

Chapter 2

Bits, Data Types, and Operations

Based on slides © McGraw-Hill
Additional material © 2004/2005/2006 Lewis/Martin

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^n** possible states

Aside: why binary?

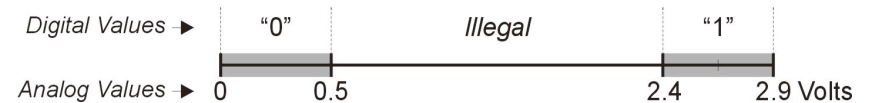
How do we represent data in a computer?

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

- Operates by controlling the flow of electrons

Easy to recognize two conditions:

1. Presence of a voltage – we’ll call this state “1”
2. 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

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, ...
- **Sound**
- **Logical** – true, false
- **Instructions**
- ...

Data type:

- *Representation* and *operations* within the computer

We’ll start with numbers...

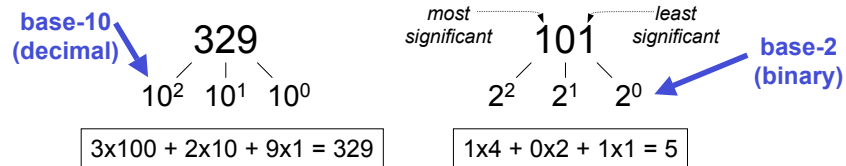
Unsigned Integers

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



CSE 240

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

CSE 240

2-6

N = 4		Number Represented
Binary	Unsigned	
0000	0	
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

From GATech's CS2110,
Kishore Ramachandran
Used with permission

Unsigned Binary Arithmetic

Base-2 addition – just like base-10!

- Add from right to left, propagating carry

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

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

Subtraction, multiplication, division,...

CSE 240

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, most significant bit indicates sign:
➤ 0=positive, 1=negative

CSE 240

2-9

N = 4		Number Represented	
Binary	Unsigned	Signed Mag	
0000	0	0	
0001	1	1	
0010	2	2	
0011	3	3	
0100	4	4	
0101	5	5	
0110	6	6	
0111	7	7	
1000	8	-0	
1001	9	-1	
1010	10	-2	
1011	11	-3	
1100	12	-4	
1101	13	-5	
1110	14	-6	
1111	15	-7	

From GATech's CS2110,
Kishore Ramachandran
Used with permission

N = 4		Number Represented		
Binary	Unsigned	Signed Mag	1's Comp	
0000	0	0	0	
0001	1	1	1	
0010	2	2	2	
0011	3	3	3	
0100	4	4	4	
0101	5	5	5	
0110	6	6	6	
0111	7	7	7	
1000	8	-0	-7	
1001	9	-1	-6	
1010	10	-2	-5	
1011	11	-3	-4	
1100	12	-4	-3	
1101	13	-5	-2	
1110	14	-6	-1	
1111	15	-7	-0	

From GATech's CS2110,
Kishore Ramachandran
Used with permission

Problem

Signed-magnitude and 1's complement

- Two representations of zero (+0 and -0)
- Arithmetic circuits are complex
➤ How do we add two sign-magnitude numbers?

– e.g., try 2 + (-3)

$$\begin{array}{r}
 00010 \quad (2) \\
 + \underline{10011} \quad (-3) \\
 \hline
 10001 \quad (-1)
 \end{array}$$



- How do we add to one's complement numbers?

– e.g., try 4 + (-3)

$$\begin{array}{r}
 00100 \quad (4) \\
 + \underline{11100} \quad (-3) \\
 \hline
 00001 \quad (1)
 \end{array}$$



CSE 240

2-12

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

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

CSE 240

2-13

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

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

CSE 240

2-14

Two's Complement Shortcut

To take the two's complement of a number:

- Copy bits from right to left until (and including) the first "1"
- Flip remaining bits to the left

$$\begin{array}{r} \curvearrowright 011010000 \\ \curvearrowright \underline{100101111} \quad (1's \text{ comp}) \\ + \quad \quad \quad \underline{1} \\ \hline 100110000 \end{array} \qquad \begin{array}{r} 011010000 \\ \text{(flip)} \downarrow \quad \downarrow \text{(copy)} \\ 100110000 \end{array}$$

CSE 240

2-15

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

CSE 240

2-16

N = 4	Number Represented			
	Binary	Unsigned	Signed Mag	1's Comp
0000	0	0	0	0
0001	1	1	1	1
0010	2	2	2	2
0011	3	3	3	3
0100	4	4	4	4
0101	5	5	5	5
0110	6	6	6	6
0111	7	7	7	7
1000	8	-0	-7	-8
1001	9	-1	-6	-7
1010	10	-2	-5	-6
1011	11	-3	-4	-5
1100	12	-4	-3	-4
1101	13	-5	-2	-3
1110	14	-6	-1	-2
1111	15	-7	-0	-1

Two's complement is used by all modern computers

From GATech's CS2110, Kishore Ramachandran
Used with permission

Converting Binary (2's C) to Decimal

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

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

Assuming 8-bit 2's complement numbers.

CSE 240

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

2-18

More Examples

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

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

Assuming 8-bit 2's complement numbers.

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

CSE 240

2-19

Converting Decimal to Binary (2's C)

First Method: Division

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

$$\begin{array}{l}
 X = 104_{\text{ten}} \\
 104/2 = 52 \text{ r}0 \quad \text{bit } 0 \\
 52/2 = 26 \text{ r}0 \quad \text{bit } 1 \\
 26/2 = 13 \text{ r}0 \quad \text{bit } 2 \\
 13/2 = 6 \text{ r}1 \quad \text{bit } 3 \\
 6/2 = 3 \text{ r}0 \quad \text{bit } 4 \\
 3/2 = 1 \text{ r}1 \quad \text{bit } 5 \\
 1/2 = 0 \text{ r}1 \quad \text{bit } 6 \\
 \\
 X = 01101000_{\text{two}}
 \end{array}$$

CSE 240

2-20

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^n
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>
$X = 01101000_{\text{two}}$		

CSE 240

2-21

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

CSE 240

2-22

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

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

Assuming 8-bit 2's complement numbers.

CSE 240

2-23

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

01101000 (104)	11110110 (-10)
$- 00010000$ (16)	$- 11110111$ (-9)
01101000 (104)	11110110 (-10)
$+ 11110000$ (-16)	$+ 00001001$ (9)
01011000 (88)	11111111 (-1)

Assuming 8-bit 2's complement numbers.

CSE 240

2-24

Sign Extension

To add

- 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)

No, 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)

CSE 240

2-25

Overflow

What if operands are too big?

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

01000 (8)	11000 (-8)
+ 01001 (9)	+ 10111 (-9)
10001 (-15)	01111 (+15)

We have overflow if

- Signs of both operands are the same, and
- Sign of sum is different

Another test (easy for hardware)

- Carry into most significant bit does not equal carry out

CSE 240

2-26

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

CSE 240

2-27

Examples of Logical Operations

AND

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

AND $\begin{array}{l} 11000101 \\ 00001111 \\ \hline 00000101 \end{array}$

OR

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

OR $\begin{array}{l} 11000101 \\ 00001111 \\ \hline 11001111 \end{array}$

NOT

- Unary operation -- one argument
- Flips every bit

NOT $\begin{array}{l} 11000101 \\ \hline 00111010 \end{array}$

CSE 240

2-28

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 misread 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

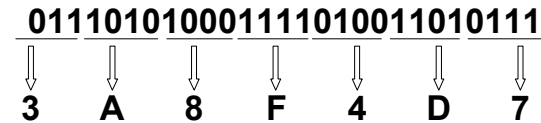
CSE 240

2-29

Converting from Binary to Hexadecimal

Every group of four bits is a hex digit

- Start grouping from right-hand side



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

CSE 240

2-30

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

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

No new operations -- same as integer arithmetic

CSE 240

2-31

Very Large and Very Small: Floating-Point

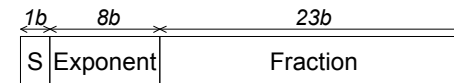
Problem

- Large values: 6.023×10^{23} -- requires 79 bits
- Small values: 6.626×10^{-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):



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

CSE 240

2-32

Floating Point Example

Single-precision IEEE floating point number - 32 bits



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

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

Double-precision IEEE floating point - 64bits

- 11-bit exponent field, 52-bit fraction field

Floating-Point Operations

Will regular 2's complement arithmetic work for Floating Point numbers?

(Hint: In decimal, how do we compute $3.07 \times 10^{12} + 9.11 \times 10^8$?)

Floating Point Specials



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

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

If exponent bits are 0, "denormalized" numbers

- Gradual underflow (also used for representing zero)

Other specials

- Two zeros (-0, 0)
- Two Infinities (-infinity, infinity)
- Not a number (negative and positive)
 - When does this occur?

Lots of corner cases (difficult to implement correctly)

- Example: rounding modes

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	dc1	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	enq	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

Interesting Properties of ASCII Code

What is relationship between a decimal digit ('0', '1', ...) and its ASCII code?

What is the difference between an upper-case letter ('A', 'B', ...) and its lower-case equivalent ('a', 'b', ...)?

Given two ASCII characters, how do we tell which comes first in alphabetical order?

Are 128 characters enough?
(<http://www.unicode.org/>)

No new operations -- integer arithmetic and logic.

CSE 240

2-37

LC-3 Data Types

Some data types are supported directly by the instruction set architecture

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

CSE 240

2-39

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
- Hardware support
 - Typically none, in general-purpose processors
 - MMX: multiple 8-bit operations on 32-bit word

Sound

- Sequence of fixed-point numbers

CSE 240

2-38

Next Time

Lecture

- Digital logic structures: transistors and gates

Reading

- Chapter 3-3.2

Quiz

- Online

Upcoming

- HW1 due this Friday!

CSE 240

2-40