Binary Arithmetic

CS 64: Computer Organization and Design Logic
Lecture #2
Winter 2019

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB

Administrative Stuff

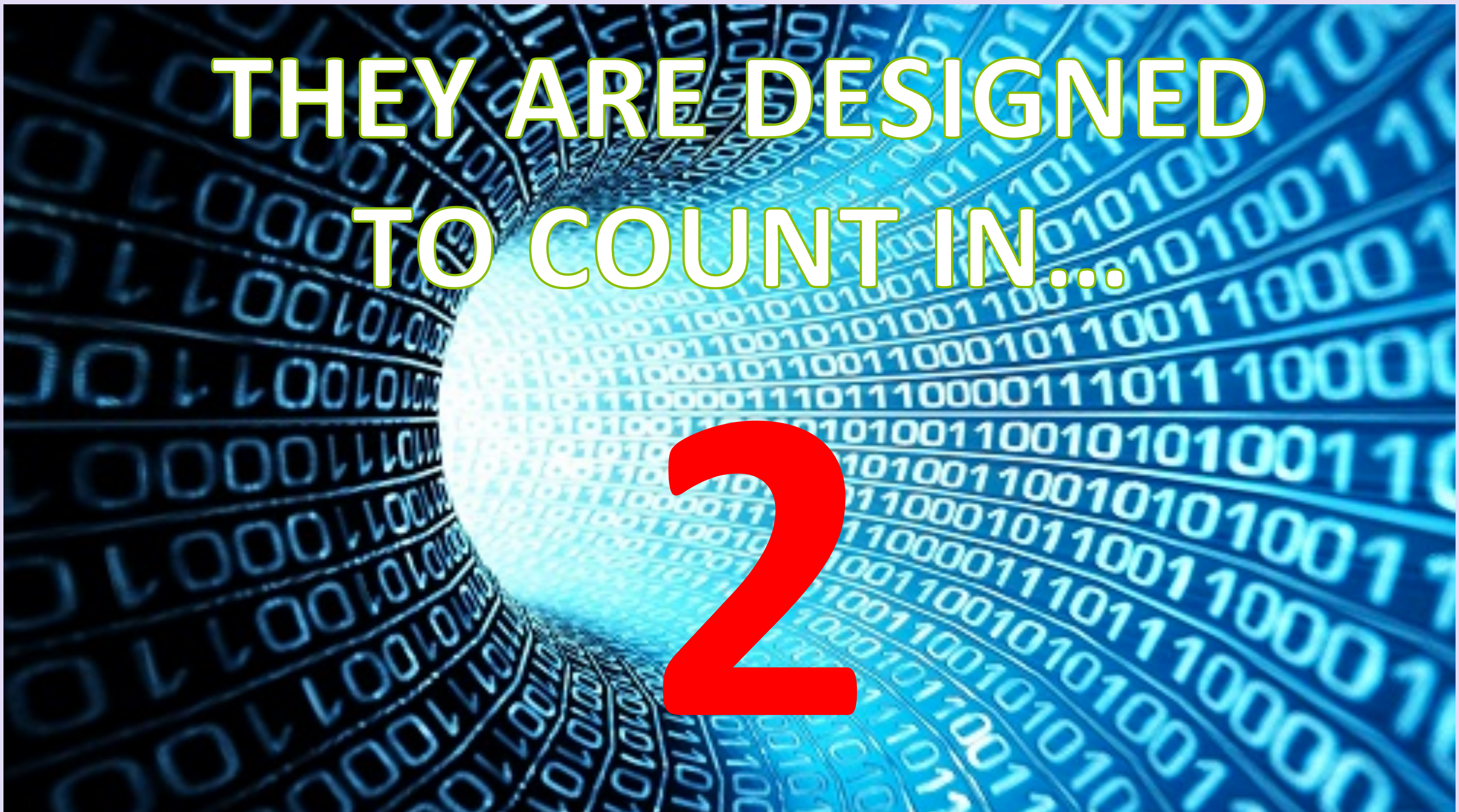
- The class is still full…

- Did you check out the syllabus?
- Did you check out the class website?
- Did you check out Piazza (and get access to it)?
- Did you go to lab today?
- Do you understand how you will be submitting your assignments?

Lecture Outline

- Review of positional notation, binary logic
- Bitwise operations
- Bit shift operations
- Two's complement
- Addition and subtraction in binary

COMPUTERS ARE DIGITAL MACHINES



THEY ARE DESIGNED TO COUNT IN…

2

Counting Numbers in Different Bases

- We "normally" count in 10s
  - Base 10: **decimal** numbers
  - We use 10 numerical symbols in Base 10: "0" thru "9"

- Computers count in 2s
  - Base 2: **binary** numbers
  - We use 2 numerical symbols in Base 2: "0" and "1"

- Represented with **1 bit** ($2^1 = 2$)

Counting Numbers in Different Bases

Other convenient bases in computer architecture:

- Base 8: **octal** numbers
  - Number symbols are 0 thru 7
  - Represented with **3 bits** ($2^3 = 8$)

- Base 16: **hexadecimal** numbers
  - Number symbols are 0 thru F:

    A = 10, B = 11, C = 12, D = 13, E = 14, F = 15
  - Represented with **4 bits** ($2^4 = 16$)

- **Why are 4 bit representations convenient???**

What's in a Number?

642

What *is* that???

Well, what NUMERICAL BASE are you expressing it in?

Positional Notation of Decimal Numbers

642 in base 10 (decimal) can be described in "*positional notation*" as:

$$
\begin{aligned}
6 \times 10^2 &= 6 \times 100 &&= 600 \\
+\ 4 \times 10^1 &= 4 \times 10 &&= 40 \\
+\ 2 \times 10^0 &= 2 \times 1 &&= 2 \quad = \text{ 642 in base 10}
\end{aligned}
$$

6	4	2
100	10	1

$$642_{\text{(base 10)}} = 600 + 40 + 2$$

Numerical Bases and Their Symbols

- How many “symbols” or “digits” do we use in Decimal (Base 10)?
- Base 2 (Binary)?
- Base 16 (Hexadecimal)?
- Base N?

Positional Notation

This is how you convert any base number into decimal!

Each digit gets multiplied by B^N

Where:

B = the base

N = the position of the digit

*Example: given the number **613** in **base 7**:*

Number in decimal = $\mathbf{6} \times 7^2 + \mathbf{1} \times 7^1 + \mathbf{3} \times 7^0 = 304$

Positional Notation in Binary

11101 in base 2 *positional notation* is:

$$
\begin{aligned}
& 1 \times 2^4 = 1 \times 16 = 16 \\
+ & 1 \times 2^3 = 1 \times 8 \;\; = 8 \\
+ & 1 \times 2^2 = 1 \times 4 \;\; = 4 \\
+ & 0 \times 2^1 = 1 \times 2 \;\; = 0 \\
+ & 1 \times 2^0 = 1 \times 1 \;\; = 1
\end{aligned}
$$

So, **11101** in base 2 is 16 + 8 + 4+ 0 + 1 = **29** in base 10

Converting Binary to Octal and Hexadecimal

(or any base that's a power of 2)

NOTE THE FOLLOWING:

- Binary is 1 bit
- Octal is 3 bits
- Hexadecimal is 4 bits

- Use the "group the bits" technique
  - Always start from the *least significant digit*
  - Group every 3 bits together for bin → oct
  - Group every 4 bits together for bin → hex

Converting Binary to Octal and Hexadecimal

- Take the example: ***10100110***

...to octal:

$\boxed{10}\boxed{100}\boxed{110}$

2 ***4*** ***6***

246 in octal

...to hexadecimal:

$\boxed{1010}\boxed{0110}$

10 ***6***

A6 in hexadecimal

Converting Decimal to Other Bases

Algorithm for converting number in base 10 to other bases

While (the quotient is not zero)

1. Divide the decimal number by the new base
2. Make the remainder the next digit to the left in the answer
3. Replace the original decimal number with the quotient
4. Repeat until your quotient is zero

Example: What is 98 (base 10) in base 8?

98 / 8 = 12 R 2

12 / 8 = 1 R 4

1 / 8 = 0 R 1

1 4 2

In-Class Exercise:
Converting Decimal into Binary & Hex

<u>Convert 54 (base 10) into binary and hex:</u>

- 54 / 2 = 27 R 0
- 27 / 2 = 13 R 1
- 13 / 2 = 6 R 1
- 6 / 2 = 3 R 0
- 3 / 2 = 1 R 1
- 1 / 2 = 0 R 1

<u>Sanity check:</u>
110110
= 2 + 4 + 16 + 32
= 54

54 (decimal) = 110110 (binary)
= 36 (hex)

Convenient Table…

HEXADECIMAL	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

HEXADECIMAL (Decimal)	BINARY
A (10)	1010
B (11)	1011
C (12)	1100
D (13)	1101
E (14)	1110
F (15)	1111

Always Helpful to Know…

N	2^N
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024 = 1 kilobits

N	2^N
11	2048 = 2 kb
12	4 kb
13	8 kb
14	16 kb
15	32 kb
16	64 kb
17	128 kb
18	256 kb
19	512 kb
20	1024 kb = 1 megabits

N	2^N
21	2 Mb
22	4 Mb
23	8 Mb
24	16 Mb
25	32 Mb
26	64 Mb
27	128 Mb
28	256 Mb
29	512 Mb
30	1 Gb

Binary Logic Refresher
NOT, AND, OR

X	NOT X $\overline{X}$
0	1
1	0

X	Y	X AND Y X && Y X.Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y X || Y X + Y
0	0	0
0	1	1
1	0	1
1	1	1

Binary Logic Refresher
Exclusive-OR (XOR)

The output is "1" only if the inputs are opposite

X	Y	X XOR Y $X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise NOT

- Similar to logical NOT (!), except it works on a bit-by-bit manner
- In C/C++, it's denoted by a tilde: **~**

```
~(1001) = 0110
```

Exercises

- Sometimes hexadecimal numbers are written in the **0xhh** notation, so for example:

  The hex 3B would be written as 0x3B

- What is ~(0x04)?
  – Ans: 0xFB

- What is ~(0xE7)?
  – Ans: 0x18

Bitwise AND

- Similar to logical AND (&&), except it works on a bit-by-bit manner
- In C/C++, it's denoted by a single ampersand: &

```
(1001 & 0101) = 1 0 0 1
              & 0 1 0 1

              = 0 0 0 1
```

Exercises

- What is (0xFF) & (0x56)?
  - Ans: 0x56
- What is (0x0F) & (0x56)?
  - Ans: 0x06
- What is (0x11) & (0x56)?
  - Ans: 0x10
- Note how & can be used as a “masking” function

Bitwise OR

- Similar to logical OR (||), except it works on a bit-by-bit manner

- In C/C++, it's denoted by a single pipe: |

```
(1001 | 0101) = 1 0 0 1
              | 0 1 0 1

              = 1 1 0 1
```

Exercises

- What is (0xFF) | (0x92)?
  - Ans: 0xFF
- What is (0xAA) | (0x55)?
  - Ans: 0xFF
- What is (0xA5) | (0x92)?
  - Ans: B7

Bitwise XOR

- Works on a bit-by-bit manner
- In C/C++, it's denoted by a single carat: ^

```
(1001 ^ 0101) =  1 0 0 1
               ^  0 1 0 1

                = 1 1 0 0
```

Exercises

- What is (0xA1) ^ (0x13)?
  - Ans: 0xB2
- What is (0xFF) ^ (0x13)?
  - Ans: 0xEC
- Note how (1^b) is always ~b
  and how (0^b) is always b

YOUR TO-DOs

- Assignment #1
  - Due on Monday at 11:59 PM!!!

- Next week, we will discuss a few more **Arithmetic** topics and start exploring **Assembly Language!**
  - Do your readings!

    (again: found on the class website)

</LECTURE>