COMPUTER SYSTEMS AND ORGANIZATION

Bitwise Operations

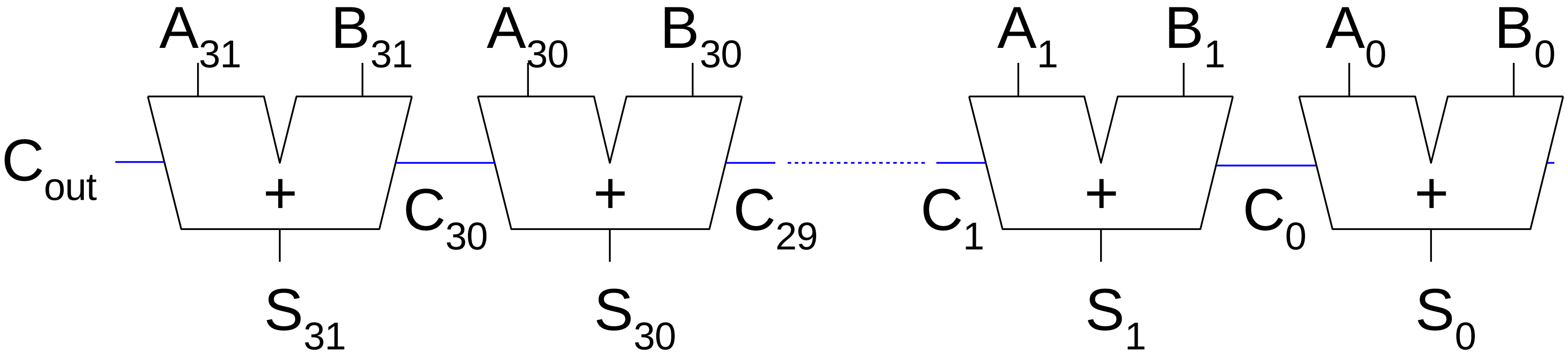
Daniel Graham

UNIVERSITY of VIRGINIA | ENGINEERING

REVIEW

RIPPLE CARRY ADDER

Next let's build a full adder



WHAT ABOUT NEGATIVE NUMBERS

How could we add 1 + (-2)

We need a way to represent negative numbers in binary.

Four Bit Space

Bits	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

How could interpret the bits as negative numbers?

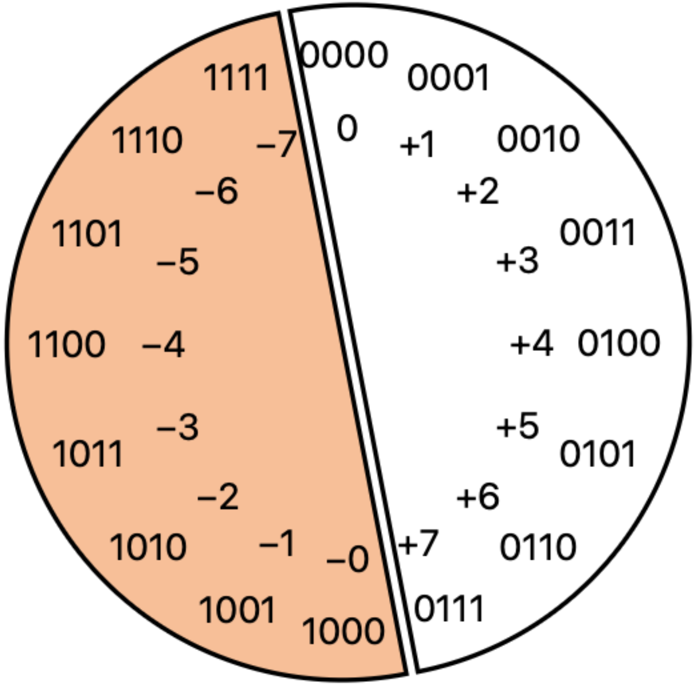
What if we decided that the top bit should indicate that the number was negative?

0010 = 2
1010 = -2

Sign Bit

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

SIGN BIT



But now we have both positive and negative zero

Will our ripple carry adder work with this representation?

```
  1 0 1 0  (-2)
+ 0 0 1 0   (2)
  1 1 0 0 (-4) <- :( doesn't work
```

Bias

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

What if we find the middle and make it represent zero? Then numbers larger than middle will be positive and numbers smaller than middle with be negative

Floor$((2^n - 1)/2) = 7$

Signed
-7
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
8

Bits
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

BIAS

From original number to BIAS

BIAS = FLOOR (MAX_NUM/2)

REPRESENTATION = ORIGINAL_NUMBER + BIAS

From BIAS to Original

BIAS = FLOOR (MAX_NUM/2)

ORIGINAL_NUMBER = REPRESENTATION - BIAS

BIAS EXAMPLE

Signed	Bits
-7	0000
-6	0001
-5	0010
-4	0011
-3	0100
-2	0101
-1	0110
0	0111
1	1000
2	1001
3	1010
4	1011
5	1100
6	1101
7	1110
8	1111

From original number to BIAS

BIAS = FLOOR (MAX_NUM/2)

REPRESENTATION = ORIGINAL_NUMBER + BIAS

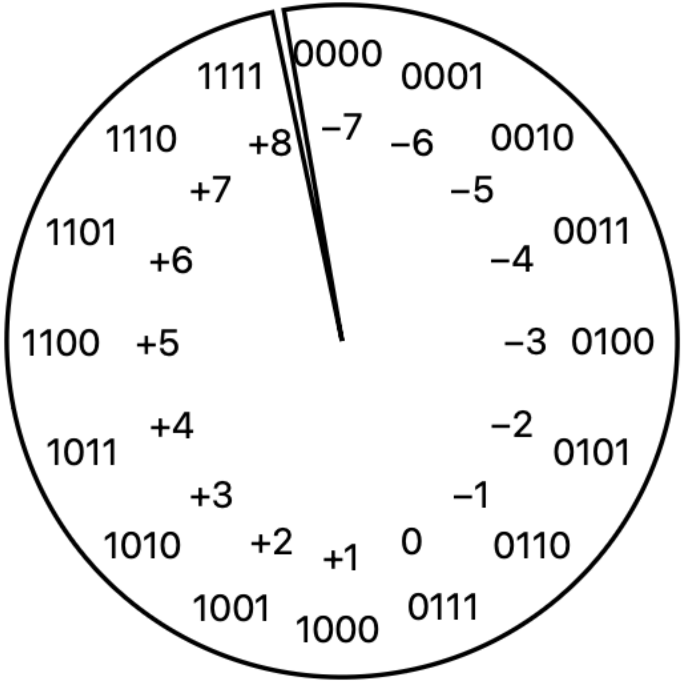
Example (original 5 to biased 5)

BIAS = FLOOR (15/2) = 7

REPRESENTATION = 5 + 7 = 12 (original) = 0b1100

0b1100 maps to 5 in our bias wheel

BIAS



From original number to BIAS

BIAS = FLOOR (MAX_NUM/2)

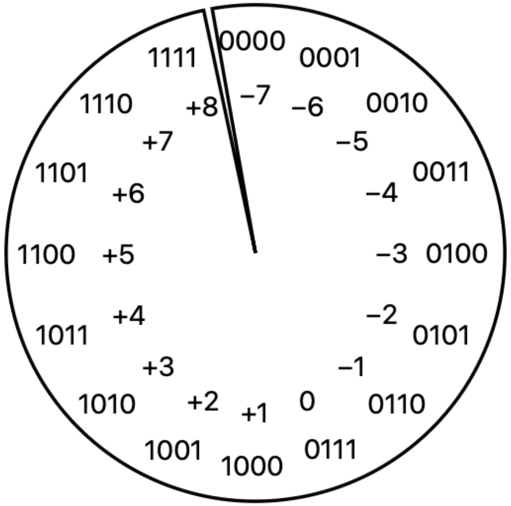
REPRESENTATION = ORIGINAL_NUMBER + BIAS

From BIAS to Original

BIAS = FLOOR (MAX_NUM/2)

ORIGINAL_NUMBER = REPRESENTATION - BIAS

DOES THE BIAS REPRESENTATION WORK WITH OUR RIPPLE CARRY ADDER?



What is the result if we add bias representation of +8 to –7. What is the result?

(Doesn't look like this works either).

University of Virginia | ENGINEERING

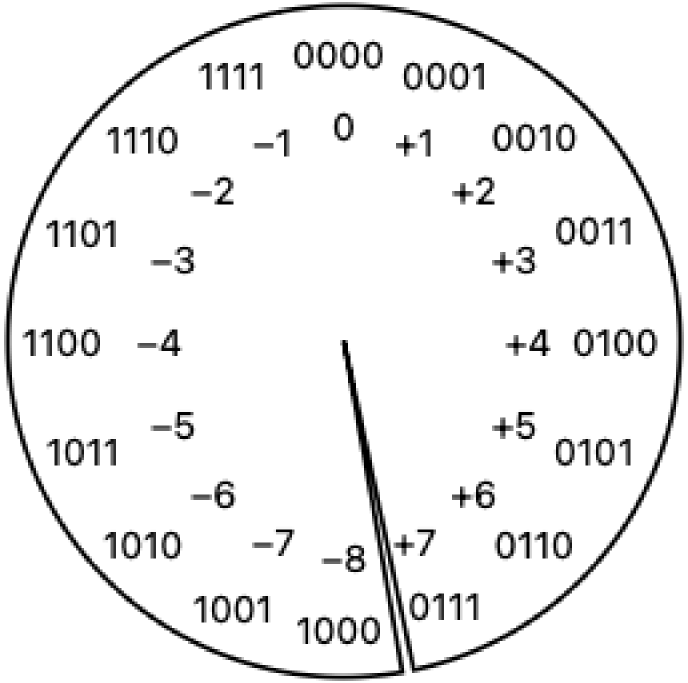
Two's Complement

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

What if the top most bit represents negative

2^3	2^2	2^1	2^0
-8	4	2	1
1	0	0	1 = -7

TWO COMPLEMENT



What if the top most bit represents negative

2^3 2^2 2^1 2^0

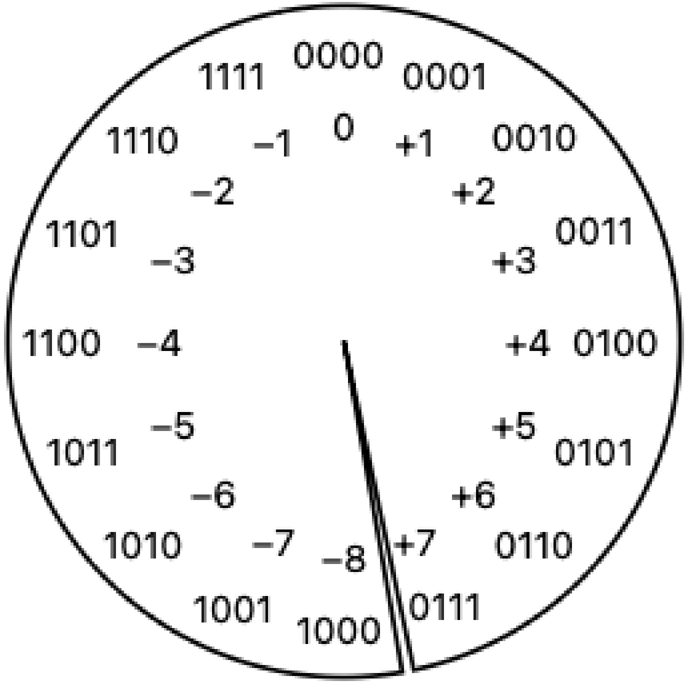
-8 4 2 1

1 0 0 1 = -7

Flip the bits and Add one

University of Virginia ENGINEERING

TWO COMPLEMENT



Flip the bits and add one trick for converting between positive and negative numbers?

What about negative to positive? Opposite process!

1. Subtract one from the binary
2. Flip the bits

Ex: -7 (0b1001) to +7 (0b0111)

1. Subtract 1 (0b1001 – 0b1 = 0b1000)
2. Flip the bits, 0b1000 --> 0b0111, which is correct!

WHY DOES FLIP THE BIT AND ONE WORK?

$X + \boxed{} = 0$

Assume $X = 1101_2$

What goes in the box?

EXAM REVIEW FALL 2018

The following assume 8-bit 2's-complement numbers. For each number, bit 0 is the low-order bit, bit 7 is the high-order bit.

Question 2 [2 pt]: (see above) Complete the following sum, showing your work (carry bits, etc)

```
  0 0 1 1 0 0 1 1
+ 0 1 1 0 0 1 1 0
-----------------
```

What is the result in base 10? Is it negative or positive? Would you get the same result in decimal if you had more bits ☺ ?

DEFINING OVERFLOW

If the sum of **two positive numbers** yields a **negative result**, the sum has **overflowed.**

If the sum of **two negative numbers** yields **a positive result**, the sum has **overflowed**.

Otherwise, the sum has not overflowed.

Overflow only exists for operations on signed numbers.

$$\begin{array}{rrl} & 0111_2 & +7 \\ + & 0001_2 & +1 \\ \hline & 1000_2 & -8 \end{array}$$

NOT OVERFLOW

If the sum of **two positive numbers** yields a **negative result**, the sum has **overflowed.**

If the sum of **two negative numbers** yields **a positive result**, the sum has **overflowed**.

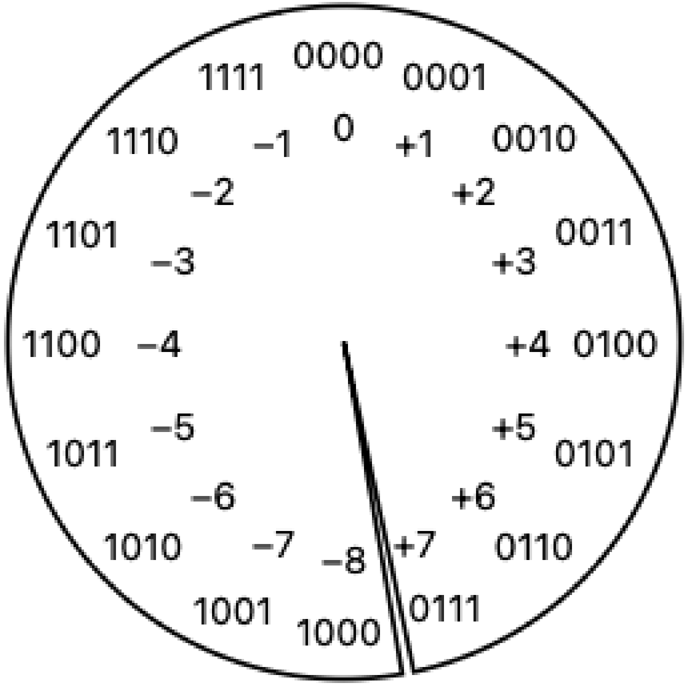
Otherwise, the sum has not overflowed.

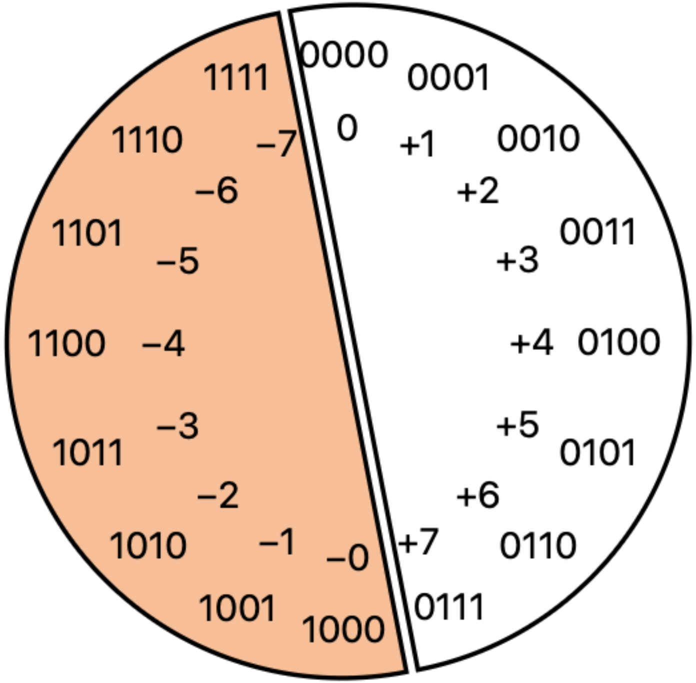
Overflow only exists for operations on signed numbers.

$$
\begin{array}{rr}
 & 1111_2 \quad -1 \\
+ & 0001_2 \quad +1 \\
\hline
 & 10000_2
\end{array}
$$

Carry Ignored, But NOT considered overflow. The answer is correctly zero

TWOS COMPLEMENT VS SIGN BIT





WRITING LONG BINARY IS NO FUN.

LET'S EXPRESS IT IN ANOTHER BASE TO MAKE IT EASIER. DEFINITELY CHOOSE SOMETHING LARGER THAN BASE 10

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

HEXADECIMAL

Convert 00101110 to hexadecimal Answer: 2E

Group them

0010 = 2

1110 = E

Final 0x2E

- Some programming languages use prefixes
  - Hex: 0x
    - 0x23AB = $23AB_{16}$
  - Binary: 0b
    - 0b1101 = 1101_2

University of Virginia | ENGINEERING

BASE 8 OCTAL

Convert 67 to octal

$67 \div 8 = 8$ remainder 3
$8 \div 8 = 1$ remainder 0
$1 \div 8 = 0$ remainder 1

103 (octal) to decimal

27 (octal) to decimal

(23) Strange right haha

Page 4: Binary

4. [6 points] Convert `219` into binary.

Answer

5. [6 points] What is `0b101100110111` in hexadecimal?

Answer

Page 4: Binary

4. [6 points] Convert `219` into binary.

128 64 32 16 8 4 2 1

$1\,1\,0\,1\,1\,0\,1\,1_2$

Answer
11011011

5. [6 points] What is `0b101100110111` in hexadecimal?

B 3 7

Answer
0xB37

NEXT LECTURE

BITWISE AND &

$$
\begin{array}{r}
1100_2 \\
\&\ 0110_2 \\
\hline
0100_2
\end{array}
$$

```python
#python example
x = 12
y = 6
z = x & y
print(z)
#prints 4
```

BITWISE OR |

$$
\begin{array}{r}
1100_2 \\
|\ \ 0110_2 \\
\hline
1110_2
\end{array}
$$

```python
#python example
x = 12
y = 6
z = x | y
print(z)
#prints 14
```

BITWISE OR XOR ^

$$
\begin{array}{r}
1100_2 \\
\wedge\ \ 0110_2 \\
\hline
1010_2
\end{array}
$$

```python
#python example
x = 12
y = 6
z = x ^ y
print(z)
#prints 10
```

BITWISE RIGHT SHIFT

$$
\begin{array}{r}
1101001_2 \\
\gg 3 \\
\hline
0001101_2
\end{array}
$$

```python
#python example
x = 105
y = x >> 3
print(y)
#prints 15
```

SIGN EXTENSIONS

$11000_2 >> 2 = 11110_2$ With Sign Extension. (The sign bit is copied)

$11000_2 >> 2 = 00110_2$ Without Sign Extension

LEFT SHIFT

$$\begin{array}{r} 1101_2 \\ \ll 3 \\ \hline 1101000_2 \end{array}$$

```python
#python example
x = 13
y = x << 3
print(y)
#prints 104
```

SHIFTING MULTIPLYING AND DIVIDING BY 2

A left shift is equivalent to multiplying by 2

0001 << 1 = 0010 (2).

0001 << 2 = 0100 (4)

0001 << 3 = 1000 (8)

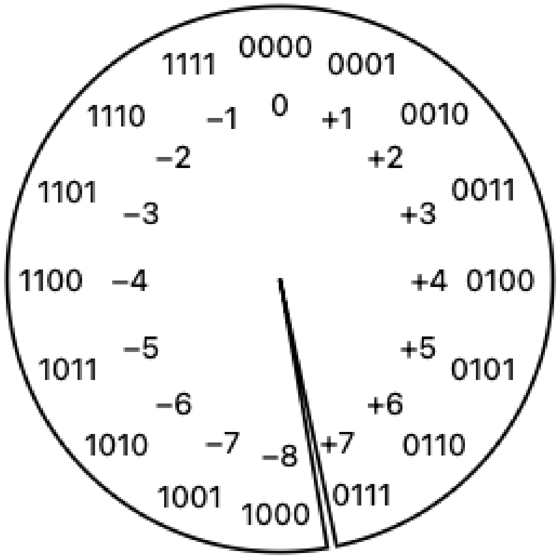
A right shift is equivalent to dividing by 2

01000 >> 1 = 0100 (4)

01000 >> 2 = 0010 (2)

01000 >> 3 = 0001 (1)

BITWISE INVERT ~

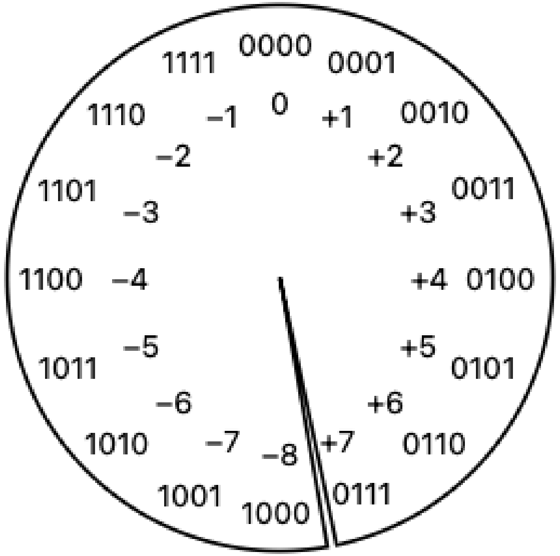


$$\begin{array}{r} \sim\ 0000_2 \\ \hline 1111_2 \end{array}$$

```
#python example
x = 0
z = ~x
print(z)
#prints -1
```


University of Virginia | Engineering

BITWISE INVERT ~



$$\begin{array}{r} \sim\ 0110_2 \\ \hline 1001_2 \end{array}$$

```
#python example
x = 6
z = ~x
print(z)
#prints -7
```

SETTING BITS TO 1

Set the last bit of this variable 1

$$
\begin{array}{r}
0000_2 \\
|\ \ 0001_2 \\
\hline
0001_2
\end{array}
$$

```
#python example
x = 0
x = x | 0x01
print(x)
#prints 1
```


UNIVERSITY of VIRGINIA | ENGINEERING

SETTING BITS TO 1

Set the third bit of x to 1

$$
\begin{array}{r}
0000_2 \\
|\ \ 0100_2 \\
\hline
0100_2
\end{array}
$$

```
#python example
x = 0
x = x | 0x04
print(x)
#prints 4
```

Question: What if it was already one?

SETTING BITS TO 1

Set the n bit of x to 1

$$
\begin{array}{r l}
 & 0000_2 \\
| & 0001_2 \quad << n\ (3) \\
\hline
 & 1000_2
\end{array}
$$

```
#python example
x = 0
n = 3
x = x | (0x01 << n)
print(x)
#prints 8
```

Question: What if it was already one?

FLIPPING BITS

Flip the second bit of x. 1 => 0 and 0 => 1

$$
\begin{array}{r}
1100_2 \\
\wedge \quad 0010_2 \\
\hline
1110_2
\end{array}
$$

What if the nth bit was 1 instead?

FLIPPING BITS

Flip the **nth** bit of x. 1 => 0 and 0 => 1

$$\begin{array}{r} 1100_2 \\ \wedge \quad 0010_2 \\ \hline 1110_2 \end{array}$$

```python
#python example
x = 12
n = 1
x = x ^ (0x01 << n)
print(x)
#prints 14
```

MASKING (EXTRACTING BITS)

The idea of masking is that we can extract a certain section of bits by anding.

$$
\begin{array}{r}
11011100_2 \\
\&\ 11110000_2 \\
\hline
\boxed{1101}0000_2
\end{array}
$$

Upper 4 bits extracted

University of Virginia ENGINEERING

MASKING (EXTRACTING BITS)

The idea of masking is that we can extract a certain section of bits by anding.

$$
\begin{array}{r}
11011100_2 \\
\&\ 00001111_2 \\
\hline
0000\boxed{1100_2}
\end{array}
$$

Lower 4 bits extracted

```python
#python example
x = 220
mask = 0x0F
x = x & mask
print(x)
#prints 12
```

MASKING (EXTRACTING BITS)

The idea of masking is that we can extract a certain section of bits by anding.

$$\begin{array}{r} 11011100_2 \\ \&\ 11110000_2 \\ \hline \boxed{1101}0000_2 \end{array}$$

Upper 4 bits extracted

```
#python example
x = 220
mask = ~0x0F
x = x & mask
print(x)
#prints 208
```

COMBINING

We can also set multiple bits simultaneously

$$
\begin{array}{r}
10100000_2 \\
|\ \ 00001111_2 \\
\hline
10101111_2
\end{array}
$$

```
#python example
b = 0x0F
a = 0xA0
x = a | b
print(hex(x))
#prints 0xAF
```

PARITY

Suppose you want to want to calculate the even parity of x.

If the total amount of "1" bits is odd, the parity value is 1, otherwise it is zero.

0010 parity bit is 1

0110 parity bit is 0

```
parity = 0
repeat 32 times:
    parity ^= (x&1)
     x >>= 1
```

PARALLEL EVALUATION

Observe that xor is both transitive and associative; thus we can re-write

$$x0 \oplus x1 \oplus x2 \oplus x3 \oplus x4 \oplus x5 \oplus x6 \oplus x7$$

using transitivity as

$$x0 \oplus x4 \oplus x1 \oplus x5 \oplus x2 \oplus x6 \oplus x3 \oplus x7$$

and using associativity as

$$(x0 \oplus x4) \oplus (x1 \oplus x5) \oplus (x2 \oplus x6) \oplus (x3 \oplus x7)$$

and then compute the contents of all the parentheses at once via

x ^ (x>>4).

PARALLEL EVALUATION

$x0 \oplus x1 \oplus x2 \oplus x3 \oplus x4 \oplus x5 \oplus x6 \oplus x7$
using transitivity as

$x0 \oplus x4 \oplus x1 \oplus x5 \oplus x2 \oplus x6 \oplus x3 \oplus x7$

and using associativity as
$(x0 \oplus x4) \oplus (x1 \oplus x5) \oplus (x2 \oplus x6) \oplus (x3 \oplus x7)$

and then compute all at once via
x ^ (x>>4).

```
x ^= (x>>16)
x ^= (x>>8)
x ^= (x>>4)
x ^= (x>>2)
x ^= (x>>1)
parity = (x & 1)
```

UNIVERSITY of VIRGINIA | ENGINEERING