CS4 Image Water Marking Lab
Topics:

- Bit Shifting
- Bitwise Logical Operations
- Combining Bits with Shifting and Logic Operators
Bit Shifting
Bit Shifting

Shifting to the right: remove bits
Bit Shifting

Shifting to the right: remove bits

Example
Bit Shifting

Shifting to the right: remove bits

Example

0b10010101

decimal value 149
Bit Shifting

Shifting to the right: remove bits

Example

0b10010101

Decimal value 149
Bit Shifting

Shifting to the right: remove bits

Example

\[ \text{bit representation: } 0b10010101 \]

Decimal value: 149

Black hole
Bit Shifting

Shifting to the right: remove bits

Example

0b10010101

Decimal value 74
Bit Shifting

Shifting to the right: remove bits

Example

```
0b10010101
```

decimal value 37
Bit Shifting

Shifting to the right: remove bits

Example

Black hole

\[ 0b100101 \]

decimal value 18
Bit Shifting

Shifting to the right: remove bits

Example

Black hole

0b1001

decimal value 9
Bit Shifting

Shifting to the right: remove bits

Example

0b100

decimal value 4
Bit Shifting

Shifting to the right: remove bits

Example

In Python, \( x >> n \) means shift \( x \) to the right by \( n \) bits.

Decimal value 4
Bit Shifting
Bit Shifting
Shifting to the left: add 0s on right
Bit Shifting

Shifting to the left: add 0s on right

Example
Bit Shifting

Shifting to the left: add 0s on right

Example

```
0b101
```

decimal value 5
Bit Shifting

Shifting to the left: add 0s on right

Example

Black hole

0b101

decimal value 5
Bit Shifting

Shifting to the left: add 0s on right

Example

Black hole

0b101

decimal value 5
Bit Shifting

Shifting to the left: add 0s on right

Example

Black hole

Decimal value 10

Ob1010
Bit Shifting

Shifting to the left: add 0s on right

Example

Decimal value 20: $0b1010000$
Bit Shifting

Shifting to the left: add 0s on right

Example

Decimal value 40

\texttt{0b10100000}

Black hole
Bit Shifting

Shifting to the left: add 0s on right

Example

```
0b10100000
```

decimal value 80
Bit Shifting

Shifting to the left: add 0s on right

Example

Decimal value 160

0b10100000

Black hole
Bit Shifting

Shifting to the left: add 0s on right

Example

In python, \( x \ll n \) means shift \( x \) to the left by \( n \) bits

Decimal value 160

0b10100000

Black hole
Bit Shifting

Q: How to transform

10110110 into 10100000

using only bit shifting?
Bit Shifting

Q: How to transform 

10110110 into 10100000

using only bit shifting?

Clue: convert last 5 bits to 0
Bit Shifting

Q: How to transform 10110110 into 101000000 using only bit shifting?

Clue: convert last 5 bits to 0

(10110110 >> 5) << 5
Bit Shifting

Q: How to transform 10110110 into 10100000 using only bit shifting?

Clue: convert last 5 bits to 0

(10110110 >> 5) << 5
Bit Shifting

Q: How to transform 10110110 into 10100000 using only bit shifting?

Clue: convert last 5 bits to 0

(10110110 >> 5) << 5

(converts 182 into 160)
Bitwise Logical Operations

Bitwise “or”

\[ b_1 | b_2 \text{ is 1 when either } b_1 = 1 \text{ or } b_2 = 1 \text{ (or both)} \]
Bitwise Logical Operations

Bitwise “or”

\[ b_1 | b_2 \text{ is 1 when either } b_1 = 1 \text{ or } b_2 = 1 \text{ (or both)} \]
\[ b_1 | b_2 \text{ is 0 if both } b_1 \text{ and } b_2 \text{ are 0} \]
Bitwise Logical Operations

Bitwise “or”

\[ b_1 | b_2 \text{ is 1 when either } b_1 = 1 \text{ or } b_2 = 1 \text{ (or both)} \]

\[ b_1 | b_2 \text{ is 0 if both } b_1 \text{ and } b_2 \text{ are 0} \]

\[ 0 | 0 = 0 \]
\[ 1 | 0 = 1 \]
\[ 0 | 1 = 1 \]
\[ 1 | 1 = 1 \]
Bitwise Logical Operations

Bitwise “or”

$b_1 | b_2$ is 1 when either $b_1 = 1$ or $b_2 = 1$ (or both)

$b_1 | b_2$ is 0 if both $b_1$ and $b_2$ are 0

$0 | 0 = 0$

$1 | 0 = 1$

$0 | 1 = 1$

$1 | 1 = 1$

$11010011 | 10010110$
Bitwise Logical Operations

Bitwise “or”

\[ b_1 | b_2 \text{ is 1 when either } b_1 = 1 \text{ or } b_2 = 1 \text{ (or both)} \]

\[ b_1 | b_2 \text{ is 0 if both } b_1 \text{ and } b_2 \text{ are 0} \]

\[
\begin{array}{c}
0 | 0 = 0 \\
1 | 0 = 1 \\
0 | 1 = 1 \\
1 | 1 = 1 \\
\end{array}
\]

\[
\begin{array}{c}
11010011 \\
10010110 \\
\hline
\end{array}
\]
Bitwise Logical Operations

Bitwise “or”

$b_1 | b_2$ is 1 when either $b_1 = 1$ or $b_2 = 1$ (or both)

$b_1 | b_2$ is 0 if both $b_1$ and $b_2$ are 0

0|0 = 0
1|0 = 1
0|1 = 1
1|1 = 1

11010011  |  10010110
11111111

11
Bitwise Logical Operations

Bitwise “or”

\[ b_1 | b_2 \] is 1 when either \( b_1 = 1 \) or \( b_2 = 1 \) (or both)
\[ b_1 | b_2 \] is 0 if both \( b_1 \) and \( b_2 \) are 0

\[
\begin{align*}
0 | 0 &= 0 \\
1 | 0 &= 1 \\
0 | 1 &= 1 \\
1 | 1 &= 1
\end{align*}
\]

\[
\begin{array}{c}
11010011 \\
| \\
10010110
\end{array}
\]

\[
111
\]
Bitwise Logical Operations

Bitwise “or”

\[ b_1 | b_2 \text{ is 1 when either } b_1 = 1 \text{ or } b_2 = 1 \text{ (or both)} \]

\[ b_1 | b_2 \text{ is 0 if both } b_1 \text{ and } b_2 \text{ are 0} \]

\[
\begin{array}{ll}
0 | 0 &= 0 \\
1 | 0 &= 1 \\
0 | 1 &= 1 \\
1 | 1 &= 1
\end{array}
\]
Bitwise Logical Operations

Bitwise “or”

$b_1 | b_2$ is 1 when either $b_1 = 1$ or $b_2 = 1$ (or both)

$b_1 | b_2$ is 0 if both $b_1$ and $b_2$ are 0

$0 | 0 = 0$

$1 | 0 = 1$

$0 | 1 = 1$

$1 | 1 = 1$

\[ \begin{array}{c}
0 | 0 = 0 \\
1 | 0 = 1 \\
0 | 1 = 1 \\
1 | 1 = 1
\end{array} \]

\[ \begin{array}{c}
11010011 \\
\downarrow \\
10010110 \\
\hline \\
11010111
\end{array} \]
Bitwise Logical Operations

Bitwise “and”

\( b_1 \& b_2 \) is 1 when both \( b_1 = 1 \) and \( b_2 = 1 \)
Bitwise Logical Operations

Bitwise “and”

\[ b_1 \& b_2 \text{ is 1 when both } b_1 = 1 \text{ and } b_2 = 1 \]
\[ b_1 \& b_2 \text{ is 0 if either of } b_1 \text{ or } b_2 \text{ is 0} \]
Bitwise Logical Operations

Bitwise “and”

$b_1 \& b_2$ is 1 when both $b_1 = 1$ and $b_2 = 1$

$b_1 \& b_2$ is 0 if either of $b_1$ or $b_2$ is 0

0&0 = 0
1&0 = 0
0&1 = 0
1&1 = 1
Bitwise Logical Operations

Bitwise “and”

\[ b_1 \& b_2 \text{ is 1 when both } b_1 = 1 \text{ and } b_2 = 1 \]
\[ b_1 \& b_2 \text{ is 0 if either of } b_1 \text{ or } b_2 \text{ is 0} \]

0&0 = 0
1&0 = 0
0&1 = 0
1&1 = 1

11010011 & 10010110
Bitwise Logical Operations

Bitwise “and”

\[ b_1 \& b_2 \text{ is 1 when both } b_1 = 1 \text{ and } b_2 = 1 \]

\[ b_1 \& b_2 \text{ is 0 if either of } b_1 \text{ or } b_2 \text{ is 0} \]

\[
\begin{array}{ccc}
0 \& 0 &=& 0 \\
1 \& 0 &=& 0 \\
0 \& 1 &=& 0 \\
1 \& 1 &=& 1 \\
\end{array}
\]

\[
\begin{array}{c}
11010011 \\
\&
10010110 \\
\hline
0
\end{array}
\]
Bitwise Logical Operations

Bitwise “and”

\[ b_1 \& b_2 \] is 1 when both \( b_1 = 1 \) and \( b_2 = 1 \)

\[ b_1 \& b_2 \] is 0 if either of \( b_1 \) or \( b_2 \) is 0

\[
\begin{align*}
0 \& 0 &= 0 \\
1 \& 0 &= 0 \\
0 \& 1 &= 0 \\
1 \& 1 &= 1
\end{align*}
\]

\[
\begin{array}{c}
11010011 \\
\& \\
10010110 \\
\end{array}
\]

\[ 10 \]
Bitwise Logical Operations

Bitwise “and”

\[ b_1 \& b_2 \text{ is 1 when both } b_1 = 1 \text{ and } b_2 = 1 \]

\[ b_1 \& b_2 \text{ is 0 if either of } b_1 \text{ or } b_2 \text{ is 0} \]

\[
\begin{align*}
0 \& 0 &= 0 \\
1 \& 0 &= 0 \\
0 \& 1 &= 0 \\
1 \& 1 &= 1
\end{align*}
\]

\[
\begin{array}{c}
11010011 \\
\& 10010110 \\
\hline
010
\end{array}
\]
Bitwise Logical Operations

Bitwise “and”

\[ b_1 \& b_2 \text{ is } 1 \text{ when both } b_1 = 1 \text{ and } b_2 = 1 \]

\[ b_1 \& b_2 \text{ is } 0 \text{ if either of } b_1 \text{ or } b_2 \text{ is } 0 \]

\[
\begin{array}{c}
0 \& 0 = 0 \\
1 \& 0 = 0 \\
0 \& 1 = 0 \\
1 \& 1 = 1 \\
\end{array}
\]

\[
\begin{array}{c}
11010011 \\
\&
10010110 \\
\hline
0010
\end{array}
\]
Bitwise Logical Operations

Bitwise “and”

\( b_1 \& b_2 \) is 1 when both \( b_1 = 1 \) and \( b_2 = 1 \)

\( b_1 \& b_2 \) is 0 if either of \( b_1 \) or \( b_2 \) is 0

\[
\begin{align*}
0 \& 0 &= 0 \\
1 \& 0 &= 0 \\
0 \& 1 &= 0 \\
1 \& 1 &= 1
\end{align*}
\]

\[
\begin{array}{c}
11010011 \\
\& 10010110 \\
\hline
10010010
\end{array}
\]
Bit Manipulating Colors

In binary

\[(0b11011111 \gg 3) \ll 3 = 0b11011000\]
Bit Manipulating Colors

In binary

\[(0b11011111 \gg 3) \ll 3 = 0b11011000\]

Same thing in decimal

\[(223 \gg 3) \ll 3 = 216\]
Bit Manipulating Colors

In binary
\[(0b11011111 >> 3) << 3 = 0b11011000\]

Same thing in decimal
\[(223 >> 3) << 3 = 216\]

Human eye can’t really tell the difference.
Apply this to all \((r,g,b)\) bytes for all pixels
Idea:

Combine significant bits of two numbers together.

Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]
Idea:

Combine significant bits of two numbers together.
Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]
Idea:

Combine significant bits of two numbers together.

Example: 3 bit embedding

\[
x = 0b11010011 \quad y = 0b01010011
\]

\[
z = 0b11010010
\]
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[
x = 0b11010011 \quad y = 0b01010011
\]

\[
z = 0b11010010
\]

\[
(x \gg 3) \ll 3 = \]

Idea:
Combine significant bits of two numbers together. Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]

\[ z = 0b11010000 \]

\[(x >>> 3) << 3 = 0b1101000000\]
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[
x = \text{0b}11010011 \\
y = \text{0b}01010011 \\
z = \text{0b}11010010
\]

\[
(x \ggg 3) \lll 3 = \text{0b}11010000 \\
y \ggg 5 = \text{ } \quad \text{0b}11010000
\]
Idea:

Combine significant bits of two numbers together.
Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]

\[ z = 0b11010010 \]

\[ (x >> 3) << 3 = 0b11010000 \]

\[ y >> 5 = 0b00000010 \]
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]

\[ z = 0b11010010 \]

0b11010000
0b00000010
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]

\[ z = 0b11010010 \]

\[ 0b11010000 \]

\[ 0b00000010 \]
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[ x = 0b11010011 \quad y = 0b01010011 \]

\[ z = 0b11010010 \]

\[ 0b11010000 \quad | \quad 0b00000010 \]
Idea:
Combining significant bits of two numbers together.

Example: 3 bit embedding

\[
x = 0b11010011 \quad y = 0b01010011
\]

\[
z = 0b11010010
\]

\[
0b11010000 \quad 0b00000010
\]

\[
0b11010010
\]
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[ x = \texttt{0b11010011} \quad y = \texttt{0b01010011} \]

\[ z = \texttt{0b11010010} \]
Idea:
Combine significant bits of two numbers together.
Example: 3 bit embedding

\[x = 0b11010011\]  \[y = 0b01010011\]

\[z = 0b11010010\]

0b11010000
\[0b11010000\]
| 0b000000010
\[0b110100010\]

\[x = 211\]
\[y = 83\]
\[z = 210\]
How to recover \( y \)?

Recall: 3 bit embedding

\[ \text{0b11010010} \]
How to recover $y$?

Recall: 3 bit embedding

$0b11010010$
How to recover $y$?

Recall: 3 bit embedding

\[
\begin{align*}
\text{0b} & 11010010 \\
\text{0b} & 000000111
\end{align*}
\]

3 bits of 1s, the rest 0
How to recover $y$?

Recall: 3 bit embedding

$0b11010010$

& $0b00000111$

3 bits of 1s, the rest 0
How to recover $y$?

Recall: 3 bit embedding

$\text{0b11010010} \& \text{0b00000111}$

$\text{0b000000010}$

3 bits of 1s, the rest 0
How to recover $y$?

Recall: 3 bit embedding

\[
\begin{array}{c}
\text{0b11010010} \\
\& \text{0b00000111} \\
\hline
\text{0b00000010}
\end{array}
\]

3 bits of 1s, the rest 0
How to recover $y$?

Recall: 3 bit embedding

\[
\begin{array}{c}
0b11010010 \\
\& 0b00000111 \\
\hline
0b000000010
\end{array}
\]

3 bits of 1s, the rest 0

\[0b000000010 \ll 5 = 010000000 \]

(64 in decimal)
How to recover \( y \)?

Recall: 3 bit embedding

\[
\begin{array}{c}
\text{0b11010010} \\
\& \text{0b00000111} \\
\hline
\text{0b00000010}
\end{array}
\]

3 bits of 1s, the rest 0

\[
\text{0b00000010} \ll 5 = 0100000000
\]

(64 in decimal)

original \( y \) was 83
Today's lab

> wget www.cs.ucsb.edu/~bang/getfiles.py
Today’s lab

wget www.cs.ucsb.edu/~bang/getfiles.py
1 bit of embedding
Today’s lab

> wget www.cs.ucsb.edu/~bang/getfiles.py
6 bits of embedding
Today’s lab

> wget www.cs.ucsb.edu/~bang/getfiles.py

Try for 1,2,3,4,5,6,7 bits of embedding.

What happens to the quality of the original image and the quality of the watermarking image as we change the number of bits used for embedding?