Wellesley CS 240

Integer Representation

Bits, binary numbers, and bytes
Fixed-width representation of integers: unsigned and signed
Modular arithmetic and overflow

positional number representation

2	4	0	
100	10	1	*weight*
10^2	10^1	10^0	*weight*
2	1	0	*position*

$= 2 \times 10^2 + 4 \times 10^1 + 0 \times 10^0$

- **Base** determines:
  -
  -
- Each position holds a digit.
- Represented value =

binary = base 2

1	0	1	1	
8	4	2	1	*weight*
2^3	2^2	2^1	2^0	*weight*
3	2	1	0	*position*

$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

When ambiguous, subscript with base:

101_{10} Dalmatians (movie) 101_{ten}

101_2-Second Rule (folk wisdom for food safety) 101_{two}

irony

ex

Powers of 2:
memorize up to ≥ 2^{10} (in base ten)

Show powers, strategies.

ex

conversion and arithmetic

$19_{10} = ?_2$

$1001_2 = ?_{10}$

$240_{10} = ?_2$

$11010011_2 = ?_{10}$

$101_2 + 1011_2 = ?_2$

$1001011_2 \times 2_{10} = ?_2$

What do you call 4 bits?

byte = 8 bits

a.k.a. octet

Smallest unit of data

used by a typical modern computer

Binary 00000000_2 -- 11111111_2

Decimal 000_{10} -- 255_{10}

Hexadecimal 00_{16} -- FF_{16}

Programmer's hex notation (C, etc.):

`0xB4` = $B4_{16}$ = $B4_{hex}$

Octal (base 8) also useful.

Why do 240 students often confuse Halloween and Christmas?

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

word |wərd|, n.

Natural (fixed size) unit of data used by processor.

– Word size determines:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

MSB: most significant bit

LSB: least significant bit

fixed-size data representations

Java Data Type	C Data Type	(size in bytes) 32-bit word	64-bit word
boolean		1	1
byte	char	1	1
char		2	2
short	short int	2	2
int	int	4	4
float	float	4	4
	long int	4	8
double	double	8	8
long	long long	8	8
	long double	8	16

Depends on word size

Fixed-width integer encodings

Unsigned ⊂ ℕ **non-negative integers** only

Signed ⊂ ℤ both **negative** and **non-negative integers**

***n* bits** offer only **2^n distinct values.**

Terminology:

"Most-significant" bit(s) or "high-order" bit(s)

"Least-significant" bit(s) or "low-order" bit(s)

MSB 0110010110101001 **LSB**

(4-bit) unsigned integer representation

1	0	1	1
8	4	2	1
2^3	2^2	2^1	2^0
3	2	1	0

$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

weight

position

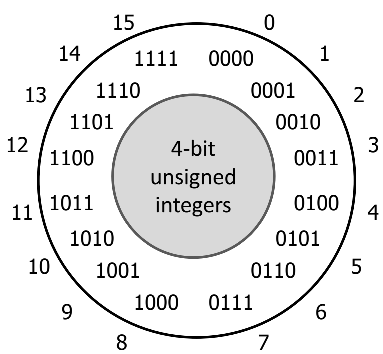
***n*-bit unsigned integers:**

minimum =

maximum =

modular arithmetic, overflow

11	1011
+ 2	+ 0010



13	1101
+ 5	+ 0101

x+y in *n*-bit unsigned arithmetic is in math

unsigned overflow =

=

Unsigned addition *overflows* if and only if

sign-magnitude

!!!

Most-significant bit (MSB) is ***sign bit***

0 means non-negative 1 means negative

Remaining bits are an unsigned magnitude

8-bit sign-magnitude:

00000000 represents _____

01111111 represents _____

10000101 represents _____

10000000 represents _____

Anything weird here?

Arithmetic?

Example:

4 - 3 != 4 + (-3)

```
 00000100
+10000011
```

Zero?

ex

(4-bit) **two's complement signed integer representation**

compare to unsigned

1	0	1	1
-2^3	2^2	2^1	2^0

$= 1 \times -2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

***4*-bit two's complement integers:**

minimum =

maximum =

two's complement vs. **unsigned**

_	_	... _	_	_
2^{n-1}	2^{n-2}	... 2^2	2^1	2^0
-2^{n-1}	2^{n-2}	... 2^2	2^1	2^0

unsigned places

two's complement places

What's the difference?

n-bit minimum =

n-bit maximum =

8-bit representations

ex

0 0 0 0 1 0 0 1

1 0 0 0 0 0 0 1

1 1 1 1 1 1 1 1

0 0 1 0 0 1 1 1

n-bit two's complement numbers:

minimum =

maximum =

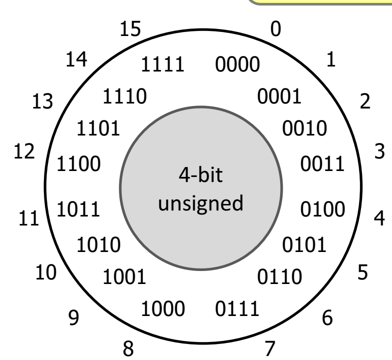
4-bit **unsigned** vs. 4-bit **two's complement**

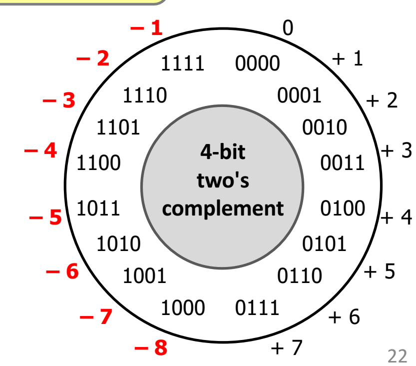
1 0 1 1

$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$1 \times -2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

11 ← **difference = ___ = 2—** → -5

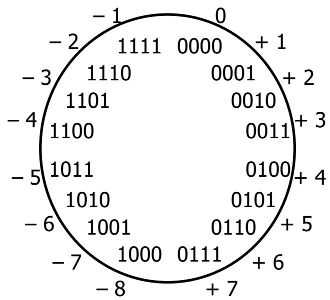




two's complement **addition**

2	0010	-2	1110
+ 3	+ 0011	+ -3	+ 1101

-2	1110	2	0010
+ 3	+ 0011	+ -3	+ 1101



Modular Arithmetic

two's complement ***overflow***

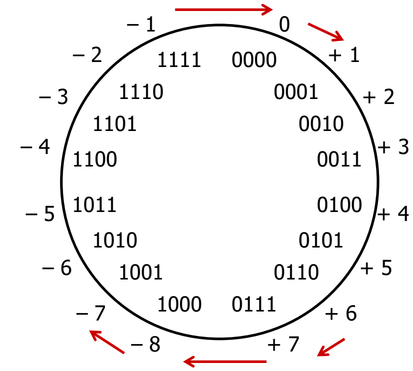
Addition *overflows*

if and only if

if and only if

-1	1111
+ 2	+ 0010

6	0110
+ 3	+ 0011



Modular Arithmetic

Some CPUs/languages raise exceptions on overflow.
C and Java cruise along silently... Feature? Oops?

A few reasons two's complement is awesome

Addition, subtraction, hardware

Sign

Negative one

Complement rules

Another derivation

ex

How should we represent 8-bit negatives?

- For all positive integers ***x***, ***x*** and ***–x*** should sum to zero.
- Use the standard addition algorithm.

00000001	00000010	00000011
+ 00000000	+ 00000000	+ 00000000

- Find a rule to represent –x where that works…