

Codes and number systems

Introduction to Computer

Yung-Yu Chuang

with slides by Nisan & Schocken (www.nand2tetris.org) and Harris & Harris (DDCA)

Coding



- Assume that you want to communicate with your friend with a flashlight in a night, what will you do?



light painting?
What's the problem?



Solution #1



- A: 1 blink
- B: 2 blinks
- C: 3 blinks
- :
- Z: 26 blinks

What's the problem?

- How are you? = 131 blinks

Solution #2: Morse code



A	· —	J	· — — —	S	· · ·
B	— · · ·	K	— · —	T	— ·
C	— · — ·	L	· — · ·	U	· · —
D	— · ·	M	— —	V	· · · —
E	·	N	— ·	W	· — —
F	· · — ·	O	— — —	X	— · · —
G	— — ·	P	· — — ·	Y	— · — —
H	· · · ·	Q	— — · —	Z	— — · ·
I	· ·	R	· — ·		

· · · · · · · — — — · · · — — — · · — — — — —

Hello

Lookup



- It is easy to translate into Morse code than reverse. Why?

Lookup



.	E
-	T

• • • • •

•

• — • •

• — • •

— — — —

• •	I	— •	N
• —	A	— —	M

• • •	S	— • •	D
• • —	U	— • —	K
• — •	R	— — •	G
• — —	W	— — —	O

• • • •	H	— • • •	B
• • • —	V	— • • —	X
• • — •	F	— • — •	C
• • — —	Ü	— • — —	Y
• — • •	L	— — • •	Z
• — • —	Ä	— — • —	Q
• — — •	P	— — — •	Ö
• — — —	J	— — — —	§

Number of
Dots and Dashes

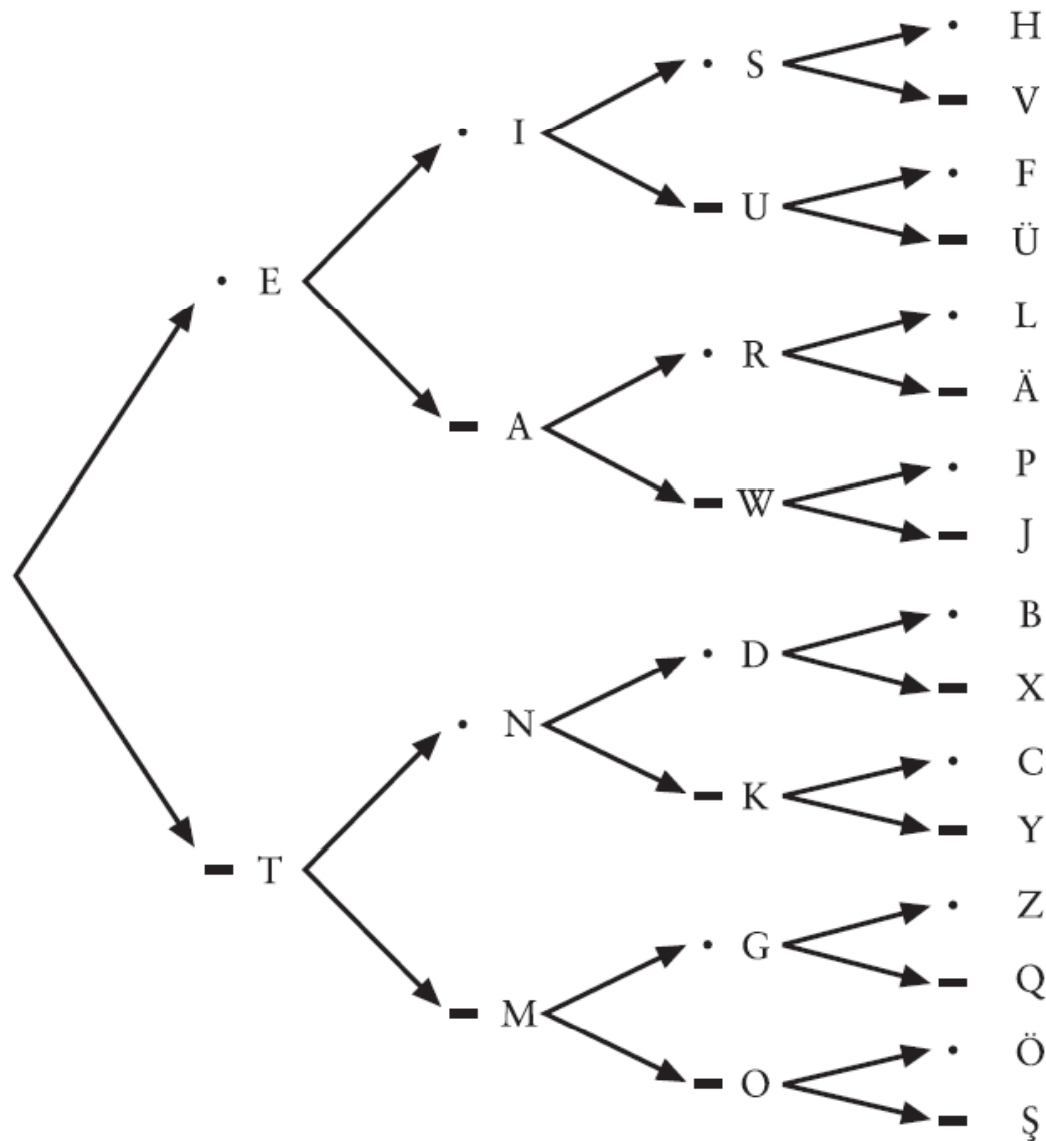
1
2
3
4

Number of Codes

2
4
8
16

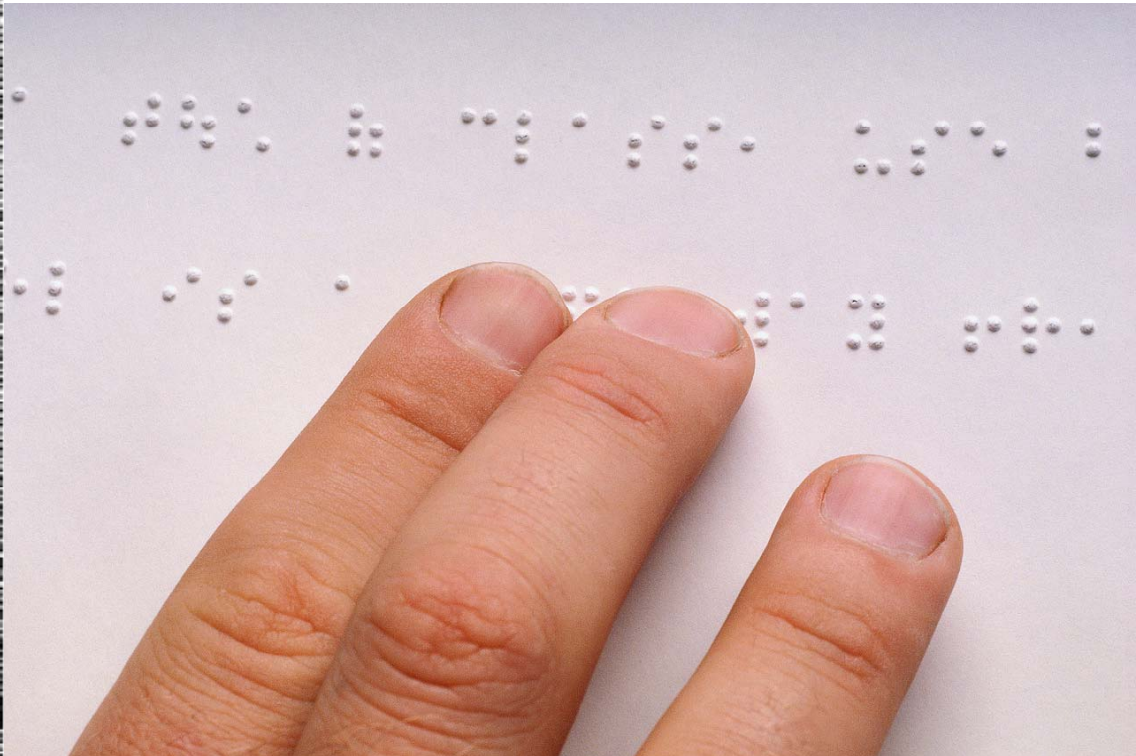
number of codes = $2^{\text{number of dots and dashes}}$

Lookup



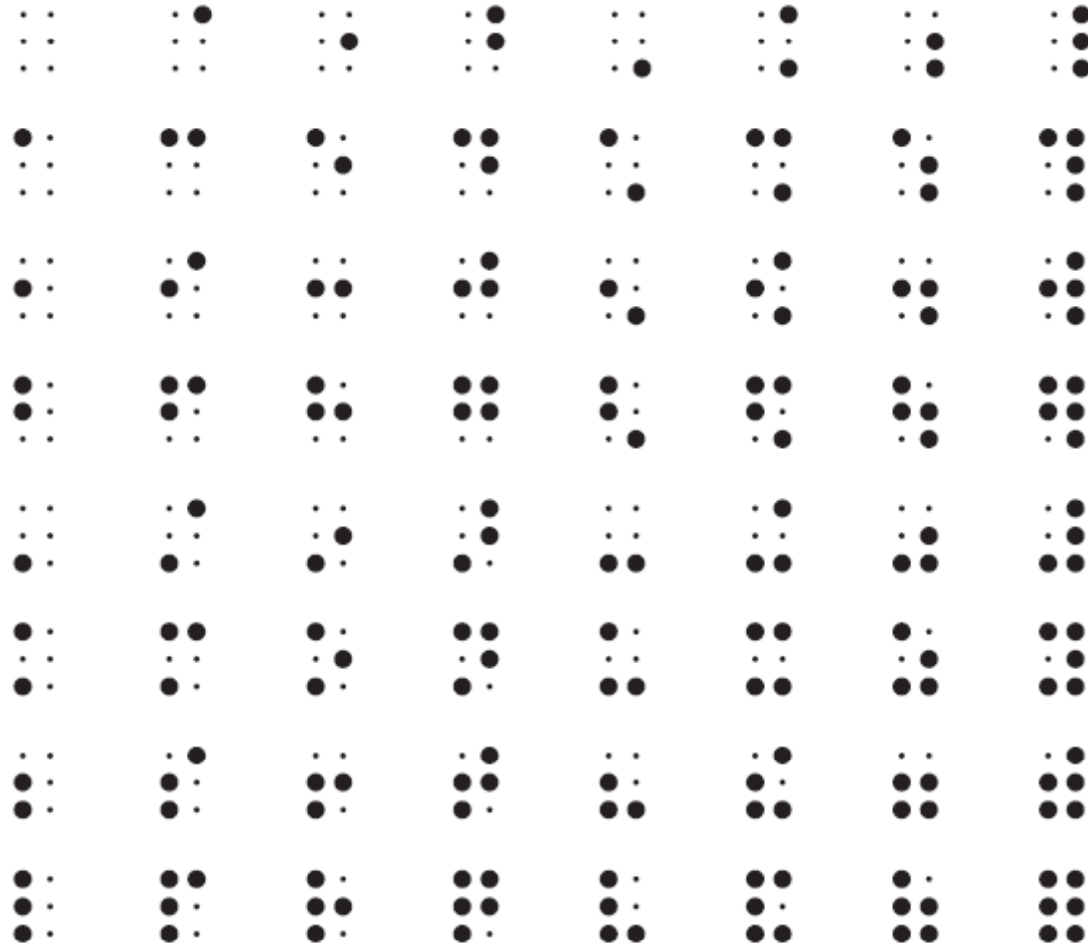
Useful for
checking the
correctness/
redundancy

Braille



1	○	○	4
2	○	○	5
3	○	○	6

Braille



What's common in these codes?

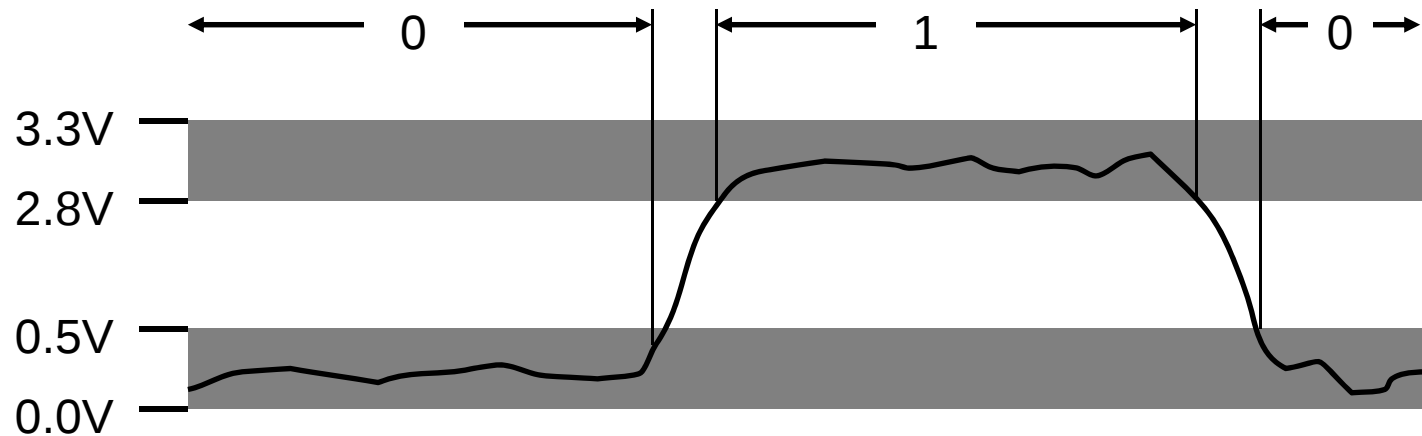


- They are both binary codes.

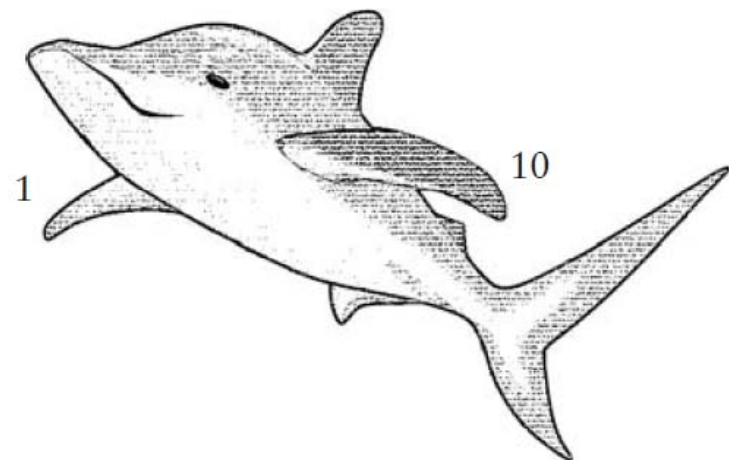
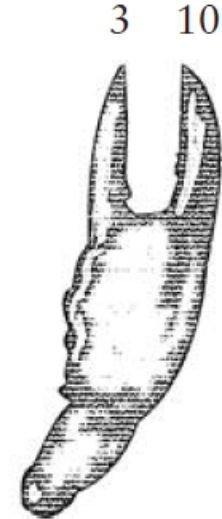
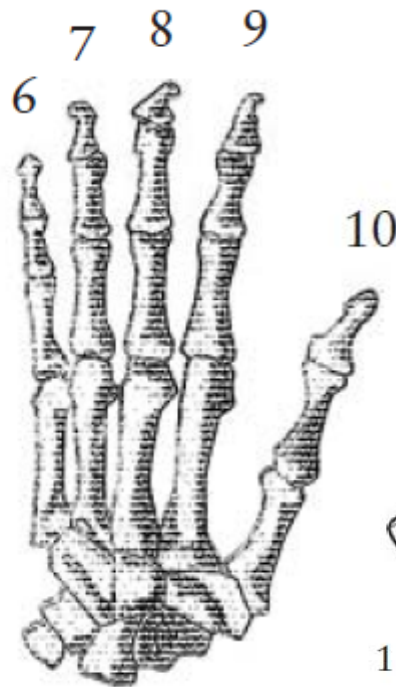
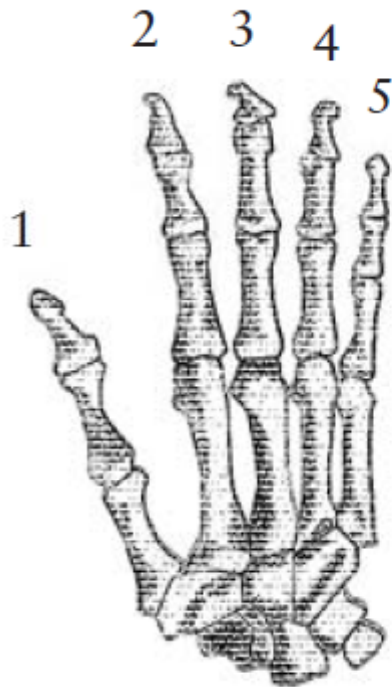
Binary representations



- Electronic Implementation
 - Easy to store with bistable elements
 - Reliably transmitted on noisy and inaccurate wires



Number systems



Number Systems

- Decimal numbers

1's column
10's column
100's column
1000's column

$$5374_{10} =$$

- Binary numbers

1's column
2's column
4's column
8's column

$$1101_2 =$$

Number Systems

- Decimal numbers

1's column
10's column
100's column
1000's column

$$5374_{10} = 5 \text{ ? } 10^3 + 3 \text{ ? } 10^2 + 7 \text{ ? } 10^1 + 4 \text{ ? } 10^0$$

five three seven four
thousands hundreds tens ones

- Binary numbers

1's column
2's column
4's column
8's column

$$1101_2 = 1 \text{ ? } 2^3 + 1 \text{ ? } 2^2 + 0 \text{ ? } 2^1 + 1 \text{ ? } 2^0 = 13_{10}$$

one one no one
eight four two one

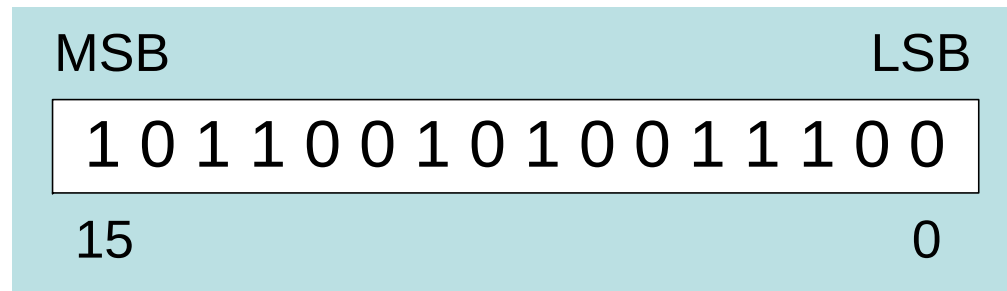


Binary numbers



- Digits are 1 and 0
(a binary digit is called a bit)
1 = true
0 = false
- MSB –most significant bit
- LSB –least significant bit

- Bit numbering:



- A bit string could have different interpretations

Powers of Two

- $2^0 =$
- $2^1 =$
- $2^2 =$
- $2^3 =$
- $2^4 =$
- $2^5 =$
- $2^6 =$
- $2^7 =$
- $2^8 =$
- $2^9 =$
- $2^{10} =$
- $2^{11} =$
- $2^{12} =$
- $2^{13} =$
- $2^{14} =$
- $2^{15} =$

Powers of Two

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$
- Handy to memorize up to 2^9

Unsigned binary integers



- Each digit (bit) is either 1 or 0
- Each bit represents a power of 2:

1	1	1	1	1	1	1	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Table 1-3 Binary Bit Position Values.

2^n	Decimal Value	2^n	Decimal Value
2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Every binary number is a sum of powers of 2

Translating binary to decimal



Weighted positional notation shows how to calculate the decimal value of each binary bit:

$$dec = (D_{n-1} \times 2^{n-1}) + (D_{n-2} \times 2^{n-2}) + \dots + (D_1 \times 2^1) + (D_0 \times 2^0)$$

D = binary digit

binary 00001001 = decimal 9:

$$(1 \times 2^3) + (1 \times 2^0) = 9$$

Translating unsigned decimal to binary



- Repeatedly divide the decimal integer by 2. Each remainder is a binary digit in the translated value:

Division	Quotient	Remainder
$37 / 2$	18	1
$18 / 2$	9	0
$9 / 2$	4	1
$4 / 2$	2	0
$2 / 2$	1	0
$1 / 2$	0	1

$$37 = 100101$$

FROM ZERO TO ONE

- Decimal to binary conversion:
 - Convert 10011_2 to decimal
- Decimal to binary conversion:
 - Convert 47_{10} to binary



Number Conversion

- Decimal to binary conversion:
 - Convert 10011_2 to decimal
 - $16 \times 1 + 8 \times 0 + 4 \times 0 + 2 \times 1 + 1 \times 1 = 19_{10}$
- Decimal to binary conversion:
 - Convert 47_{10} to binary
 - $32 \times 1 + 16 \times 0 + 8 \times 1 + 4 \times 1 + 2 \times 1 + 1 \times 1 = 101111_2$

FROM ZERO TO ONE

- # FROM ZERO TO ONE



Binary Values and Range

- N -digit decimal number
 - How many values? 10^N
 - Range? $[0, 10^N - 1]$
 - Example: 3-digit decimal number:
 - $10^3 = 1000$ possible values
 - Range: $[0, 999]$
- N -bit binary number
 - How many values? 2^N
 - Range: $[0, 2^N - 1]$
 - Example: 3-digit binary number:
 - $2^3 = 8$ possible values
 - Range: $[0, 7] = [000_2 \text{ to } 111_2]$

Integer storage sizes



Standard sizes:

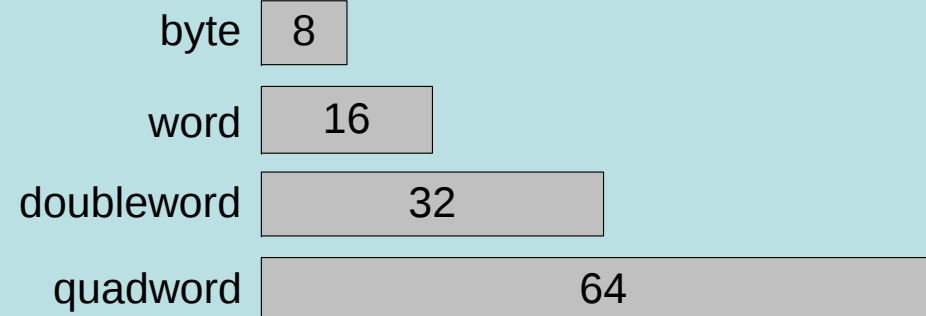


Table 1-4 Ranges of Unsigned Integers.

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to ($2^8 - 1$)
Unsigned word	0 to 65,535	0 to ($2^{16} - 1$)
Unsigned doubleword	0 to 4,294,967,295	0 to ($2^{32} - 1$)
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to ($2^{64} - 1$)

Practice: What is the largest unsigned integer that may be stored in 20 bits?

Bits, Bytes, Nibbles...

- Bits

10010110
most significant bit least significant bit

- Bytes & Nibbles

byte
10010110
nibble

- Bytes

CEBF9AD7
most significant byte least significant byte

Large Powers of Two

- $2^{10} = 1 \text{ kilo} \approx 1000 \text{ (1024)}$
- $2^{20} = 1 \text{ mega} \approx 1 \text{ million (1,048,576)}$
- $2^{30} = 1 \text{ giga} \approx 1 \text{ billion (1,073,741,824)}$

Estimating Powers of Two

- What is the value of 2^{24} ?
- How many values can a 32-bit variable represent?

Estimating Powers of Two

- What is the value of 2^{24} ?
 - $2^4 \times 2^{20} \approx 16$ million
- How many values can a 32-bit variable represent?
 - $2^2 \times 2^{30} \approx 4$ billion

Large measurements



- Kilobyte (KB), 2^{10} bytes
- Megabyte (MB), 2^{20} bytes
- Gigabyte (GB), 2^{30} bytes
- Terabyte (TB), 2^{40} bytes
- Petabyte
- Exabyte
- Zettabyte
- Yottabyte

Hexadecimal Numbers

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	
8	8	
9	9	
A	10	
B	11	
C	12	
D	13	
E	14	
F	15	

Hexadecimal Numbers

Hex Digit	Decimal Equivalent	Binary Equivalent
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

Hexadecimal Numbers

- Base 16
- Shorthand for binary

Translating binary to hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits.
- Example: Translate the binary integer 00010110101001110010100 to hexadecimal:

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100

Converting hexadecimal to decimal



- Multiply each digit by its corresponding power of 16:

$$\text{dec} = (D_3 \times 16^3) + (D_2 \times 16^2) + (D_1 \times 16^1) + (D_0 \times 16^0)$$

- Hex 1234 equals $(1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1) + (4 \times 16^0)$, or decimal 4,660.
- Hex 3BA4 equals $(3 \times 16^3) + (11 * 16^2) + (10 \times 16^1) + (4 \times 16^0)$, or decimal 15,268.

FROM ZERO TO ONE

- # FROM ZERO TO ONE



FROM ZERO TO ONE

- # FROM ZERO TO ONE

Powers of 16



Used when calculating hexadecimal values up to 8 digits long:

16^n	Decimal Value	16^n	Decimal Value
16^0	1	16^4	65,536
16^1	16	16^5	1,048,576
16^2	256	16^6	16,777,216
16^3	4096	16^7	268,435,456

Converting decimal to hexadecimal



Division	Quotient	Remainder
$422 / 16$	26	6
$26 / 16$	1	A
$1 / 16$	0	1

decimal 422 = 1A6 hexadecimal

Addition

- Decimal

$$\begin{array}{r} 11 \leftarrow \text{carries} \\ 3734 \\ + 5168 \\ \hline 8902 \end{array}$$

- Binary

$$\begin{array}{r} 11 \leftarrow \text{carries} \\ 1011 \\ + 0011 \\ \hline 1110 \end{array}$$

Binary Addition Examples

- Add the following 4-bit binary numbers

$$\begin{array}{r} 1001 \\ + 0101 \\ \hline \end{array}$$

- Add the following 4-bit binary numbers

$$\begin{array}{r} 1011 \\ + 0110 \\ \hline \end{array}$$

Binary Addition Examples

- Add the following 4-bit binary numbers

$$\begin{array}{r} 1 \\ 1001 \\ + 0101 \\ \hline 1110 \end{array}$$

- Add the following 4-bit binary numbers

$$\begin{array}{r} 111 \\ 1011 \\ + 0110 \\ \hline 10001 \end{array}$$

Overflow!

Overflow

- Digital systems operate on a **fixed number of bits**
- Overflow: when result is too big to fit in the available number of bits
- See previous example of $11 + 6$

Hexadecimal addition



Divide the sum of two digits by the number base (16). The quotient becomes the carry value, and the remainder is the sum digit.

		1	1
36	28	28	6A
42	45	58	4B
78	6D	80	B5

Important skill: Programmers frequently add and subtract the addresses of variables and instructions.

FROM ZERO TO ONE

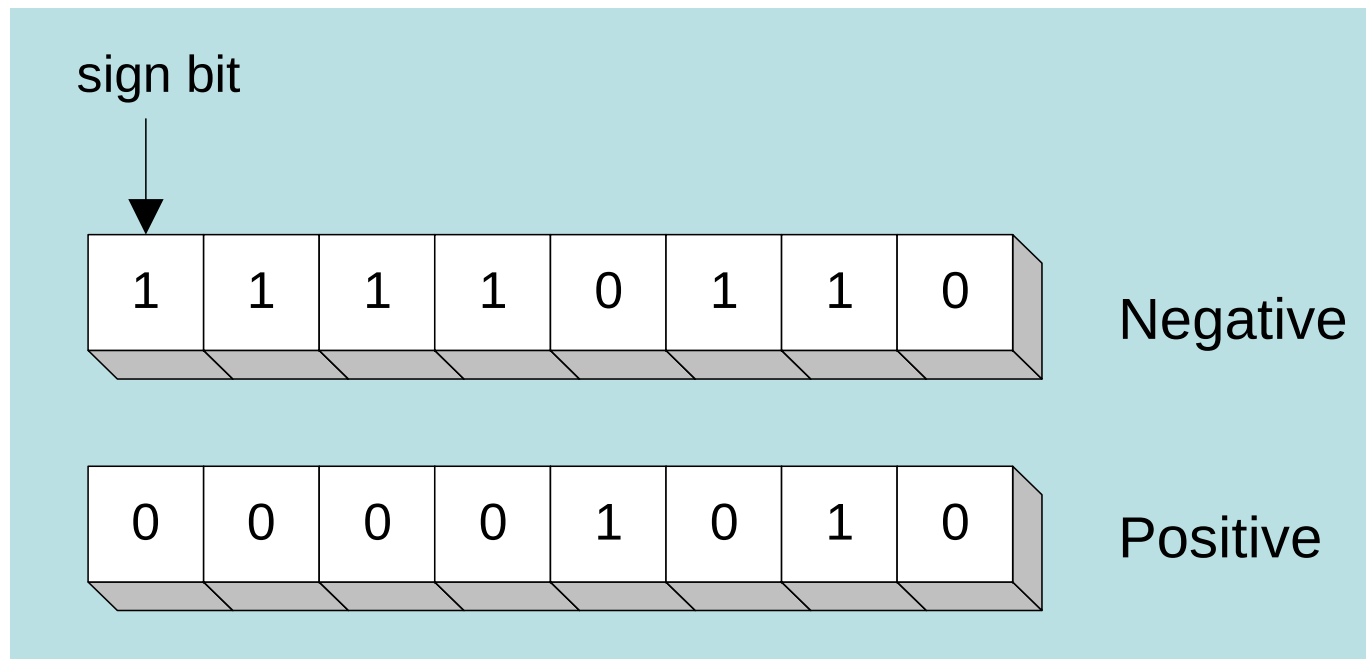
- Sign/Magnitude Numbers
- Two's Complement Numbers



Signed integers



The highest bit indicates the sign. 1 = negative,
0 = positive



If the highest digit of a hexadecimal integer is > 7 , the value is negative. Examples: 8A, C5, A2, 9D

Sign/Magnitude Numbers

- 1 sign bit, $N-1$ magnitude bits
- Sign bit is the most significant (left-most) bit
 - Positive number: sign bit = 0 $A: \{a_{N-1}, a_{N-2}, \dots, a_2, a_1, a_0\}$
 - Negative number: sign bit = 1 $A = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} a_i 2^i$
- Example, 4-bit sign/mag representations of ± 6 :
 - +6 =
 - 6 =
- Range of an N -bit sign/magnitude number:

Sign/Magnitude Numbers

- 1 sign bit, $N-1$ magnitude bits
- Sign bit is the most significant (left-most) bit
 - Positive number: sign bit = 0 $A: \{a_{N-1}, a_{N-2}, \dots, a_2, a_1, a_0\}$
 - Negative number: sign bit = 1 $A = (-1)^{a_{n-1}} \sum_{i=0}^{n-2} a_i 2^i$
- Example, 4-bit sign/mag representations of ± 6 :
 - +6 = **0110**
 - 6 = **1110**
- Range of an N -bit sign/magnitude number:
 $[-(2^{N-1}-1), 2^{N-1}-1]$

Sign/Magnitude Numbers

- Problems:
 - Addition doesn't work, for example $-6 + 6$:

$$\begin{array}{r} 1110 \\ + 0110 \\ \hline 10100 \text{ (wrong!)} \end{array}$$

- Two representations of 0 (± 0):

1000

0000

Two's Complement Numbers

- Don't have same problems as sign/magnitude numbers:
 - Addition works
 - Single representation for 0

Two's complement notation



Steps:

- Complement (reverse) each bit
- Add 1

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two's complement representation	11111111

Note that $00000001 + 11111111 = 00000000$

“Taking the Two’s Complement”

- Flip the sign of a two’s complement number
- Method:
 1. Invert the bits
 2. Add 1
- Example: Flip the sign of $3_{10} = 0011_2$

“Taking the Two’s Complement”

- Flip the sign of a two’s complement number
- Method:
 1. Invert the bits
 2. Add 1
- Example: Flip the sign of $3_{10} = 0011_2$

1. 1100

2. + 1

1101 = -3_{10}



Two's Complement Examples

- Take the two's complement of $6_{10} = 0110_2$
- What is the decimal value of 1001_2 ?

Two's Complement Examples

- Take the two's complement of $6_{10} = 0110_2$

1. 1001

2. $\begin{array}{r} + \quad 1 \\ \hline \end{array}$

$1010_2 = -6_{10}$

- What is the decimal value of the two's complement number 1001_2 ?

1. 0110

2. $\begin{array}{r} + \quad 1 \\ \hline \end{array}$

$0111_2 = 7_{10}$, so $1001_2 = -7_{10}$



Binary subtraction



- When subtracting $A - B$, convert B to its two's complement
- Add A to $(-B)$

$$\begin{array}{r} 01010 \\ - 01011 \\ \hline \end{array} \longrightarrow \begin{array}{r} 01010 \\ 10100 \\ \hline 11111 \end{array}$$

Advantages for 2's complement:

- No two 0's
- Sign bit
- Remove the need for separate circuits for add and sub

Two's Complement Addition

- Add $6 + (-6)$ using two's complement numbers

$$\begin{array}{r} 0110 \\ + 1010 \\ \hline \end{array}$$

- Add $-2 + 3$ using two's complement numbers

$$\begin{array}{r} 1110 \\ + 0011 \\ \hline \end{array}$$

Two's Complement Addition

- Add $6 + (-6)$ using two's complement numbers

$$\begin{array}{r} 111 \\ 0110 \\ + 1010 \\ \hline 10000 \end{array}$$

- Add $-2 + 3$ using two's complement numbers

$$\begin{array}{r} 111 \\ 1110 \\ + 0011 \\ \hline 10001 \end{array}$$

FROM ZERO TO ONE

- # FROM ZERO TO ONE



Sign-Extension

- Sign bit copied to msb's
- Number value is same
- **Example 1:**
 - 4-bit representation of 3 = 0011
 - 8-bit sign-extended value: 00000011
- **Example 2:**
 - 4-bit representation of -5 = 1011
 - 8-bit sign-extended value: 11111011

Zero-Extension

- Zeros copied to msb's
- Value changes for negative numbers

- **Example 1:**

- 4-bit value = $0011_2 = 3_{10}$
- 8-bit zero-extended value: $00000011 = 3_{10}$

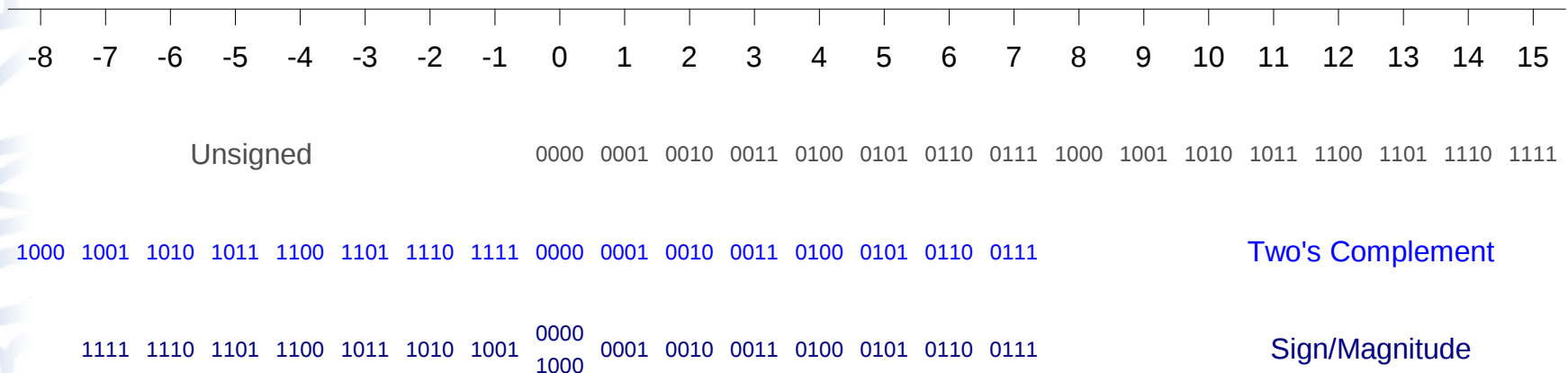
- **Example 2:**

- 4-bit value = $1011 = -5_{10}$
- 8-bit zero-extended value: $00001011 = 11_{10}$

Number System Comparison

Number System	Range
Unsigned	$[0, 2^N-1]$
Sign/Magnitude	$[-(2^{N-1}-1), 2^{N-1}-1]$
Two's Complement	$[-2^{N-1}, 2^{N-1}-1]$

For example, 4-bit representation:



Ranges of signed integers



The highest bit is reserved for the sign. This limits the range:

Storage Type	Range (low–high)	Powers of 2
Signed byte	–128 to +127	-2^7 to $(2^7 - 1)$
Signed word	–32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Signed doubleword	–2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31} - 1)$
Signed quadword	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

Character



- Character sets
 - Standard ASCII(0 – 127)
 - Extended ASCII (0 – 255)
 - ANSI (0 – 255)
 - Unicode (0 – 65,535)
- Null-terminated String
 - Array of characters followed by a *null byte*
- Using the ASCII table
 - back inside cover of book

DECIMAL VALUE	➡	0	16	32	48	64	80	96	112
➡	HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7
0	0	BLANK (NULL)	▶	BLANK (SPACE)	0	@	P	'	p
1	1	☺	◀	!	1	A	Q	a	q
2	2	☹	↕	"	2	B	R	b	r
3	3	♥	!!	#	3	C	S	c	s
4	4	♦	¶	\$	4	D	T	d	t
5	5	♣	§	%	5	E	U	e	u
6	6	♠	—	&	6	F	V	f	v
7	7	•	↕	'	7	G	W	g	w
8	8	•	↑	(	8	H	X	h	x
9	9	○	↓	)	9	I	Y	i	y
10	A	○	→	*	:	J	Z	j	z
11	B	♂	←	+	;	K	I	k	{
12	C	♀	└	,	<	L	\	l	!
13	D	♪	↔	—	=	M	I	m	}
14	E	♪	▲	.	>	N	^	n	~
15	F	☼	▼	/	?	O	_	o	△

DECIMAL VALUE	➡	128	144	160	176	192	208	224	240
➡	HEXA DECIMAL VALUE	8	9	A	B	C	D	E	F
0	0	Ç	É	á	⋮	└	└	∞	≡
1	1	ü	æ	í	⋮	└	└	β	±
2	2	é	Æ	ó	⋮	└	└	Γ	≥
3	3	â	ô	ú	└	└	└	π	≤
4	4	ä	ö	ñ	└	└	└	Σ	∫
5	5	à	ò	Ñ	└	└	└	σ	∫
6	6	å	û	ä	└	└	└	μ	÷
7	7	ç	ù	o	└	└	└	τ	≈
8	8	ê	ÿ	ı	└	└	└	ø	°
9	9	ë	Ö	└	└	└	└	θ	•
10	A	è	Ü	└	└	└	└	Ω	•
11	B	ï	ç	½	└	└	└	δ	√
12	C	î	£	¼	└	└	└	∞	n
13	D	ì	¥	ı	└	└	└	φ	2
14	E	Ä	℞	«	└	└	└	∈	■
15	F	Å	ƒ	»	└	└	└	∩	BLANK 'FF'

Representing Instructions



```
int sum(int x, int y)
{
    return x+y;
}
```

- For this example, Alpha & Sun use two 4-byte instructions
 - Use differing numbers of instructions in other cases
- PC uses 7 instructions with lengths 1, 2, and 3 bytes
 - Same for NT and for Linux
 - NT / Linux not fully binary compatible

Alpha sum

00
00
30
42
01
80
FA
6B

Sun sum

81
C3
E0
08
90
02
00
09

PC sum

55
89
E5
8B
45
0C
03
45
08
89
EC
5D
C3

Different machines use totally different instructions and encodings