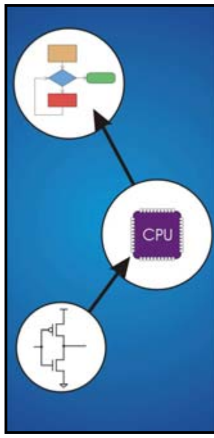
Introduction to Computer Engineering

ECE/CS 252, Fall 2010
Prof. Mikko Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin – Madison



Chapter 2
Bits, Data Types, and Operations - Part 1

Slides based on set prepared by
Gregory T. Byrd, North Carolina State University

How do we represent data in a computer?

At the lowest level, a computer is an electronic machine.

- **works 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"**

Could base state on *value* of voltage, but control and detection circuits more complex.

- **compare turning on a light switch to measuring or regulating voltage**

We'll see examples of these circuits in the next chapter.

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 is the *binary digit*, or *bit*.

Values with more than two states 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.***

What kinds of data do we need to represent?

- **Numbers – signed, unsigned, integers, 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**

329
10^2 10^1 10^0
3x100 + 2x10 + 9x1 = 329

most significant 101 *least significant*
2^2 2^1 2^0
1x4 + 0x2 + 1x1 = 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	
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

Adding Binary Numbers

Just like decimal arithmetic

Arithmetic tables:

- **1+1 = 2**
- **1+2 = 3**
- **…**
- **9+9 = 18**
- ***If sum > 9 we have to carry***

Binary table is much smaller

- ***If sum > 1 we have to carry***

A	B	C	A+B+C (carry|sum)
0	0	0	00
0	1	0	01
1	0	0	01
1	1	0	10
0	0	1	01
0	1	1	10
1	0	1	10
1	1	1	11

Unsigned Binary Arithmetic

Base-2 addition – just like base-10!

- **add from right to left, propagating carry**

```
  10010        carry
+  1001        10010         1111
  11011      +  1011       +    1
               11101        10000

               10111
             +   111
               11110
```

Subtraction, multiplication, division,…

Converting Binary to Decimal

Add powers of 2 that have "1" in the corresponding bit positions.

$X = 01101000_{two}$
$= 2^6+2^5+2^3 = 64+32+8$
$= 104_{ten}$

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

Converting Decimal to Binary

First Method: *Subtract Powers of Two*

1. **Subtract largest power of two less than or equal to number.**
2. **Put a one in the corresponding bit position.**
3. **Keep subtracting until result is zero.**

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

$X = 104_{ten}$

104 - 64 =	40	*bit 6*
40 - 32 =	8	*bit 5*
8 - 8 =	0	*bit 3*

$X = 01101000_{two}$

Converting Decimal to Binary

Second Method: *Division*

1. **Divide by two – remainder is least significant bit.**
2. **Keep dividing by two until answer is zero, writing remainders from right to left.**

$X = 104_{ten}$

104/2 =	52 r0	*bit 0*
52/2 =	26 r0	*bit 1*
26/2 =	13 r0	*bit 2*
13/2 =	6 r1	*bit 3*
6/2 =	3 r0	*bit 4*
3/2 =	1 r1	*bit 5*
1/2 =	0 r1	*bit 6*

$X = 01101000_{two}$

Operations: Arithmetic and Logical

Recall:
a data type includes *representation* and *operations*.
We have a good representation for unsigned integers, and one operation: Addition
- Will look at other operations later

Logical operations are also useful:
- AND
- OR
- NOT

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Logical Operations

Operations on logical TRUE or FALSE
- two states -- takes one bit to represent: TRUE=1, FALSE=0

View *n*-bit number as a collection of *n* logical values
- operation applied to each bit independently

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Examples of Logical Operations

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

```
     11000101
AND  00001111
     00000101
```

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

```
     11000101
OR   00001111
     11001111
```

NOT
- unary operation -- one argument
- flips every bit

```
NOT  11000101
     00111010
```


Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Converting from Binary to Hexadecimal

Every four bits is a hex digit.
- start grouping from right-hand side

011101010001111010011010111

3 A 8 F 4 D 7

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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Text: ASCII Characters

ASCII: Maps 128 characters to 7-bit code.
- both printable and non-printable (BEL, 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.

Other Data Types

Text strings
- **sequence of characters, terminated with NULL (0)**
- **typically, no hardware support**

Image
- **array of pixels**
  - **monochrome: one bit (1/0 = 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**

Summary

Binary digital system
Data type: *representation and operations*
Unsigned integers
- **Weighted positional notation**
- **Addition**
- **Conversion from binary to decimal**
- **Converstion from decimal to binary (2 methods)**

Logical operations: AND/OR/NOT
Hexadecimal notation
ASCII representation for characters/text
Other data types