INSTRUCTOR: *JIANGFANG YI* TUTOR: *YILONG LI*

ICS Tutorial 2016

Integers & Floating Point

INTEGER REPRESENTATION

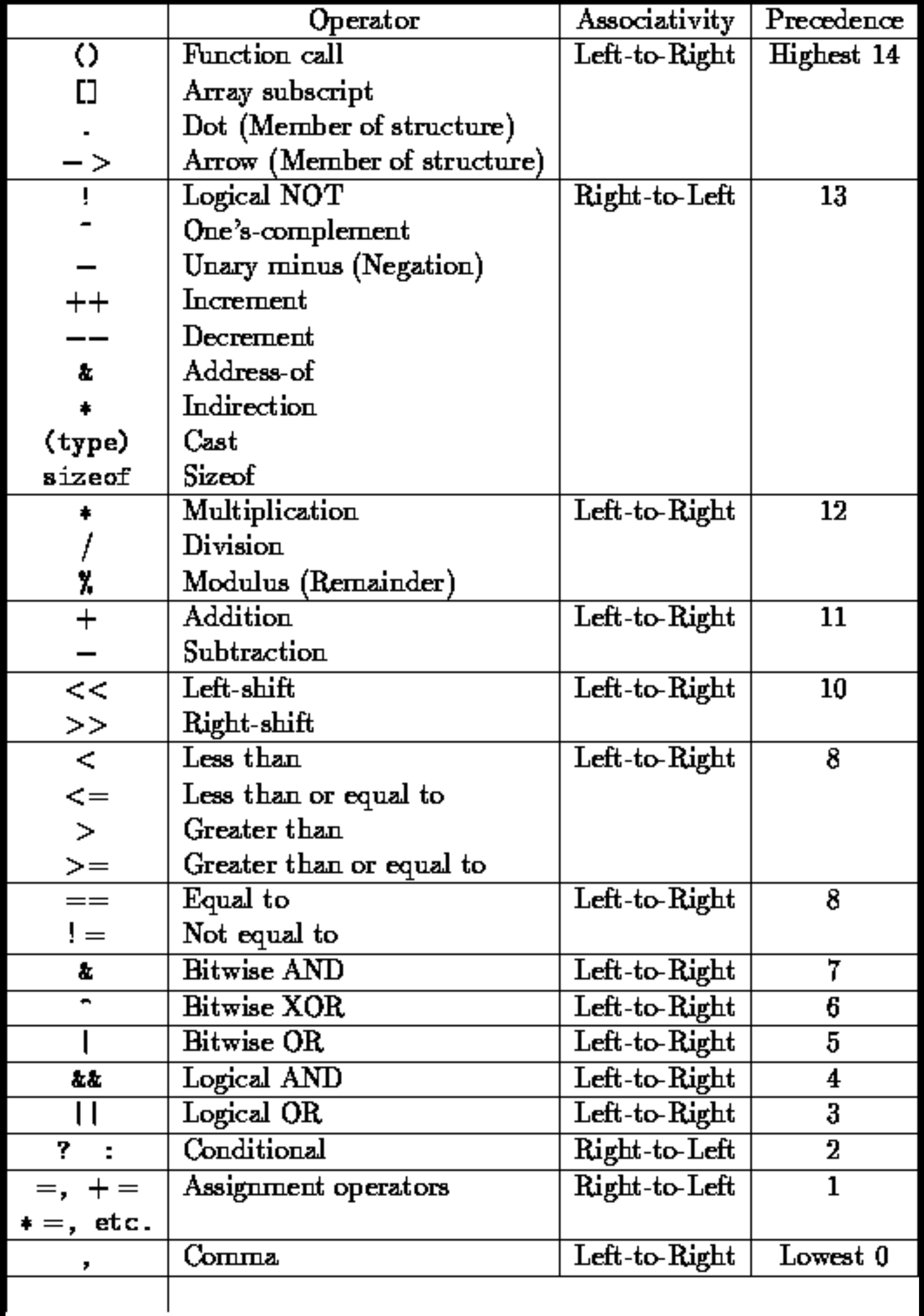
- Two’s complement representation
  - Advantages
  - Representation
- Overflow
  - a + b
  - a – b

BIT MANIPULATION

- Logical operation — return zero and non-zero
  - Unary: !x
    Binary: &&, ||
    Ternary: x ? y : z
- Bitwise operation
  - Unary: ~x
    Binary: &, |, ^, <<, >>

OPERATOR PRECEDENCE

- Example:
  - x << 3 + x
  - x > 0 ? x = 3 : x = 6
- Use parentheses when you are not sure



	Operator	Associativity	Precedence
()	Function call	Left-to-Right	Highest 14
[]	Array subscript		
.	Dot (Member of structure)		
− >	Arrow (Member of structure)		
!	Logical NOT	Right-to-Left	13
~	One's-complement		
−	Unary minus (Negation)		
++	Increment		
−−	Decrement		
&	Address-of		
*	Indirection		
(type)	Cast		
sizeof	Sizeof		
*	Multiplication	Left-to-Right	12
/	Division		
%	Modulus (Remainder)		
+	Addition	Left-to-Right	11
−	Subtraction		
<<	Left-shift	Left-to-Right	10
>>	Right-shift		
<	Less than	Left-to-Right	8
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equal to	Left-to-Right	8
! =	Not equal to		
&	Bitwise AND	Left-to-Right	7
^	Bitwise XOR	Left-to-Right	6
|	Bitwise OR	Left-to-Right	5
&&	Logical AND	Left-to-Right	4
||	Logical OR	Left-to-Right	3
? :	Conditional	Right-to-Left	2
=, + = * =, etc.	Assignment operators	Right-to-Left	1
,	Comma	Left-to-Right	Lowest 0

FLOATING POINTS

- Representation
  - IEEE 754 Std
- Normalized Values
- Denormalized Values
  - (abs) too small
  - +0, –0 // Q: how to produce +0, –0? Is (+0 == –0) true?
  - Inf, –Inf, NaN; // Q: how to produce Infs, NaNs?
- Distribution

FP ARITHMETICS

- a + b
- a * b
- a / 2^k

TYPE CONVERSION

- int -> float
- float -> int

HOW ROUNDING WORKS

CREATING FLOATING POINT NUMBER

s	exp	frac
1	4-bits	3-bits

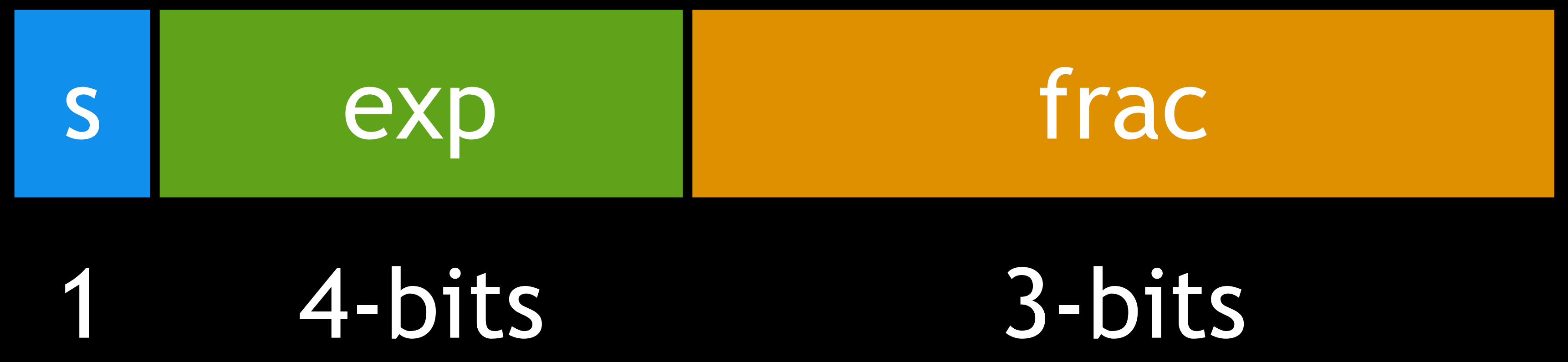
- Steps
  - Normalize to have leading 1
  - Round to fit within fraction
  - Postnormalize to deal with effects of rounding

- Case Study
  - Convert 8-bit unsigned numbers to tiny floating point format

Example Numbers

128	10000000
15	00001101
33	00010001
35	00010011
138	10001010
63	00111111

NORMALIZE

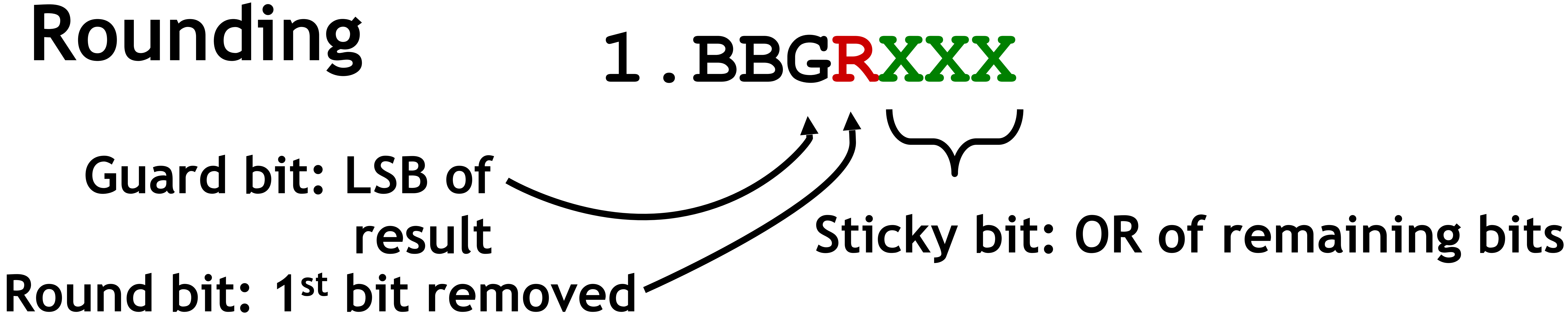


- Requirement
  - Set binary point so that numbers of form 1.xxxxx
  - Adjust all to have leading one

    Decrement exponent as shift left

Value	*Binary*	*Fraction*	*Exponent*
128	10000000	1.0000000	7
15	00001101	1.1010000	3
17	00010001	1.0001000	4
19	00010011	1.0011000	4
138	10001010	1.0001010	7
63	00111111	1.1111100	5

Rounding



1.BBGRXXX

Guard bit: LSB of result

Round bit: 1st bit removed

Sticky bit: OR of remaining bits

- Round up conditions
  - Round = 1, Sticky = 1 ➙ > 0.5
  - Guard = 1, Round = 1, Sticky = 0 ➙ Round to even

Value	*Fraction*	*GRS*	*Incr?*	*Rounded*
128	1.0000000	000	N	1.000
15	1.1010000	100	N	1.101
17	1.0001000	010	N	1.000
19	1.0011000	110	Y	1.010
138	1.0001010	011	Y	1.001
63	1.1111100	111	Y	10.000

POSTNORMALIZE

- Issue
  - Rounding may have caused overflow
  - Handle by shifting right once & incrementing exponent

Value	*Rounded*	*Exp*	*Adjusted*	*Numeric Result*
128	1.000	7		128
15	1.101	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000/6	64

INTERESTING NUMBERS

Description	*exp*	*frac*	*Numeric Value*
Zero			
Smallest Pos. Denorm.			
Largest Denormalized			
Smallest Pos. Normalized			
One			
Largest Normalized			

INTERESTING NUMBERS

Description	*exp*	*frac*	*Numeric Value*
Zero	00…00	00…00	0.0
Smallest Pos. Denorm.	00…00	00…01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
Single ≈ 1.4×10^{-45}			
Double ≈ 4.9×10^{-324}			
Largest Denormalized	00…00	11…11	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
Single ≈ 1.18×10^{-38}			
Double ≈ 2.2×10^{-308}			
Smallest Pos. Normalized	00…01	00…00	$1.0 \times 2^{-\{126,1022\}}$
Just larger than largest denormalized			
One	01…11	00…00	1.0
Largest Normalized	11…10	11…11	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
Single ≈ 3.4×10^{38}			
Double ≈ 1.8×10^{308}			

NOTES

- What happens when x << 32, x >> 32 ?
- What happens when (-1) << 3, (0xFFFFFFFFU) << 3 ?
- What happens when (-1) >> 3, (0xFFFFFFFFU) >> 3 ?
- xor: a ^ b ^ b = a

NOTES

- Swap two numbers:
  - `temp = x; x = y; y = temp;`
  - `x = x + y; y = x - y; x = x - y;`
  - `x = x ^ y; y = x ^ y; x = x ^ y;`

NOTES

- Type conversion
  - FP <-> Ints
  - Ints <-> Ints

INSTRUCTIONS ON DATALAB

- Use specific operators (bitwise, logical, arithmetic) to implement some functions

- eg. absVal(x), assume -Tmax ≤ x ≤ Tmax
  Legal ops: ! ~ & ^ | + << >>
  Max ops: 10

- 老虎吃天——无从下口

USE BASIC COMPONENTS

- **Predicates**
- $f(x) = 1$, iff $x = 0$; otherwise $f(\mathrm{x}) = 0$.
  - $f(x) = !x$
- $f(x) = 1$, iff $x < 0$; otherwise $f(x) = 0$
- $f(x) = 1$, iff $x > 0$; otherwise $f(x) = 0$

USE BASIC COMPONENTS

- **Predicates**
- $f(x) = 1$, iff $x < 0$; otherwise $f(x) = 0$
  - f(x) = (x >> 31) & 1
- $f(x) = 1$, iff $x > 0$; otherwise $f(x) = 0$
  - f(x) = ((~x + 1) >> 31) & 1
  - Can it be simplified?

USE BASIC COMPONENTS

- Masks
  - 0000110010100101
    & 0000111100011111
    = 0000110000000101
- Masks with predicate
  - f = 111111 … when x = 0, f = 0 when x = 1
    f(x) = x + (~0)

INSTRUCTIONS ON DATALAB

- eg. absVal(x), assume $-Tmax \le x \le Tmax$
  Legal ops: ! ~ & ^ | + << >>
  Max ops: 10
- Analysis
  - $x \ge 0$: abs(x) = x
  - $x < 0$: abs(x) = -x = ~x + 1

PSEUDOCODE

- int lt0_predicate = (x >> 31) & 1;
- int positive_mask = lt0_predicate + ~0;
- int negative_mask = ~positive_mask;
- return (positive_mask & x) | (negative_mask & (~x + 1))
- BETTER SOLUTION?

HOW TO CHECK YOUR SUBMIT

- (1) make; ./btest
  - Code correctness
- (2) ./dlc
  - Code validity
- (3) printf