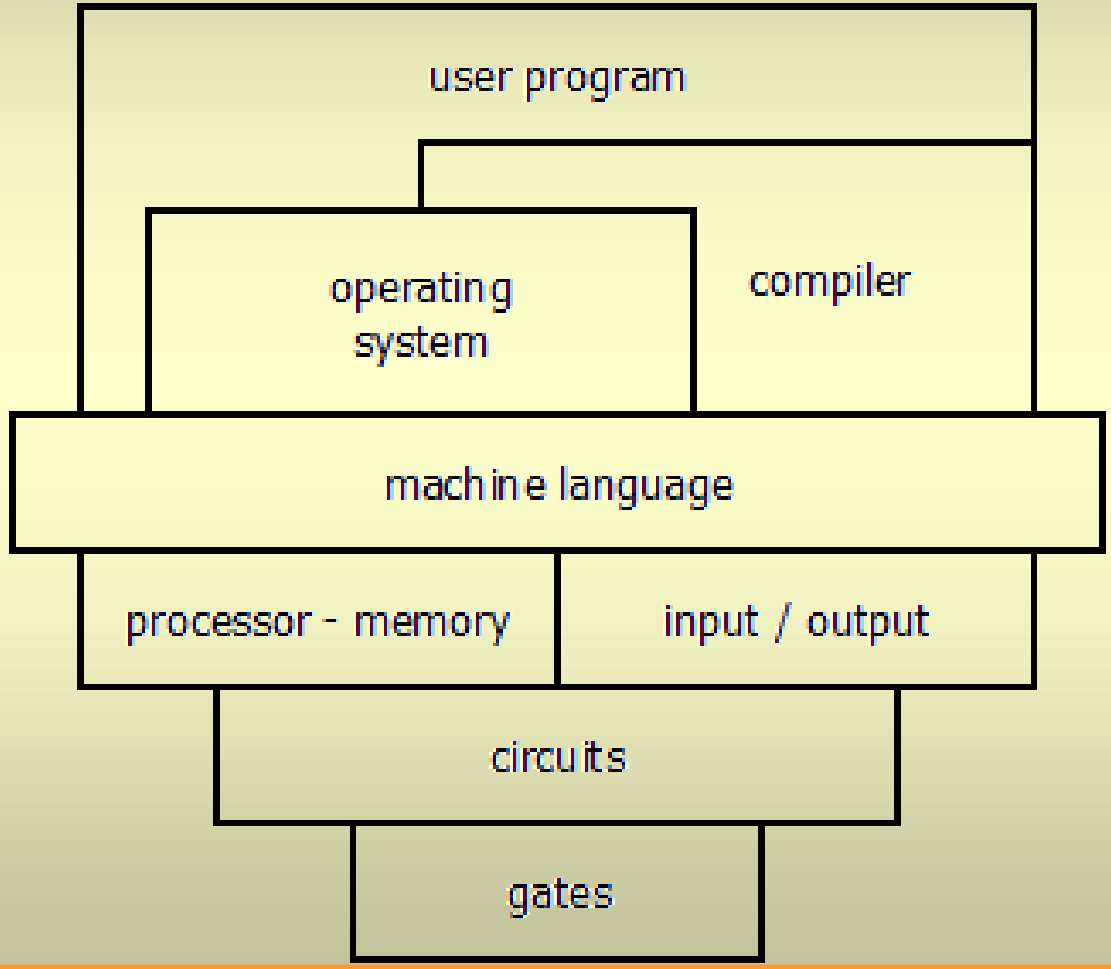
The von Neumann Computer (1)

- memorized program
  - instruction codes are stored in the same memory as the data
- memory
  - infinite (ideally), equal access time to all locations
- upon execution, instruction *i* is followed by
  - the instruction memorized right after *i*
  - the instruction indicated by *i* (branch)

The von Neumann Computer (2)

- the address of the next instruction is permanently maintained by the processor
  - into a dedicated register - PC (Program Counter)
  - and permanently updated, depending on the instruction type ("normal" or branch)
- at each moment, only one instruction is loaded for execution

Computing System - Architecture



Computer Organization