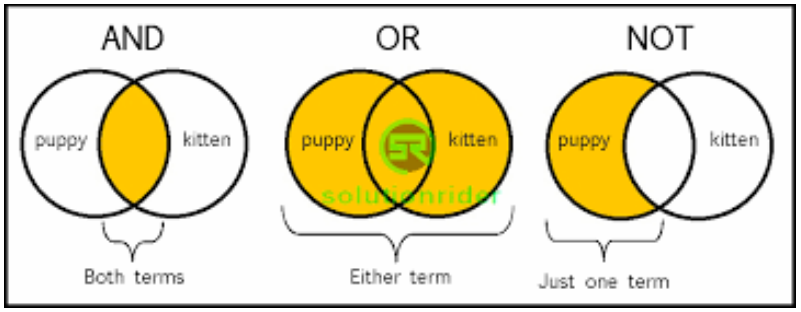

Logic Operations on Binaries
Intro to MIPS

CS 64: Computer Organization and Design Logic

Lecture #3

Fall 2019

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB

Why do CPU programmers celebrate Christmas and Halloween on the same day?

Because Oct-31 = Dec-25 !!!

Administrative Stuff

- The class is full… waitlist is closed… no adds given… ☹
- Assignment 1 is due on Wednesday
  - How was lab on Friday?
- Assignment 2 will be issued on Wednesday

Any Questions From Last Lecture?

5-Minute Pop Quiz!!!

YOU MUST SHOW YOUR WORK!!!

1. Calculate, give your answer in hexadecimal, AND identify carry out (C) and overflow (V) bit values:

(0x3E + 0xFC)

2. Convert from binary to decimal AND to hexadecimal. Use any technique(s) you like:

1001001

Answers…

1. Calculate, give your answer in hexadecimal, AND identify carry out (C) and overflow (V) bit values: **(0x3E + 0xFC)**

```
      0011 1110
+     1111 1100
= 1 0011 1010
```

There is a carry out, so C = 1
There's no overflow (why?), so V = 0

2. Convert from binary to decimal AND hexadecimal. Use any technique you like: **1001001**

= 0100 1001 = 0x49 *(collect-the-bits method)*
= 64 + 8 + 1 = 73 *(positional notation method)*

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 ⊕ 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

- Remember: hexadecimal numbers are often 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
  - Masking??! What’s being “masked”???

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: 0xB7

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 the inverse of b (**~b**) and how (**0**^b) is always just **b**

Bit Shift *Left*

- Move all the bits N positions to the left
- What do you do the positions now empty?
  - You put in N number of 0s

- Example: Shift "1001" 2 positions to the left

1001 << 2 = **1001**00

- Why is this useful as a form of *multiplication*?

Multiplication by Bit Left Shifting

- Veeeery useful in CPU (ALU) design
  - Why?
- Because you don't have to design a "multiplier" function
- You just have to design a way for the bits to shift (which is a relatively easier design)

Bit Shift *Right*

- Move all the bits N positions to the ***right***, subbing-in either N number of 0s or N 1s on the left
- Takes on two different forms

- Example: Shift "1001" 2 positions to the right

  1001 >> 2 = either **0010** or **1110**

- The information carried in the last 2 bits is *lost*.
- If Shift Left does *multiplication*, what does Shift Right do?
  - It divides, **but** it truncates the result

Two Forms of Shift Right

- Subbing-in 0s makes sense
- What about subbing-in the leftmost bit with 1?

- It's called ***"arithmetic"*** shift right:

  1100 (arithmetic) >> 1 = 1110

- It's used for *twos-complement* purposes
  - *What?*

YOUR TO-DOs

- Finish your reading for this week's classes
  - Ch. 2.2, 2.3, 2.6, 3.2, 3.3

- Finish Assignment #1
  - You have to submit it as a **PDF** using ***Gradescope***
  - Due on **Wednesday, 10/9, by 11:59:59 PM**


</LECTURE>