

# Chapter 2

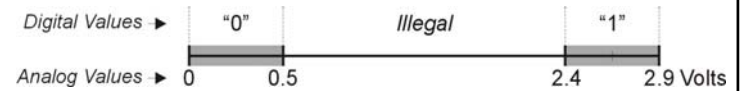
## Bits, Data Types, and Operations

Based on slides © McGraw-Hill  
Additional material © 2004/2005 Lewis/Martin

### How do we represent data in a computer?

At the lowest level, a computer has electronic “plumbing”

- Operates by controlling the flow of electrons



Easy to recognize two conditions:

- Presence of a voltage – we’ll call this state “1”
- Absence of a voltage – we’ll call this state “0”

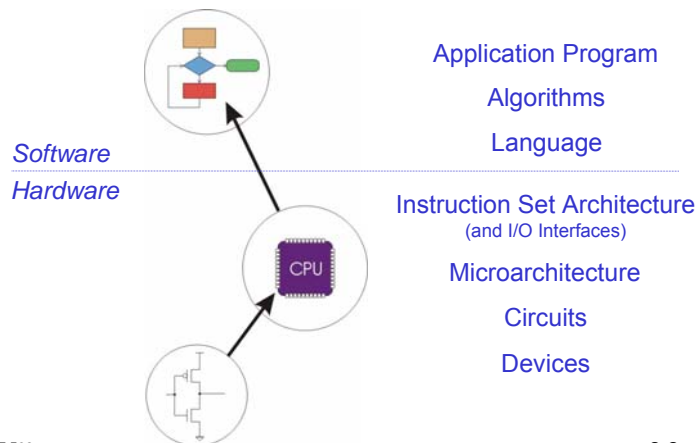
Alternative: Base state on *value* of voltage

- On/Off light switch versus dimmer switch
- Problem: Control/detection circuits more complex

CIT 593

2-3

### Last Class



CIT 593

2-2

### Computer is a binary digital system

Digital system:

- Finite number of symbols

Binary (base two) system:

- Has two states: 0 and 1

Basic unit of information: the *binary digit*, or *bit*

3+ state values require multiple bits

- A collection of **two** bits has **four** possible states:  
**00, 01, 10, 11**
- A collection of **three** bits has **eight** possible states:  
**000, 001, 010, 011, 100, 101, 110, 111**
- A collection of **n** bits has **2<sup>n</sup>** possible states

CIT 593

2-4

## Data

### What kinds of data do we need to represent?

- **Numbers** – signed, unsigned, integers, real, floating point, complex, rational, irrational, ...
- **Text** – characters, strings, ...
- **Images** – pixels, colors, shapes, ...
- **Logical** – true, false
- .....

### Data type:

- *Representation* and *operations* within the computer

We'll start with numbers...

CIT 593

2-5

## Unsigned Integers (cont.)

An  $n$ -bit unsigned integer represents  $2^n$  values

- From 0 to  $2^n - 1$

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

CIT 593

2-7

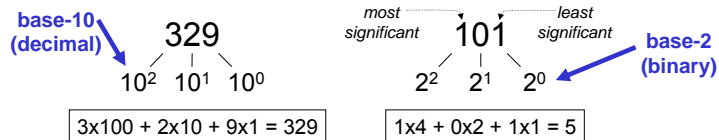
## Unsigned Integers

### Non-positional notation

- Could represent a number ("5") with a string of ones ("11111")
- Problems?

### Weighted positional notation

- Like decimal numbers: "329"
- "3" is worth 300, because of its position, while "9" is only worth 9



CIT 593

2-6

## Operation on Unsigned Data

Base-2 addition – just like base-10!

- Add from right to left, propagating carry

$$\begin{array}{r}
 10010 \text{ (18)} \\
 + \underline{01001} \text{ (9)} \\
 \hline
 11011 \text{ (27)}
 \end{array}
 \quad
 \begin{array}{r}
 \overset{\text{carry}}{\curvearrowright} \\
 10010 \text{ (18)} \\
 + \underline{01011} \text{ (11)} \\
 \hline
 11101 \text{ (29)}
 \end{array}
 \quad
 \begin{array}{r}
 \overset{\curvearrowright}{\curvearrowright}{\curvearrowright}{\curvearrowright} \\
 01111 \text{ (15)} \\
 + \underline{00001} \text{ (1)} \\
 \hline
 10000 \text{ (16)}
 \end{array}$$
  

$$\begin{array}{r}
 10111 \text{ (23)} \\
 + \underline{111} \text{ (7)} \\
 \hline
 11110 \text{ (30)}
 \end{array}$$

CIT 593

2-8

## Signed Integers

With  $n$  bits, we have  $2^n$  distinct values

- Assign “half” to positive integers (1 through  $\sim 2^{n-1}$ ) and “half” to negative ( $\sim -2^{n-1}$  through -1)
- That leaves two values: one for 0, and one extra

### Positive integers

- Just like unsigned with zero in most significant bit  
 $00101 = 5$

### Negative integers

- Sign-magnitude: set high-order bit to show negative, other bits are the same as unsigned  
 $10101 = -5$
- One’s complement: flip every bit to represent negative  
 $11010 = -5$
- In either case, MS bit indicates sign: **0=positive, 1=negative**

CIT 593

2-9

## Two’s Complement

### Idea

- Find representation to make arithmetic simple and consistent

### Specifics

- For each positive number ( $X$ ), assign value to its negative ( $-X$ ), such that  $X + (-X) = 0$  with “normal” addition, ignoring carry out

$$\begin{array}{r} 00101 \quad (5) \\ + \underline{11011} \quad (-5) \\ \hline 00000 \quad (0) \end{array} \qquad \begin{array}{r} 01001 \quad (9) \\ + \underline{10111} \quad (-9) \\ \hline 00000 \quad (0) \end{array}$$

CIT 593

2-11

## Problem

### Signed-magnitude and 1’s complement

- Two representations of zero (+0 and -0)
- Addition is complex

➤ add two sign-magnitude numbers?

– e.g., try  $-12 + 13$

$$\begin{array}{r} 10011 \quad (-12) \\ + \underline{01101} \quad (13) \\ \hline 1\ 00000 \quad (0) \end{array}$$

➤ Need to add add back the carry into least significant bit (LSB)

➤ Which will result in the correct result,  $-12+13 = 1$

- Read handout on Chp 3 section 3.2.3 from Principles of Computer Architecture by Murdocca (gives more detail explanation compared to Yale & Patel txt).

CIT 593

2-10

## Two’s Complement (cont.)

### If number is positive or zero

- Normal binary representation, zeroes in upper bit(s)

### If number is negative

- Start with positive number
- Flip every bit (i.e., take the one’s complement)
- Then add one

$$\begin{array}{r} \curvearrowright 00101 \quad (5) \\ \curvearrowleft 11010 \quad (1's\ comp) \\ + \underline{\quad 1} \\ \hline 11011 \quad (-5) \end{array} \qquad \begin{array}{r} \curvearrowright 01001 \quad (9) \\ \curvearrowleft 10110 \quad (1's\ comp) \\ + \underline{\quad 1} \\ \hline 10111 \quad (-9) \end{array}$$

CIT 593

2-12

## Two's Complement Signed Integers

MS bit is sign bit: it has weight  $-2^{n-1}$

Range of an n-bit number:  $-2^{n-1}$  through  $2^{n-1} - 1$

- Note: most negative number ( $-2^{n-1}$ ) has no positive counterpart

$-2^3$	$2^2$	$2^1$	$2^0$		$-2^3$	$2^2$	$2^1$	$2^0$	
0	0	0	0	0	1	0	0	0	-8
0	0	0	1	1	1	0	0	1	-7
0	0	1	0	2	1	0	1	0	-6
0	0	1	1	3	1	0	1	1	-5
0	1	0	0	4	1	1	0	0	-4
0	1	0	1	5	1	1	0	1	-3
0	1	1	0	6	1	1	1	0	-2
0	1	1	1	7	1	1	1	1	-1

CIT 593

2-13

## More Examples

$$\begin{aligned} X &= 00100111_{\text{two}} \\ &= 2^5 + 2^2 + 2^1 + 2^0 = 32 + 4 + 2 + 1 \\ &= 39_{\text{ten}} \end{aligned}$$

$$\begin{aligned} X &= 11100110_{\text{two}} \\ -X &= 00011010 \\ &= 2^4 + 2^3 + 2^1 = 16 + 8 + 2 \\ &= 26_{\text{ten}} \\ X &= -26_{\text{ten}} \end{aligned}$$

n	$2^n$
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Assuming 8-bit 2's complement numbers.

CIT 593

2-15

## Converting Binary (2's C) to Decimal

- If leading bit is one, take two's complement to get a positive number
- Add powers of 2 that have "1" in the corresponding bit positions
- If original number was negative, add a minus sign

$$\begin{aligned} X &= 01101000_{\text{two}} \\ &= 2^6 + 2^5 + 2^3 = 64 + 32 + 8 \\ &= 104_{\text{ten}} \end{aligned}$$

n	$2^n$
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Assuming 8-bit 2's complement numbers.

CIT 593

2-14

## Converting Decimal to Binary (2's C)

First Method: *Division*

- Change to positive decimal number
- Divide by two – remainder is least significant bit
- Keep dividing by two until answer is zero, recording remainders from right to left
- Append a zero as the MS bit; if original number negative, take two's complement

$$\begin{aligned} X &= 104_{\text{ten}} & 104/2 &= 52 \text{ r}0 & \text{bit } 0 \\ & & 52/2 &= 26 \text{ r}0 & \text{bit } 1 \\ & & 26/2 &= 13 \text{ r}0 & \text{bit } 2 \\ & & 13/2 &= 6 \text{ r}1 & \text{bit } 3 \\ & & 6/2 &= 3 \text{ r}0 & \text{bit } 4 \\ & & 3/2 &= 1 \text{ r}1 & \text{bit } 5 \\ & & 1/2 &= 0 \text{ r}1 & \text{bit } 6 \end{aligned}$$

$$X = 01101000_{\text{two}}$$

CIT 593

2-16

## Converting Decimal to Binary (2's C)

**Second Method:** *Subtract Powers of Two*

1. Change to positive decimal number
2. Subtract largest power of two less than or equal to number
3. Put a one in the corresponding bit position
4. Keep subtracting until result is zero
5. Append a zero as MS bit; if original was negative, take two's complement

n	2 <sup>n</sup>
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

$X = 104_{\text{ten}}$	$104 - 64 = 40$	<i>bit 6</i>
	$40 - 32 = 8$	<i>bit 5</i>
	$8 - 8 = 0$	<i>bit 3</i>
$X = 01101000_{\text{two}}$		

CIT 593

2-17

## Operations: Arithmetic and Logical

### Recall

- A data type includes *representation* and *operations*

### Operations for signed integers

- Addition
- Subtraction
- Sign Extension

### Logical operations are also useful

- AND
- OR
- NOT

### And. . .

- Overflow conditions for addition

CIT 593

2-19

## Sign Extension

### To get correct results

- Must represent numbers with same number of bits

### What if we just pad with zeroes on the left?

<u>4-bit</u>	<u>8-bit</u>
0100 (4)	00000100 (still 4)
1100 (-4)	00001100 (12, not -4)

### Now, let's replicate the MSB (the sign bit)

<u>4-bit</u>	<u>8-bit</u>
0100 (4)	00000100 (still 4)
1100 (-4)	11111100 (still -4)

CIT 593

2-18

## Addition

### 2's comp. addition is just binary addition

- Assume all integers have the same number of bits
- Ignore carry out
- For now, assume that sum fits in n-bit 2's comp. representation
  - Assuming 8-bit 2's complement numbers

01101000 (104)	11110110 (-10)
+ 11110000 (-16)	+ 11110111 (-9)
1 01011000 (88)	1 11101101 (-19)

carry(discard) →

CIT 593

2-20

## Subtraction

### Negate 2nd operand and add

- Assume all integers have the same number of bits
- Ignore carry out
- For now, assume that difference fits in  $n$ -bit 2's comp. representation

$$\begin{array}{r}
 01101000 \text{ (104)} \quad 11110110 \text{ (-10)} \\
 - \underline{00010000 \text{ (16)}} \quad - \underline{11110111 \text{ (-9)}} \\
 \hline
 01101000 \text{ (104)} \quad 11110110 \text{ (-10)} \\
 + \underline{11110000 \text{ (-16)}} \quad + \underline{00001001 \text{ (9)}} \\
 \hline
 01011000 \text{ (88)} \quad 11111111 \text{ (-1)}
 \end{array}$$

Assuming 8-bit 2's complement numbers.

CIT 593

2-21

## Logical Operations

### Operations on logical TRUE or FALSE

- Two states: TRUE=1, FALSE=0

A	B	A AND B	A B	A OR B	A	NOT A
0	0	0	0 0	0	0	1
0	1	0	0 1	1	1	0
1	0	0	1 0	1		
1	1	1	1 1	1		

### View $n$ -bit number as a collection of $n$ logical values

- Operation applied to each bit independently

CIT 593

2-23

## Overflow

Overflow is said to occur if result is too large to fit in the number of bits used in the representation.

- Sum cannot be represented as  $n$ -bit 2's comp number

$$\begin{array}{r}
 01000 \text{ (8)} \quad 11000 \text{ (-8)} \\
 + \underline{01001 \text{ (9)}} \quad + \underline{10111 \text{ (-9)}} \\
 \hline
 10001 \text{ (-15)} \quad 01111 \text{ (+15)}
 \end{array}$$

### We have overflow if

- Signs of both numbers are the same, and Sign of sum is different
- If Positive number is subtracted from a Negative number, result is positive and vice versa

### Another test (easy for hardware)

- Carry into MSB does not equal carry out

CIT 593

2-22

## Examples of Logical Operations

### AND

- Useful for clearing bits
  - AND with zero = 0
  - AND with one = no change

$$\begin{array}{r}
 11000101 \\
 \text{AND } \underline{00001111} \\
 \hline
 00000101
 \end{array}$$

### OR

- Useful for setting bits
  - OR with zero = no change
  - OR with one = 1

$$\begin{array}{r}
 11000101 \\
 \text{OR } \underline{00001111} \\
 \hline
 11001111
 \end{array}$$

### NOT

- Unary operation -- one argument
- Flips every bit

$$\begin{array}{r}
 \text{NOT } \underline{11000101} \\
 \hline
 00111010
 \end{array}$$

CIT 593

2-24

## Hexadecimal Notation

It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead

- Fewer digits: four bits per hex digit
- Less error prone: easy to corrupt long string of 1's and 0's

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

CIT 593

2-25

## Fractions: Fixed-Point

How can we represent fractions?

- Use a “binary point” to separate positive from negative powers of two (just like “decimal point”)
- 2's comp addition and subtraction still work

➤ If binary points are aligned

$$\begin{array}{r}
 \phantom{000}2^{-1} = 0.5 \\
 \phantom{000}2^{-2} = 0.25 \\
 \phantom{000}2^{-3} = 0.125 \\
 \hline
 00101000.101 \text{ (40.625)} \\
 + \underline{11111110.110} \text{ (-1.25)} \\
 \hline
 00100111.011 \text{ (39.375)}
 \end{array}$$

No new operations -- same as integer arithmetic

CIT 593

2-27

## Hexadecimal Conversions

Binary to Hex

- Every group of four bits is a hex digit
- Start grouping from right-hand side

0011 1010 1000 1111 0100 1101 0111  
 ↓ ↓ ↓ ↓ ↓ ↓ ↓  
 3 A 8 F 4 D 7

Hex to Decimal

$$1AC = 1 \times 16^2 + 10 \times 16^1 + 12 \times 16^0 = 428$$

This is not a new machine representation, just a convenient way to write the number.

CIT 593

2-26

## Very Large and Very Small: Floating-Point

Problem

- Large values:  $6.023 \times 10^{23}$  -- requires 79 bits
- Small values:  $6.626 \times 10^{-34}$  -- requires >110 bits

Use equivalent of “scientific notation”:  $F \times 2^E$

Need to represent  $F$  (fraction),  $E$  (exponent), and sign

IEEE 754 Floating-Point Standard (32-bits):



$$N = -1^S \times 1.\text{fraction} \times 2^{\text{exponent} - 127}, \quad 1 \leq \text{exponent} \leq 254$$

$$N = -1^S \times 0.\text{fraction} \times 2^{-126}, \quad \text{exponent} = 0$$

CIT 593

2-28

## Floating Point Example

### Single-precision IEEE floating point number

10111111010000000000000000000000

↑            ↑                            ↑  
*sign*    *exponent*                            *fraction*

- Sign is 1: number is negative
- Exponent field is 01111110 = 126 (decimal)
- Fraction is 0.10000000000... = 1/10 = 0.5 (decimal)

Value =  $-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75$

CIT 593

2-29

## Other Data Types

### Text strings

- Sequence of characters, terminated with NULL (0)
- Typically, no hardware support

### Image

- Array of pixels
  - Monochrome: one bit (0/1 = black/white)
  - Color: red, green, blue (RGB) components (e.g., 8 bits each)
  - Other properties: transparency
  - Typically no hardware support

### Sound

- Sequence of fixed-point numbers

CIT 593

2-31

## Text: ASCII Characters

### ASCII: Maps 128 characters to 7-bit code.

- Both printable and non-printable (ESC, DEL, ...) characters

00 nul	10 dle	20 sp	30 0	40 @	50 P	60 `	70 p
01 soh	11 dcl	21 !	31 1	41 A	51 Q	61 a	71 q
02 stx	12 dc2	22 "	32 2	42 B	52 R	62 b	72 r
03 etx	13 dc3	23 #	33 3	43 C	53 S	63 c	73 s
04 eot	14 dc4	24 \$	34 4	44 D	54 T	64 d	74 t
05 eng	15 nak	25 %	35 5	45 E	55 U	65 e	75 u
06 ack	16 syn	26 &	36 6	46 F	56 V	66 f	76 v
07 bel	17 etb	27 '	37 7	47 G	57 W	67 g	77 w
08 bs	18 can	28 (	38 8	48 H	58 X	68 h	78 x
09 ht	19 em	29 )	39 9	49 I	59 Y	69 i	79 y
0a nl	1a sub	2a *	3a :	4a J	5a Z	6a j	7a z
0b vt	1b esc	2b +	3b ;	4b K	5b [	6b k	7b {
0c np	1c fs	2c ,	3c <	4c L	5c \	6c l	7c
0d cr	1d gs	2d -	3d =	4d M	5d ]	6d m	7d }
0e so	1e rs	2e .	3e >	4e N	5e ^	6e n	7e ~
0f si	1f us	2f /	3f ?	4f O	5f _	6f o	7f del

CIT 593

2-30

## LC-3 Data Types

### For LC-3, there is only one supported data type

- 16-bit 2's complement signed integer
- Operations: ADD, AND, NOT (and sometimes MUL)

### Other data types?

- Supported by interpreting 16-bit values as logical, text, fixed-point, etc., in the software that we write

CIT 593

2-32

## Next Time

### Lecture

- Brief overview of chp 3
- Von Nuemann Model and LC3 Instruction Processing Chp 4

### Reading

- Chp 2 – Yale & Patel

### Homework

- Assigned on blackboard due Thursday 9/21