# Data representation

College of Saint Benedict & Saint John's University

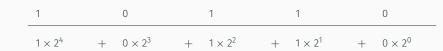
58036

5		8		0		3		6	
50000	+	8000	+	0	+	30	+	6	

5		8		0		3		6
50000	+	8000	+	0	+	30	+	6
5 × 10000		8 > 1000		0 × 100		3 > 10		6 × 1

5		8		0		3		6
50000	+	8000	+	0	+	30	+	6
5 × 10000	+	8 × 1000	+	0 × 100	+	3 × 10	+	6 × 1
5 × 10 <sup>4</sup>	+	8 × 10 <sup>3</sup>	+	0 × 10 <sup>2</sup>	+	3 × 10 <sup>1</sup>	+	6 × 10 <sup>0</sup>

10110



1	0	1	1	0
$1 \times 2^4$	$+ 0 \times 2^{3}$	$+ 1 \times 2^{2}$	$+ 1 \times 2^{1}$	$+ 0 \times 2^{0}$
1 × 16	+ 0 × 8	+ 1 × 4	+ 1 × 2	+ 0 × 1

1	0	1	1	0
1 × 2 <sup>4</sup>	$+ 0 \times 2^{3}$	$+ 1 \times 2^{2}$	$+ 1 \times 2^{1}$	$+ 0 \times 2^{0}$
1 × 16	+ 0 × 8	+ 1 × 4	+ 1 × 2	+ 0 × 1
16	+ 0	+ 4	+ 2	+ 0

• this is the representation for unsigned binary integers

# unsigned addition

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

# unsigned addition

$$0 + 0 = 0$$
 $0 + 1 = 1$ 
 $1 + 0 = 1$ 

1 + 1 = 10

 the hardware has a special bit known as the carry bit, denoted by C, which stores a 1 if the result of the addition was a carry, and 0 otherwise.

# more unsigned addition

ADD 1 1 0 0 0 0 = 4

$$C \leftarrow 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 =$$

$$0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 = +5$$

ADD 1 0 0 1 0 1 =

# signed addition

- designate first bit as 0 (+) or 1 (-)
- · what is the problem
  - in this case, we had two additional symbols, + and -, and we were making some assumptions about their behavior.
  - For example, we know that 5ADD5 = 10, but what does +ADD-equal? we have no rule for that in our definition of decimal.
  - we are trying to apply the addition algorithm, when we should be applying a different algorithm, called *subtraction*
  - can we choose a different representation that we can directly use the addition algorithm with?

5

### method of complements

- technique used to subtract one number from another using only addition of positive numbers
- represent negative numbers as two's complement of their positive counterparts

• so naturally we ask, well how to get the two's complement of a number?

### method of complements

- technique used to subtract one number from another using only addition of positive numbers
- represent negative numbers as two's complement of their positive counterparts

#### two's complement

- find one's complement
- · add one

• so naturally we ask, well how to get the two's complement of a number?

NOT 0 0 0 1 0 1

NOT	0	0	0	1	0	1
	1	1	1	0	1	0

one's complement

NOT 0 0 0 1 0 1

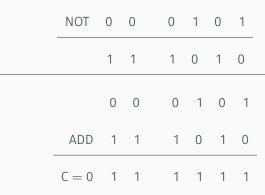
1 1 1 0 1 0

0 0 0 1 0 1

ADD 1 1 1 1 0 1 0

• in binary,one's complement is known as logical not

## one's complement



- in binary,one's complement is known as logical not
- adding the one's complement will always result in all 1s

## two's complement

NOT	0	0	0	1	0	1
	1	1	1	0	1	0
	0	0	0	1	0	1
ADD	1	1	1	0	1	0
C = 0	1	1	1	1	1	1
ADD	0	0	0	0	0	1

- in binary,one's complement is known as logical not
- adding the one's complement will always result in all 1s

## two's complement

	NOT	0	0	0	1	0	1
		1	1	1	0	1	0
		0	0	0	1	0	1
_	ADD	1	1	1	0	1	0
	C = 0	1	1	1	1	1	1
	ADD	0	0	0	0	0	1
	C = 1	0	0	0	0	0	0

- in binary,one's complement is known as logical not
- adding the one's complement will always result in all 1s
- so two's complement is NOT + 1

7

	1	1	1	0	1	0
ADD	0	0	0	0	0	1
	1	1	1	0	1	1

## cpu bits

	0	1
N	otherwise	result is negative
Z	otherwise	result is all zeros
V	otherwise	signed integer overflow occurred
С	otherwise	unsigned integer overflow occurred

- C is set to carry out of leftmost bit
- V is set to detects an overflow by comparing the carry into the leftmost bit with the C bit. If they are different, an overflow has occurred, and V gets 1. If they are the same, V gets 0.

9

# register transfer language

operation	RTL symbol
AND	$\wedge$
OR	$\vee$
XOR	$\oplus$
NOT	$\neg$
Implies	$\rightarrow$
Transfer	$\leftarrow$
Bit index	⟨ ⟩
Informal description	{ }
Sequential separator	•
Concurrent separator	,

register transfer language

operation	RTL symbol
AND	^
OR	V
XOR	$\oplus$
NOT	_
Implies	$\rightarrow$
Transfer	←
Bit index	()
Informal description	{ }
Sequential separator	;
Concurrent separator	,

$$c \leftarrow a \oplus b$$
;  $N \leftarrow c < 0, Z \leftarrow c = 0$ 

## another example

## another example

#### arithmetic shift

#### arithmetic shift left (asl)

$$C \leftarrow r\langle 0 \rangle$$
,  $r\langle 0..4 \rangle \leftarrow \langle 1..5 \rangle$ ,  $r\langle 5 \rangle \leftarrow 0$ ;  
 $N \leftarrow r < 0$ ,  $Z \leftarrow r = 0$ ,  $V \leftarrow \{\text{overflow}\}$ 

#### arithmetic shift right (asr)

?

- how will you assign V bit? what is the RTL?
- $V \leftarrow C = r\langle 0 \rangle$
- what is RTL for ASR?
- · where are the N and V bits for ASR?

#### arithmetic shift

#### arithmetic shift left (asl)

$$C \leftarrow r\langle 0 \rangle$$
,  $r\langle 0..4 \rangle \leftarrow \langle 1..5 \rangle$ ,  $r\langle 5 \rangle \leftarrow 0$ ;  
 $N \leftarrow r < 0$ ,  $Z \leftarrow r = 0$ ,  $V \leftarrow \{\text{overflow}\}$ 

#### arithmetic shift right (asr)

$$C \leftarrow r\langle 5 \rangle, \ r\langle 1...5 \rangle \leftarrow \langle 0...4 \rangle;$$
  
 $Z \leftarrow r = 0$ 

- how will you assign V bit? what is the RTL?
- $V \leftarrow C = r\langle 0 \rangle$
- what is RTL for ASR?
- · where are the N and V bits for ASR?

## hexadecimal

### unicode

Hello world.

¡Hola!, Grüß Gott, Hyvää päivää, Tere õhtust, Bonġu Cześć!, Dobrý den 你好, 早晨, こんにちは

- used to access >127 different characters
- backwards compatible with ASCII, i.e., code points have same value in UTF that they have in ASCII
- · comes in different flavors
  - UTF-32 easier to understand, requires 32bits to represent each of the code points (glyphs), but wasteful if you are mostly storing ascii characters
  - UTF-8 variable width code, i.e., some bits in the pattern are reserved for storing information about the structure of the pattern instead of information about the code point
- also define some emojis

### unicode

Hello world.

¡Hola!, Grüß Gott, Hyvää päivää, Tere õhtust, Bonġu Cześć!, Dobrý den 你好, 早晨, こんにちは

https://www.paypal.com

- used to access >127 different characters
- backwards compatible with ASCII, i.e., code points have same value in UTF that they have in ASCII
- · comes in different flavors
  - UTF-32 easier to understand, requires 32bits to represent each of the code points (glyphs), but wasteful if you are mostly storing ascii characters
  - UTF-8 variable width code, i.e., some bits in the pattern are reserved for storing information about the structure of the pattern instead of information about the code point
- also define some emojis
- can also be used for nefarious things

### unicode

Hello world.

¡Hola!, Grüß Gott, Hyvää päivää, Tere õhtust, Bonġu Cześć!, Dobrý den 你好, 早晨, こんにちは

https://www.paypal.com

- used to access >127 different characters
- backwards compatible with ASCII, i.e., code points have same value in UTF that they have in ASCII
- · comes in different flavors
  - UTF-32 easier to understand, requires 32bits to represent each of the code points (glyphs), but wasteful if you are mostly storing ascii characters
  - UTF-8 variable width code, i.e., some bits in the pattern are reserved for storing information about the structure of the pattern instead of information about the code point
- also define some emojis
- can also be used for nefarious things

# floating-point

#### **IEEE 754**

single precision 1.8.23 — excess 127 / 126 double precision 1.11.52 — excess 1023 / 1022 special values

	exponent	significand
zero	all zeros	all zeros
denormalized	all zeros	non-zero
inifinity	all ones	all zeros
not a number (NaN)	all ones	non-zero

#### operations that result in NaN

- The divisions 0/0 and  $\pm \infty / \pm \infty$
- The multiplications 0×± ∞ and ± ∞ ×0
- The additions  $\infty + (-\infty), (-\infty) + \infty$  and equivalent subtractions

- mantissa → significand
- more bits in exponent  $\rightarrow$  more range
- more bits in significand  $\rightarrow$  more precision
- how to find the range for floating-point numbers

# floating-point

#### **IEEE 754**

single precision 1.8.23 — excess 127 / 126 double precision 1.11.52 — excess 1023 / 1022 special values

	exponent	significand
zero	all zeros	all zeros
denormalized	all zeros	non-zero
inifinity	all ones	all zeros
not a number (NaN)	all ones	non-zero

#### operations that result in NaN

- The divisions 0/0 and  $\pm \infty / \pm \infty$
- The multiplications 0×± ∞ and ± ∞ ×0
- The additions  $\infty + (-\infty), (-\infty) + \infty$  and equivalent subtractions

- mantissa → significand
- more bits in exponent → more range
- more bits in significand → more precision
- how to find the range for floating-point numbers
- draw attention to Figure 3.38 make connection with radix sort —
  floats can be sorted using radix sort by NOT'ing the values then treating
  them like unsigned integers



except where otherwise noted, this worked is licensed under creative commons attribution-sharealike 4.0 international license