Herbert Wertheim
College of Engineering
UNIVERSITY of FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

Department of Electrical & Computer Engineering

Digital Logic And Computer Systems

Chapter 01 – Numeral Systems

Dr. Christophe Bobda

Agenda

- Polyadic Number Systems
- Negative Numbers
- Operation on Numbers
- Real Numbers
  - Fixed-Point Numbers
  - Floating-Point Numbers

Polyadic Number Systems

Polyadic Number Systems
Introduction

- Transparent Representation of Numbers through
  - Appropriate digit representation and
  - Systematic arrangement

- Polyadic Number System: Representation of a number through digit sequence, to a certain base

$Z_B = (z_{n-1} z_{n-2} \cdots z_1 z_0)_B$ corresponds to the decimal number: $Z_B = z_{n-1} B^{n-1} + z_{n-2} B^{n-2} + \cdots + z_1 B^1 z_0 B^0$

Z_B: Number in B-adic Number System (B)

B: Base, (Radix) $B \geq 2$

B^i: i-th exponent

z_i: i-th digit (coefficient)

Z: Digit set: $z_i \in Z = \{0,1,2, \ldots , B-1\}$

Polyadic Number Systems Introduction

- Commonly Used Number Systems:
  - Binary Numbers: B=2 → 2 Symbols {0, 1}
  - Octal Numbers: B=8 → 8 Symbols {0, 1, ..., 7}
  - Decimal Numbers: B=10 → 10 Symbols {0, 1, ..., 9}
  - Hexadecimal Numbers: B=16 → 16 Symbols {0, 1, ..., 9, A, B, C, D, E, F}

- Theorem: A number in {0, 1, ..., B^n-1} has exactly one n-digit B-adic representation

- Proof:
  - A number has at least one n-digit B-adic representation:
    - per Induction on n
  - A number has at most one representation:
    - Enumeration of digit combination

Polyadic Number Systems Introduction

- Simplified B-adic Representation

  $(b_{n-1}b_{n-2}\ldots b_1b_0)_B$

- When the base is obvious, parenthesis and base can be omitted

  $b_{n-1}b_{n-2}\ldots b_1b_0$

- Example

$$(2186)_{10} = 2 * 10^3 + 1 * 10^2 + 8 * 10^1 + 6 * 10^0 = 2186$$
$$(3014)_5 = 3 * 5^3 + 0 * 5^2 + 1 * 5^1 + 4 * 5^0 = 384$$
$$(1101)_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13$$
$$(719)_{13} = 7 * 13^2 + 1 * 13^1 + 9 * 13^0 = 1205$$

Polyadic Number Systems Introduction

- Non-Polyadic Number Systems: Roman Numerals
  - Additive System,
  - No Digit Representation
    - I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000
  - Example:
    - IV = 4
    - XIII = 13
    - MMII = 2002
    - MCMLXXVII = 1977

Polyadic Number Systems
Optimal Base

- Given a base B and n digit representation
  - Precision $Z=B^n$ = set of all numbers that can be represented:

{00…0, 00…1, 00…B-1, 00…10, …, 00…1B-1, B-1…B-1}

  - Cost: B symbols, per digit → E = B*n
  - Which Base Minimize the Cost E for a given Z

$$Z = B^n \rightarrow n = \frac{\ln(Z)}{\ln(B)}, \quad E = B*n = B*\frac{\ln(Z)}{\ln(B)}, \quad \frac{dE}{dB} = \ln(Z)\frac{\ln(B)-1}{\ln^2(B)}$$

Minimum at B = e = 2,71828182

- Optimal Base at B=3
  - The binary system has been preferred to the tertiary system in the digital world because of it's simple and efficient hardware implementation

Polyadic Number Systems Conversion

- Conversion of a number $X = (x_{n-1}\ldots x_0)_A$ to $X = (x_{m-1}\ldots x_0)_B$
- Horner Scheme

$$Z_B = \sum z_i B^i = (((\cdots((z_{n-1})B + z_{n-2})B + z_{n-3})B + \cdots)B + z_1)B + z_0$$

- Efficient method for polynomic evaluation with n multiplications and n additions
- "Inversion" = successive integer divisions by B, is used for base conversion:

$$N = q.B + r$$

$$r = N \bmod B = N - \left\lfloor \frac{N}{B} \right\rfloor \cdot B$$

$$q = N \operatorname{div} B = \left\lfloor \frac{N}{B} \right\rfloor$$

Polyadic Number Systems Conversion

- Division with remainder
  - conversion from $X = (x_{n-1}...x_0)_A$ to $X = (x_{m-1}...x_0)_B$
  - Method: successive divisions with remainder

    The target base must first be converted in target number system

    X : $B = q_1$ remainder b_0

    $q_1 : B = q_2$ remainder b_1

    $q_2 : B = q_3$ remainder b_2

    . . . . . .

    $q_{N-2} : B = q_{N-1}$ remainder b_{N-2}

    $q_{N-1} : B = 0$ remainder b_{N-1}

Polyadic Number Systems Conversion

- Conversion in source system
  - $X=(x_{n-1}\ldots x_0)_A \rightarrow X=(x_{m-1}\ldots x_0)_B$

    Convert target base B to source system.

    $q_0 = X$

    Repeat until $q_i = 0$

    increment i:

    $q_{i+1} = q_i$ div B;
    $r_i = q_i$ mod B

$r_{m-1} r_{m-2} \ldots r_1 r_0$ *is the* B-adic representation of X *to base* B

Polyadic Number Systems Conversion

LSB = Least Significant Bit

- Example $X=(231)_{10} \rightarrow (x_{m-1}...x_0)_2$

Step 1: $q_0 = 231$

Step 2: $231 = 115 * 2 + \text{remainder } 1 \ (231 \bmod 2) \mid (115 \neq 0)$

Step 3: $115 = 57 * 2 + \text{remainder } 1 \ (115 \bmod 2) \mid (57 \neq 0)$

Step 4: $57 = 28 * 2 + \text{remainder } 1 \ (57 \bmod 2) \mid (57 \neq 0)$

Step 5: $28 = 14 * 2 + \text{remainder } 0 \ (28 \bmod 2) \mid (14 \neq 0)$

Step 6: $14 = 7 * 2 + \text{remainder } 0 \ (14 \bmod 2) \mid (7 \neq 0)$

Step 7: $7 = 3 * 2 + \text{remainder } 1 \ (7 \bmod 2) \mid (3 \neq 0)$

Step 8: $3 = 1 * 2 + \text{remainder } 1 \ (3 \bmod 2) \mid (1 \neq 0)$

Step 9: $1 = 0 * 2 + \text{remainder } 1 \ (1 \bmod 2) \mid (0 = 0)$ *Stop*

$Y = 11100111_2$

$q_{i+1} = q_i \text{ div } B$

MSB = Most Significant Bit

Polyadic Number Systems Conversion

```
   617
12 | 7632
     74
     23
     12
     112
     106
     004
```

- Example:
  - $X = (7632)_8 \xrightarrow{?} X=(x_{m-1}\ldots x_0)_{10}$

$$12_8 \hat{=} 10_{10}$$

Step 1: $q_0 = 7632$

Step 2: $7632 = 617 * 12 + \text{remainder } 4 \ (7632 \bmod 12) \mid (617 \neq 0)$

Step 3: $617 = 47 * 12 + \text{remainder } 11 \ (617 \bmod 12) \mid (47 \neq 0)$

Step 4: $47 = 3 * 12 + \text{remainder } 11 \ (47 \bmod 12) \mid (3 \neq 0)$

Step 5: $3 = 0 * 12 + \text{remainder } 3 \ (3 \bmod 12) \mid (0 = 0)$ Stop

$Y = 3994_{10}$

LSB = Least Significant Bit

MSB = Most Significant Bit

Polyadic Number Systems Conversion

- Conversion in target system:
  - $X=(x_{n-1}\ldots x_0)_A \rightarrow X=(x_{m-1}\ldots x_0)_B$
  - Processing of Horner's scheme from left to right:

    $((..(b_{N-1}B+ b_{N-2})*B+ b_{N-3})*B\ldots+b_1)*B+ b_0$

  - $X = ((..(x_{N-1}A+ x_{N-2})*A+ x_{N-3})*A\ldots+x_1)*A+ x_0$
  - Convert every x_i to B-adic system
  - Convert A to B-adic system
  - Compute result in B-adic System

$Y \leftarrow 0$

for $i=n-1,\ldots,0$:

$Y \leftarrow Y.A_B+(x_i)_B$

$147_{10} \rightarrow XXX_3$

$$147_{10} = 1 \times (101)^2 + (11) \times (101)^1 + (21) \times (101)^0$$

$147_{10} \rightarrow XXX_2$

$$147_{10} = 1 \times (1010)^2 + (100) \times (1010)^1 + (111) \times (1010)^0$$

Polyadic Number Systems Conversion

- Example: $X=(1101011)_2 \rightarrow (x_{m-1}\ldots x_0)_{10}$

$$X = ((((((1)*2+1)*2+0)*2+1)*2+0)*2+1)2+1$$

$$= ((((((1_{10})*2_{10}+1_{10})*2+0_{10})*2_{10}+1_{10})*2_{10}+0_{10})*2_{10}+1_{10})2_{10}+1_{10}$$

Step 1:		$Y = 0$
Step 2:	$i = 6$:	$Y \leftarrow 0*2 + 1 = 1$
Step 3:	$i = 5$:	$Y \leftarrow 1*2 + 1 = 3$
Step 4:	$i = 4$:	$Y \leftarrow 3*2 + 0 = 6$
Step 5:	$i = 3$:	$Y \leftarrow 6*2 + 1 = 13$
Step 6:	$i = 2$:	$Y \leftarrow 13*2 + 0 = 26$
Step 7:	$i = 1$:	$Y \leftarrow 26*2 + 1 = 53$
Step 8:	$i = 0$:	$Y \leftarrow 53*2 + 1 = 107$

$Y = 107_{10}$

Polyadic Number Systems Conversion

- Example: $X=(5246)_7 \rightarrow (x_{m-1}\ldots x_0)_9$

$$X = ((((5)*7+2)*7+4)*7+6$$

$$= (((5_9)*7_9+2_9)*7_9+4_9)*7_9+6_9$$

Step 1: $Y = 0$

Step 2: $i = 3$: $Y \leftarrow 0*7 + 5 = 5$

Step 3: $i = 2$: $Y \leftarrow 5*7 + 2 = (38_9 + 2_9) = 41_9$

Step 4: $i = 1$: $Y \leftarrow 41*7 + 4 = (317_9 + 4_9) = 322_9$

Step 5: $i = 0$: $Y \leftarrow 322*7 + 7 = (2465_9 + 6_9) = 2472_9$

$Y = 2472_9$

Polyadic Number Systems Conversion

- Conversion to base which are exponent of to
  - Convert the input number digit wise to binary number
  - Group corresponding number of bits (LSB-first) for a digit in target system (pad 0s left if needed)
  - Replace the groups with the digits in target system
- Example: from Octal to Hexadecimal
  - $X = 2467_8 \rightarrow ?_H$
  - $X = |010|100|110|111|_2$
    $= |0101|0011|0111|_2 = 537_H$

Negative Numbers

Polyadic Number Systems
Negative Numbers

- Negative numbers can be represented in two different ways
  - Sign and magnitude
  - Complement to a given base B
    - Radix complement (2s complement for binary numbers)
    - (Radix – 1) complement (1s complement in binary system)

- Sign and magnitude
  - $Z_B = z_{n-1} \; z_{n-2} \ldots z_1 \; z_0$

    Sign Magnitude
  - $Z_B = + \sum_{i=0}^{n-1} z_i * B^i$ *(positive) if* $z_{n-1} = 0$
  - $Z_B = - \sum_{i=0}^{n-1} z_i * B^i$ *(negative) if* $z_{n-1} = 1$

Polyadic Number Systems
Sin and Magnitude Representation

- Classic representation of numbers
- Interval: [$-(2^{n-1} - 1)$, $+(2^{n-1} - 1)$]
- Two zeros: +0 (00…0), – 0 (10…0)
  - 0-test must be done twice
- Negation is easy $-X \rightarrow X$
- Complex addition and Subtraction process: involves case differentiation
  - Addition of a positive number X with a negative Number -Y: X+(-Y)
    - If Y>X, Result is -(Y-X)
  - Method
    - Swap operands
    - Perform subtraction (instead of addition)
    - Append sign
- Easy to perform Multiplication and division

Polyadic Number Systems
Excess Representation

- A n-bit-Number *d* is represented as d‘ with d‘ := d + m
- With excess 2^{n-1} the number interval is asymmetric:
  - $[\ (0 - 2^{n-1}), (2^n - 1 - 2^{n-1})\ ] = [\ -2^{n-1}, 2^{n-1} - 1\ ]$
- Application: Exponent representation in Floating point numbers
- Example: *n = 3*

	Excess-4		
x_{10}	x_2	x_1	x_0
-4	0	0	0
-3	0	0	1
-2	0	1	0
-1	0	1	1
0	1	0	0
1	1	0	1
2	1	1	0
3	1	1	1

Polyadic Number Systems
Complement Representation

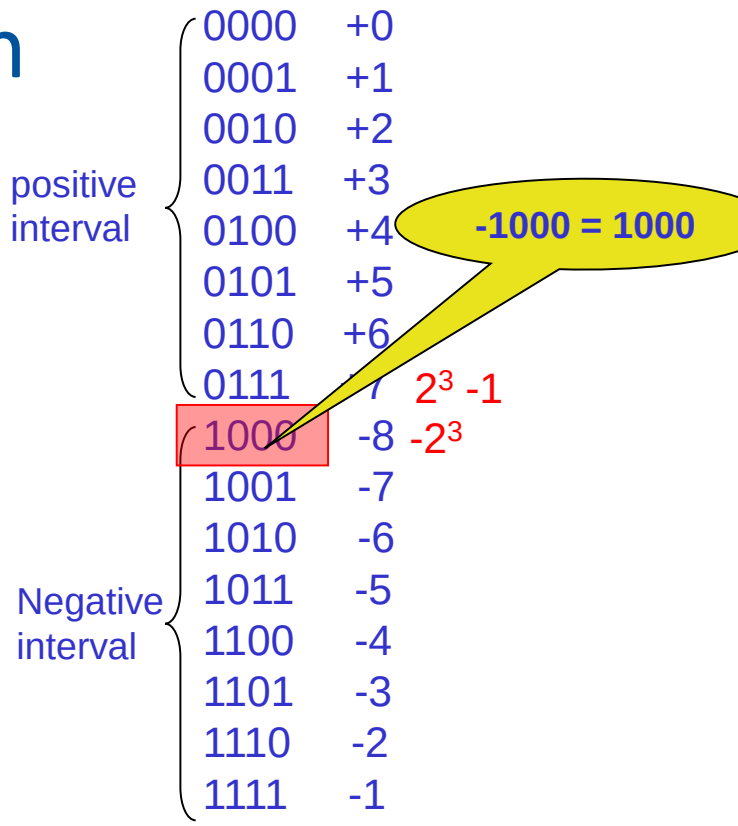
- The B-complement of a *n*-digit B-adic number N is the *n*-digit B-adic number that consists of the last *n* digits of B^n-N.
  - B-complement of n is interpreted as -n
- Two's complement
  - *Two's complement of X_2 is $2^n - X_2$*
  - *Two's complement of 11011_2* is *$2^n - X_2 = 00101$*
- Application: Datapath in modern computers
  - Positive numbers in interval $[0, 2^{n-1} - 1]$ are represented "normally"
  - Negative number X in interval $[-2^{n-1} - 1, 0]$ are represented with their two's complement $2^n - |X|$

Polyadic Number Systems
Complement Representation

- Negating a number (building its B-complement):
  - $X=(x_{n-1}x_{n-2}...x_1x_0)_B$
  - Digit wise conversion: x_i becomes $(B\text{-}1\text{-}x_i) = \overline{x}_i$
  - Add 1 to the resulting number
  - $\sum \overline{x}_i 2^i + 1 = \sum (1 - x_i) 2^i + 1 = \sum 2^i - \sum x_i 2^i + 1$
    $= 2^n - 1 - |X| + 1 = 2^n - |x|$
  - Example: 01101 → 10010+1 = 10011
    10111 → 01000+1 = 01001

Polyadic Number Systems
Complement Representation

- Properties
  - Only one representation of zero
  - Easy sign-test
    - Sign-bit (x_{n-1}= 1 for positive numbers x_{n-1}= 0 for negative numbers)
  - Easy addition und subtraction
    - X-Y=X+(-Y)= X+($2^n - Y$)
  - Number interval asymmetric
    - [-(2^{n-1}), +(2^{n-1}-1)]
    - There is an anomaly for number 2^{n-1} which is equal to its negation



Polyadic Number Systems
Complement Representation

- One's complement
  - Two's complement -1
  - Conversion to 1 complement through digit wise negation
  - Symmetric number interval
  - easy to perform addition, subtraction, multiplication and division
  - Two representations of zero

- Negating a number (computing it's (B-1)-complement):
  - $X=(x_{n-1}x_{n-2}...x_1x_0)_B$
  - Replace each digit x_i with $(B\text{-}1\text{-}x_i) = \overline{x}_i$
  - $\sum \overline{x}_i 2^i = \sum (1 - x_i) 2^i = \sum 2^i - \sum x_i 2^i$
    $= 2^n - 1 - |X| = 2^n - |x| - 1$
  - Example 01101 → 10010
    10111 → 01000

Polyadic Number Systems Side-by-Side Comparison

Sequence	Two's complement	One's complement	Signed-magnitude
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0
1111	−1	−0	−7
1110	−2	−1	−6
1101	−3	−2	−5
1100	−4	−3	−4
1011	−5	−4	−3
1010	−6	−5	−2
1001	−7	−6	−1
1000	−8	−7	−0

Polyadic Number Systems Addition/Subtraction

- Classic method to perform addition Z = X+Y
  - Digit wise addition from right to left $x_i + y_i + c_{i-1}$
    - C_i *is the carry from digit i-1* ($C_{-1} = 0$)
  - Carry c_i is propagated to the next left position
- Subtraction: Z = X-Y = Z + (-Y)
- Example

0	1	1	0	0 (carry)
0	0	1	1	1 + 7
0	0	1	1	0 + 6
0	1	1	0	1 +13

1	1	1	1	0
1	1	1	0	1 - 3
1	0	1	1	1 - 9
1	0	1	0	0 -12

1	0	0	0	0
0	1	0	1	1 +11
0	1	1	0	0 +12
1	0	1	1	1 +23

1	1	1	1	0
1	1	1	0	1 - 3
0	1	1	1	1 +15
0	1	1	0	0 +12

x_i	y_i	c_{i-1}	z_i	c_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Polyadic Number Systems Subtraction

- Classic method to perform addition Z = X-Y
  - Digit wise subtraction from right to left $x_i - (y_i + c_{i-1})$
    - B_i *is the borrow from digit i-1* ($B_{-1} = 0$)
  - Borrow B_i is propagated to the next left position
- Subtraction: Z = X-Y = Z + (-Y)
- Example

X_i	Y_i	B_{i-1}	Z_i	B_i
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

```
0 0 0 0 0 (borrow)
0 1 1 1 1  +15
0 0 1 1 0  + 6
-----------------
0 1 0 0 1  + 9
```

```
0 1 1 0 0
1 1 1 0 1  - 3
1 0 1 1 1  - 9
-----------------
0 0 1 1 0  + 6
```

```
1 1 1 0 0
0 1 0 0 1  +9
0 1 1 1 0  +14
-----------------
1 1 0 1 1  - 5
```

```
0 0 1 0 0
1 1 1 0 1   -3
0 1 0 1 1   11
-----------------
1 0 0 1 0  -14
```

Polyadic Number Systems Subtraction

- Two Complement
  - Build the two complement of the subtrahend
  - Add the result to the minuend
- One's complement
  - Take 1's complement of the subtrahend
  - Add with minuend
    - If the result of above addition has carry bit 1, then add it to the least significant bit (LSB) of given result
    - If there is no carry bit 1, then take 1's complement of the result which will be negative

Polyadic Number Systems Addition/Subtraction

- Overflow happens when the result cannot fit on the number of available digits

	0	1	1	0	0	
	0	1	1	1	1	+15
	0	1	1	1	0	+14
0	1	1	1	0	1	- 3

	0	0	0	0	0	
	1	0	0	1	1	-13
	1	0	1	0	0	-12
1	0	0	1	1	1	+ 7

- Overflow not possible in following cases
  - Addition of two number with opposite signs

    (-10 + 5) = -5

  - Subtraction of two numbers with identical signs

    (-10 - (- 5)) = -10 + 5 = -5)

Polyadic Number Systems Addition/Subtraction

Operation	A	B	Z indicates overflow
A + B	> 0	> 0	< 0
A + B	< 0	< 0	> 0
A - B	> 0	< 0	< 0
A - B	< 0	> 0	> 0

- Overflow test
  - Sign of Operands ≠ Sign of Results

```
  0 1 1 0 0
  0 1 1 1 1    +15
  0 1 1 1 0    +14
-------------------
0 1 1 1 0 1    - 3
```

```
  0 0 0 0 0
  1 0 0 1 1    -13
  1 0 1 0 0    -12
-------------------
1 0 0 1 1 1    + 7
```

Polyadic Number Systems Multiplication

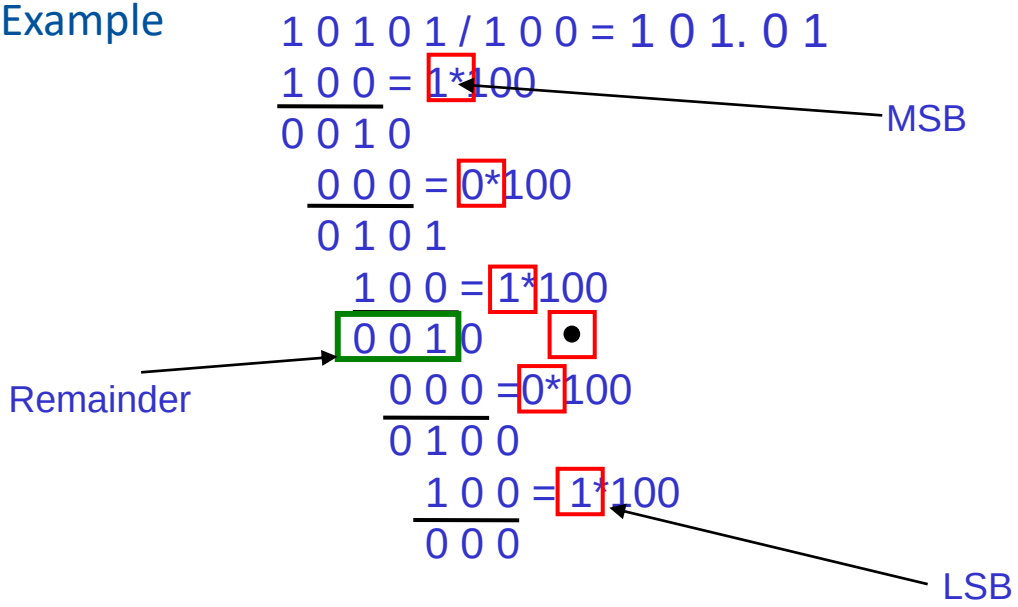
- Like addition/subtraction, multiplication Z = X*Y performed after "textbook" method
  - "Shift and Add"-Principle
  - The product of *n*-digit number with a *m*-digit number is a (*n+m)*-digit number
  - Example

```
  0 1 1 0   +6
  0 1 0 1   +5
  ---------
  0 1 1 0
0 0 0 0
0 1 1 0
0 0 0 0
---------------
0 0 0 1 1 1 1 0  +30
```

  - Efficient multiplication methods will be considered later

Polyadic Number Systems
Division

- Division Z = X/Y also performed using textbook method
  - Example



  - Size of fraction increases with continuous division
  - The final result of an integer division is the pair <Quotient, Remainder>

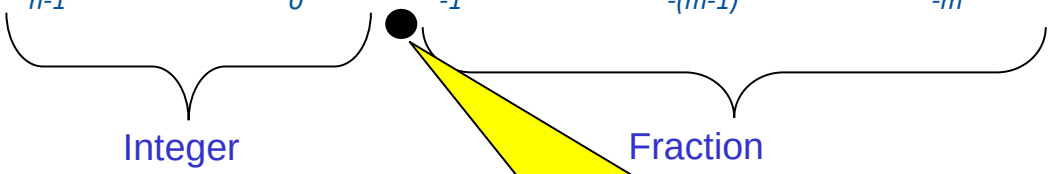
Real Numbers

Polyadic Number Systems
Fix-Point Numbers

- Beside integer numbers, real numbers must also be represented:
  - 3,141592 (π); 2,71828 (e); 0,00000001
- Fixpoint Numbers:
  - N Bits available, logically divided into (integer, fraction)

(n bits integer and m bits fraction)

$$Z_B = z_{n-1} * B^{n-1} + \ldots + z_0 * B^0 + z_{-1} * B^{-1} + \ldots + z_{-(m-1)} * B^{-(m-1)} + z_{-m} * B^{-m}$$



- $Z_B = \sum_{i=0}^{n-1} z_i * B^i + \sum_{i=-1}^{-m} z_i * B^{-i}$

Logic point, no physical representation

Polyadic Number Systems
Fix Point Numbers

- Example
  - $3.1415_D = 3*10^0 + 1*10^{-1} + 4*10^{-2} + 1*10^{-3} + 5*10^{-4}$
  - $11011.1011_D = 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3} + 1*2^{-4} = 27.6875_D$
- Fix points are the general form of polyadic number representation
- Fix point conversion
  - Convert integer and fraction parts separately
    - Integer part conversion using a method previously seen (Ex. successive divisions)
    - For the fraction part, apply successive multiplications (instead of division) to obtain B^{-m}

Polyadic Number Systems
Fix Point Numbers

- Example: convert 13.455_{10} to binary
  - *Integer part:* 13

Step 1: $q_0 = 13$

Step 2: $13 = 6*2 + \text{remainder } 1 \ (13 \bmod 2) \mid (6 \neq 0)$

Step 3: $6 = 3*2 + \text{remainder } 0 \ (6 \bmod 2) \mid (3 \neq 0)$

Step 4: $3 = 1*2 + \text{remainder } 1 \ (3 \bmod 2) \mid (1 \neq 0)$

Step 5: $1 = 0*2 + \text{remainder } 1 \ (1 \bmod 2) \mid (0 = 0)$ Stop

$13_{10} = 1101_2$

  - *Fraction part: .455*

Step 1: $.455*2 \ (.455/2^{-1}) = 0.910 \ (=0 + .910)$

Step 2 $.910*2 \ (.910/2^{-1}) = = 1.820 \ (1 + .820)$

Step 3: $.820*2 \ (.8202^{-1}) = = 1.640 \ (1 + .640)$

Step 4: $.640*2 \ (.640/2^{-1}) = = 1.280 \ (1 + .280)$

Step 5: $.280*2 \ (.280/2^{-1}) = = 0.560 \ (0 + .560)$

$.455_{10} = .01110_2 \rightarrow 13.455_{10} = 1101.01110_2$

Floating-Point Numbers

- Polyadic number systems are not appropriate to represent very large / very small numbers
- Using floating-point, the range and precision can be dynamically adjusted.
- Exponent-Representation: $Z = (-1)^s \cdot M \cdot b^E$

  M: Mantissa, B: Base, S: Sign

  - Base in given architecture is fixed
    - Must not be specified (always a power of 2)
  - Exponent-representation is not unique
    - $37.145 = 0.037145 \cdot 10^3 = 3714.5 \cdot 10^{-2}$

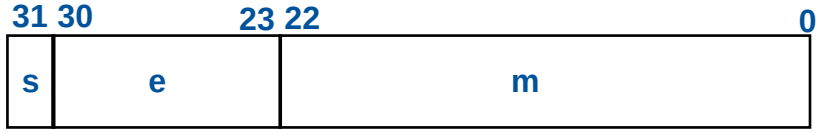
Floating-Point Numbers

- Unique representation is obtained with the Normalization
  - The normalization fixes the position of the comma
- The two possible normalizations are:
  - The binary point is placed before the MSB ≠ 0 of the mantissa
    - 5,25 = 101.01000 · 2 = 0.1010100 · 2^3
    - Mantissa: 1/2 = M < 1
    - Smallest normalized number that can be represented: $0.1 \cdot 2^{-E_{\max}}$ (*Emax* is the maximal exponent)
  - The binary point is placed after the MSB ≠ 0 of the mantissa
    - 5,25 = 101.01000 · 2 = 1.0101000 · 2^2
    - Mantissa: 1 = M < 2
    - Smallest normalized number: $1.0 \cdot 2^{-E_{\max}}$

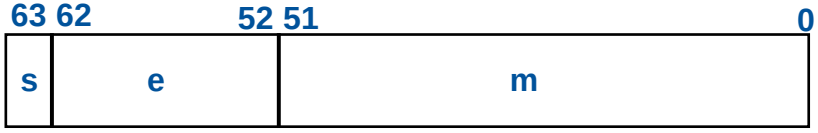
Floating-Point Numbers

IEEE 754 Norm

- Defines:
  - a standard representation of binary floating-point numbers in computers
  - Methods to perform mathematic operations on those numbers
- 32-bit Format (Single precision)



  - 1 sign-bit (s), 8-bit Exponent (e), 23-bit Mantissa (m)
- 64-bit Format (Double precision)



  - 1 sign-bit, 11-bit Exponent, 52-bit Mantissa
- Interpretation: $Z = (-1)^s \cdot 1.m \cdot 2^{e-B}$
  - B = 127 (single precision) / 1023 (double precision)
  - B = offset/bias

Floating-Point Numbers
IEEE 754 Norm

- "Hidden bit": Normalized exponent means that the MSB is always 1
  - No need to represent → Save one-digit position (cost reduction)

- ∃ small (non-normalized) number, example: $0.01 \cdot 2^{-E_{\max}}$
  - These number are saved as denormalized numbers
    - $1.m \cdot 2^{-E_{\max}}$ instead of $1.m \cdot 2^{e}$

- Null represents the absolute Null
  - Too small numbers (Underflow) are rounded to Null → error propagation
    - Sign of number is preserved with on underflow
      - Negative small numbers are rounded to –0.0 and positive ones to +0.0

Floating-Point Numbers

IEEE 754 Norm

- Not a Number (NaN)
  - Is a convention used to represent invalid or non-defined results
    - Example: Division by zero (0.0/0.0)
- Valid number range

Exponent	Mantissa	Meaning
111...111	000...000	± infinite
111...111	≠ 000...000	NaN
000...000	000...000	± 0
000...000	≠ 000...000	Denormalized number
000...001 to 111…110	any	Normalized number

Floating-Point Numbers

IEEE 754 Norm

- Range: set of all possible valid numbers
  - Wit a *k*-bit mantissa the range of possible number is
    - $[\ -(1-2^{-k}) \cdot 2^{Emax}\ ,\ +(1-2^{-k}) \cdot 2^{Emax}\ ]$ (with the first normalized option)
    - $[\ -(2-2^{-k}) \cdot 2^{Emax}\ ,\ +(2-2^{-k}) \cdot 2^{Emax}\ ]$ (with the second normalized option)
    - Proof: k-digit range shift followed by a division of the extrema by 2^k
- Precision: smallest distance between two valid numbers
  - Large distance → smaller precision
  - The set of numbers depends on E_{max} , and the precision
  - Compromise must be found between range and precision for the application at hand

Floating-Point Numbers

Conversion

- Input: decimal number; Output: Binary number in IEEE 754 Format
- Step 1:
  - Convert integer part and fraction part in binary
  - Assemble the real number (Integer.Fraction)
- Step 2:
  - Normalize the resulting real number
    - Shift the comma left/right, right after the MSB
      - Increment the exponent for every left-shift position
      - Decrement the exponent for every shift to the right
- Step 3:
  - Convert the exponent Excess-Representation (127 rsp. 1023)
- Step 4
  - Represent the result using sign, exponent and mantissa (IEEE 754)

Floating-Point Numbers

Conversion

- Example: 78.8125_{10} in IEEE 754 single precision
  - Step 1 (Binary conversion of integer and fraction):
    - $78_{10} = 1001110_2$, $0.8125_{10} = 0.1101_2$
    - $78.8125_{10} = 1001110.01101_2$
  - Step 2 (Normalization):
    - 1001110.01101 = 1.00111001101E110 (Exponent = 6)
  - Step 3 (Convert into Excess-Representation):
    - 110 + 1111111 = 10000101 (6 + 127 = 133)
  - Step 4 (Final representation in IEEE 754 Format)
    - 0 10000101 00111001101000000000000

Floating-Point Numbers
Addition/Subtraction

- Given: $A = (-1)^{S_A} \cdot 1.m_A \cdot 2^{e_A}, B = (-1)^{S_B} \cdot 1.m_B \cdot 2^{e_B}$
  - We assume that: $e_A \leq e_B$

- Addition / Subtraction Algorithm
  1. Identify the operand with bigger exponent *e*
  2. shift the mantissa of the operand with smaller exponent to the left until the two exponents match (Exponent = *e*)
  3. perform the addition/subtraction of the resulting mantissa
  4. Normalize the result
  5. perform underflow- / overflow test

Floating-Point Numbers

Addition/Subtraction

- Example:

$$A = (-1)^0 \cdot 1.0011 \cdot 2^2 (4.75_{10})$$
$$B = (-1)^0 \cdot 1.10101 \cdot 2^3 (13.25_{10}) \qquad e_A \leq e_B$$

  - Step 1: Align the two exponents

$$A = (-1)^0 * 0.10011 * 2^3 \rightarrow m'_A = 0.10011$$

  - Step 2: compute the mantissa

$$m'_A + m_B = 0.10011 + 1.10101 = 10.01$$

  - Step 3: Normalize the result

$$10.01 \cdot 2^3 = 1.001 \cdot 2^4$$

  - Underflow- / Overflow test: No Underflow / Overflow

$$A + B = (-1)^0 \cdot 1.001 \cdot 2^4 (18_{10})$$

- This algorithm can be used without modification on IEEE 754-numbers

Floating-Point Numbers

Multiplication

- Multiplication

$$A = (-1)^{S_A} \cdot 1.m_A \cdot 2^{e_A}$$
$$B = (-1)^{S_B} \cdot 1.m_B \cdot 2^{e_B}$$
$$C = A \cdot B = (m_A \cdot m_B) \cdot 2^{e_A + e_B}$$

1. Sign = Product of the two signs
2. Exponent = $e_A + e_B$
3. Mantissa = $m_A * m_B$
4. Normalize the result, if needed
5. Underflow- / Overflow test

- In IEEE 754
  - The addition of the two exponents generates two Excess in result
  - Correction must be done by subtracting Excess from the result

Floating-Point Numbers

Multiplication

- Example: $A = (-1)^1 \cdot 1.01 \cdot 2^{10} \triangleq (-1)^1 \cdot 1.01 \cdot 2^{137}$ *(IEEE754)*

  $B = (-1)^0 \cdot 1.01 \cdot 2^{-5} \triangleq (-1)^0 \cdot 1.01 \cdot 2^{122}$ *(IEEE754)*

  - Step 1: Sign $(-1)^0 \cdot (-1)^1 = (-1)^1$
  - Step 2: Exponent:

    General case: $e_Z = 10 - 5 = 5$

    in IEEE 754: $e_Z = 137 + 122 = 259 - 127 = 132 \triangleq 5 + 127$
  - Step 3: Mantissa: $1.101 \cdot 1.01 = 10.00001$
  - Step 4: Normalize: $A \cdot B = (-1)^1 \cdot 10.00001 \cdot 2^5 = (-1)^1 \cdot 1.000001 \cdot 2^6$
  - Step 5: Underflow- / Overflow test: Number is normalized IEEE 754

UF Herbert Wertheim College of Engineering
UNIVERSITY of FLORIDA

POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE