clicker.cs.illinois.edu

Q1

~Code~
340

CS 340

Bitwise Operations

Updates

1. Exam 1 scores are released.
    a. More information on Campus Wire

1. MP 3 - PNG due today

1. MP 4 - UTF-8 out today (due next Tuesday)

Bitwise Operations

Today's LGs - Build on your mental model of how data is stored and interpreted on a computer (1's and 0's).

Be able to converse about bits
Be able to shift bits
Be able to apply logical operations at a bit level
Be able to use bit operations to isolate the bits you need
Be able to convert between code points and UTF-8

Agenda

1. Addressing misconceptions
    a. Important terms
    b. Representing bits versus interpreting bits
2. Bitwise Operations
    a. Bit shifting
    b. Bit logic operations

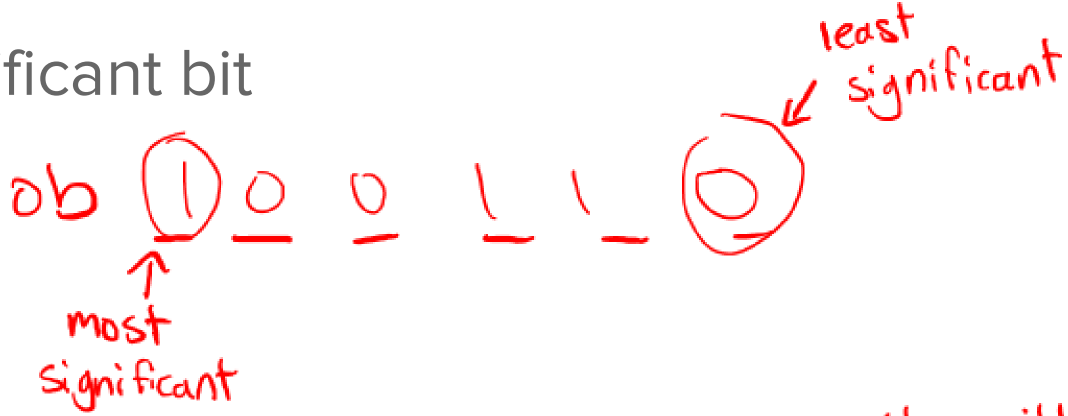
2. Bit Mask

3. MP4 - UTF-8

Important Terms 0b01

Bit Vector - a fixed length sequence of bits

Most/least significant bit



Clear

↳verb - replace a bit with 0 OR replace all bits with 0

Important Terms 0b10

ith bit -The place in a bit vector

2nd

ob 0100 1001

1st 0th

Set - set a bit = make it 1

The 3rd bit is set = that bit is 1

a data structure

Zero - opposite of 1

a bit vector of all 0's > another word to mean clear

a single bit of value 1

Representing Bits Versus Interpreting Bits

information is stored in 1's and 0's

Hex use case 1 – to indicate the 1's and 0's set

Hex use case 2 – to represent an amount

Example 0b01

Address	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009
Value	F6	DC	6D	CD	58	AF	22	49	BC	E3

indicates

0010 0010

but could represent the middle of an int value

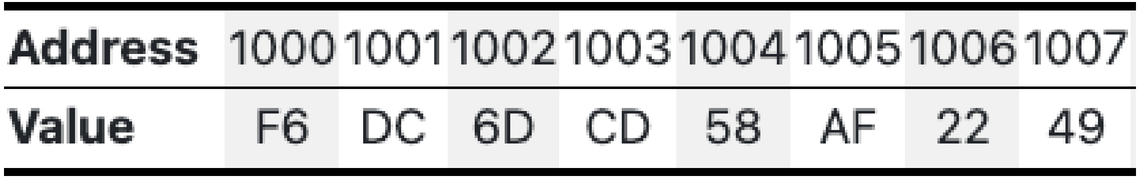
Example 0b10

bit vector =

$$0\,1\,0\,0\;\;1\,1\,0\,0 = 76 = \overset{\overset{\text{amount}}{\downarrow}}{0x4C}$$

but the 1's and 0's could mean something else

What could the value at address 1006 be?



Address	1000	1001	1002	1003	1004	1005	1006	1007
Value	F6	DC	6D	CD	58	AF	22	49

clicker.cs.illinois.edu

Q2

~Code~
340

[✓ A)] A bit vector

[✓ B)] An ascii character

[✓ C)] The middle byte of an int

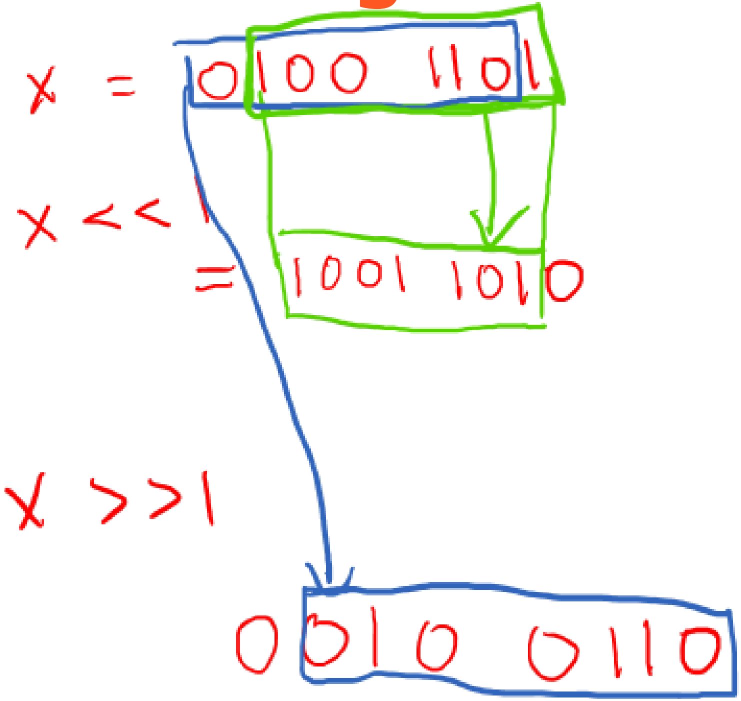
[✓ D)] Part of a struct

Bitwise Operations in C

1. Bit shifting << or >>

1. Bit operations (AND, OR, XOR, NOT)

Bit Shifting - shift bits left or right and add 0's



x = 0100 1101

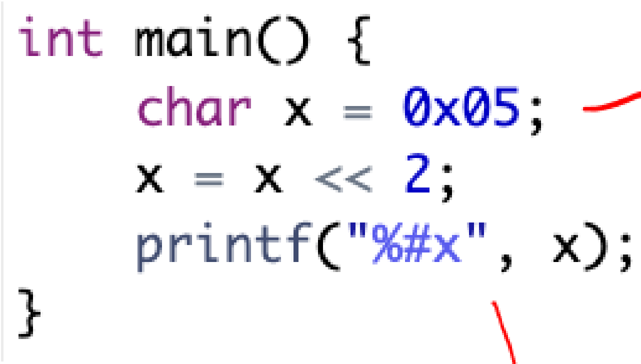
x << 1 = 1001 1010

x >> 1

0010 0110

use unsigned!

Bit Shifting Example in C



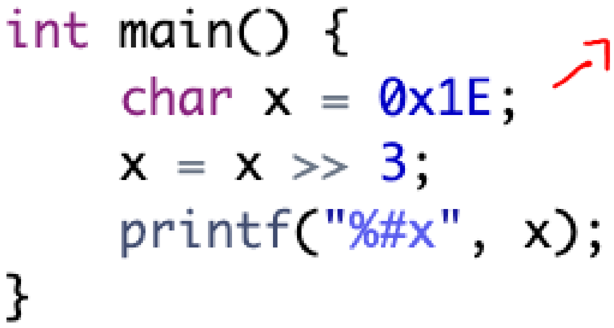
```
int main() {
    char x = 0x05;
    x = x << 2;
    printf("%#x", x);
}
```

~~0~~000 0101

0001 0100

0x14

What prints?



```
int main() {
    char x = 0x1E;
    x = x >> 3;
    printf("%#x", x);
}
```

0001 1~~110~~

0000 0011

0x03

clicker.cs.illinois.edu

Q3

~Code~
340

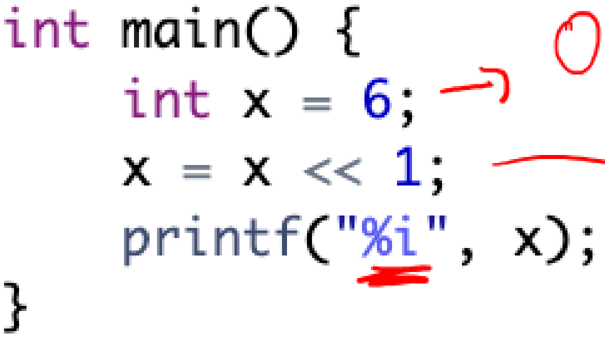
A) 0x01

B) 0xE0

[✓ C) 0x03]

D) 0x0E

What prints (challenge)?



```
int main() {
    int x = 6;
    x = x << 1;
    printf("%i", x);
}
```

00... 0110

00... 1100

clicker.cs.illinois.edu

Q4

~Code~
340

A) 0x0C

B) 0x12

[✓ C) 12]

D) 24

Bit Operations

logic	C
AND	&
OR	|
XOR	^
NOT	~

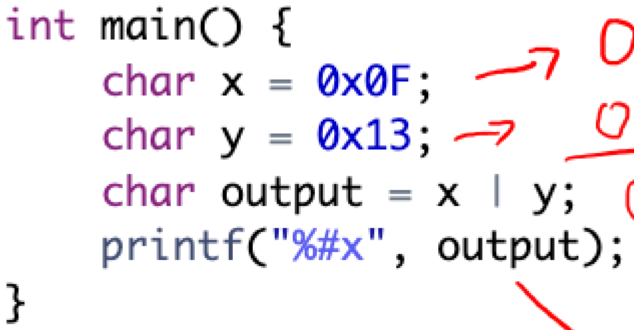
9 AND 1

```
  1001
& 0001
  ----
  0001
```

9 XOR 1

```
  1001
^ 0001
  ----
  1000
```

Bit Operations Example in C



```c
int main() {
    char x = 0x0F;
    char y = 0x13;
    char output = x | y;
    printf("%#x", output);
}
```

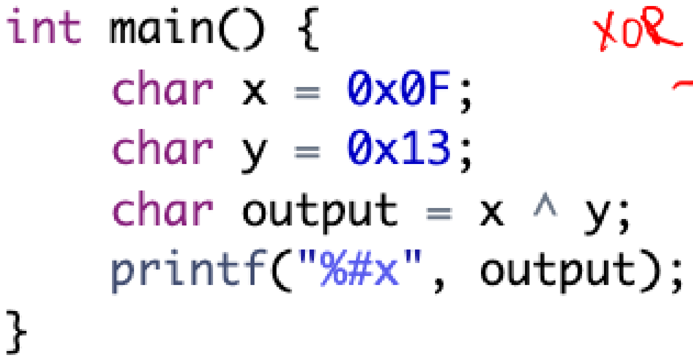
0000 1111

0001 0011 OR

0001 1111

0x1F

What prints?



```c
int main() {
    char x = 0x0F;
    char y = 0x13;
    char output = x ^ y;
    printf("%#x", output);
}
```

```
    0000 1111
XOR 0001 0011
-------------
    0001 1100
        ↓
      0x1C
```

clicker.cs.illinois.edu

Q5

~Code~
340

A) 0x28

B) 0x1F

C) 0x30

[✓ D) 0x1C]

What 8 bit value does this produce?

NOT

`((~0) << 3)`

1111 1111

← 3

clicker.cs.illinois.edu

Q6

~Code~
340

A) 0000 0111

B) 0000 0011

[✓ C) 1111 1000]

D) 1111 1011

What 8 bit value does this produce?

$$\underbrace{(((\sim 0) \ll 3)}_{1111\ 1000}\ \underset{\text{XOR}}{\text{\^{}}}\ \underbrace{((\sim 0) \ll 5))}_{1110\ 0000}$$

clicker.cs.illinois.edu

Q7

~Code~
340

A) 1111 1000

B) 1110 0000

~~C) 0001 1000~~

D) 0011 1000

I want just the middle 4 bits from a byte to remain.

Mask!

x = 1001 0011

Mask = 0001 1000

x & mask = 0001 0000

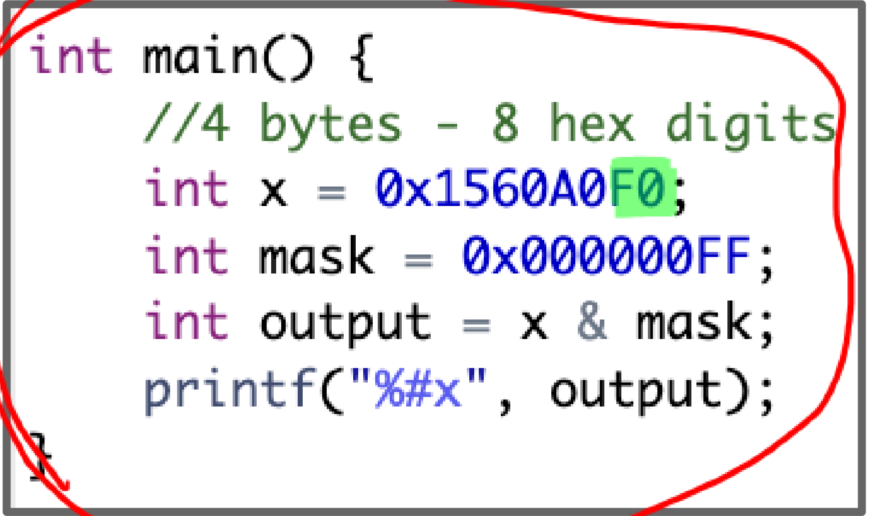
How would I get only the 8 least significant bits from from x?

goal: 0x00 00 00 F0

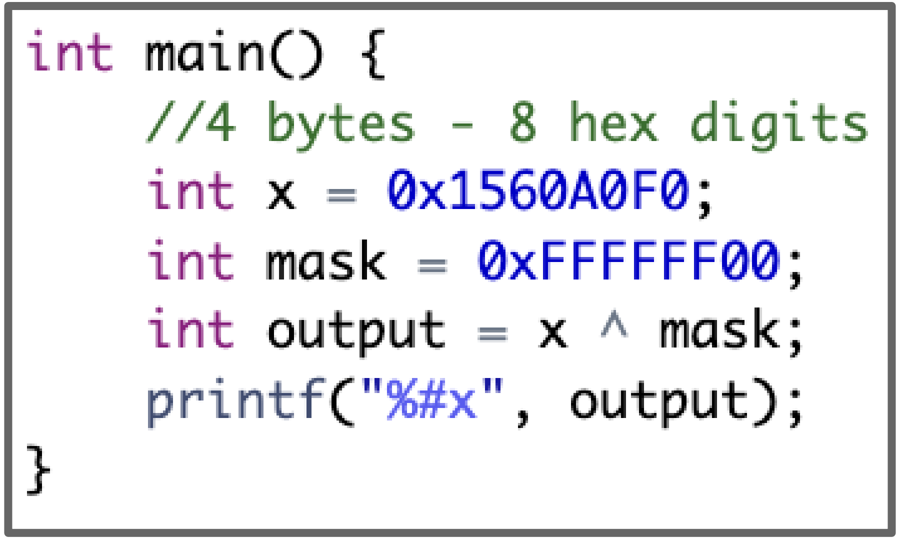
clicker.cs.illinois.edu

Q8

~Code~
340



```
int main() {
    //4 bytes - 8 hex digits
    int x = 0x1560A0F0;
    int mask = 0x000000FF;
    int output = x & mask;
    printf("%#x", output);
}
```



```
int main() {
    //4 bytes - 8 hex digits
    int x = 0x1560A0F0;
    int mask = 0xFFFFFF00;
    int output = x ^ mask;
    printf("%#x", output);
}
```

Okay… but why?

To use bits within a byte!

UTF-8

bit sets

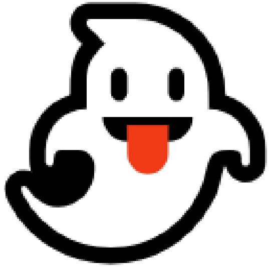
UTF-8

Char - 1 byte value — values 0-255

ASCII - mapping from 0-127 values to characters

Unicode - bigger ascii table!

UTF-8 - variable length encoding for unicode



UTF-8

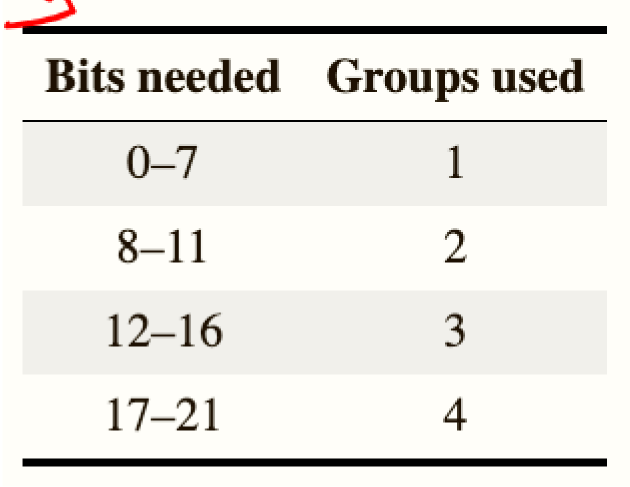
Code Point - a number that can be held in an int

UTF-8 - 1-4 bytes representing a code point

(Encoding) Code point -> UTF-8

~~3001 → [0x67][78][96]~~

code point	# of bits
2	2 bits
512	10 bits
0x1F33D	17 bits



Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

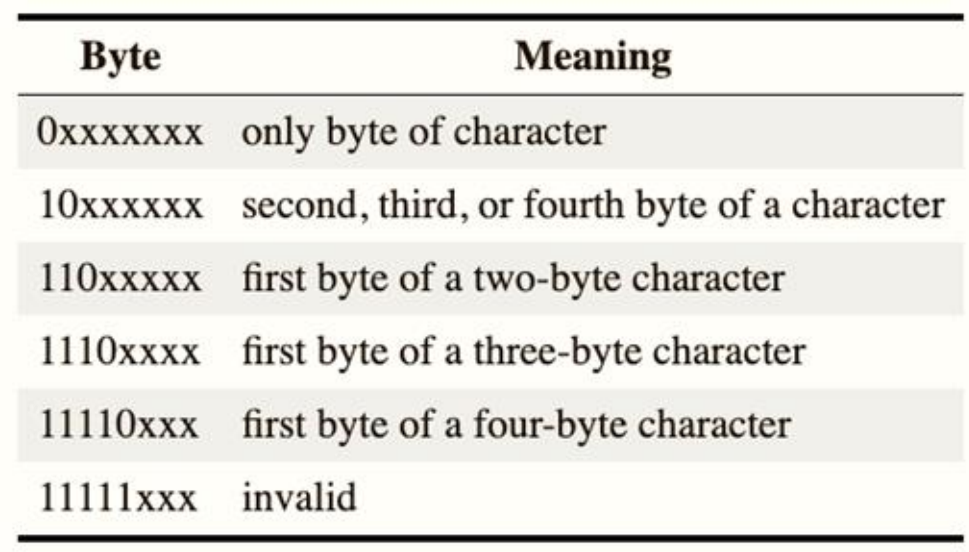
(Encoding) Code point -> UTF-8

group1 = 0xxx xxxx

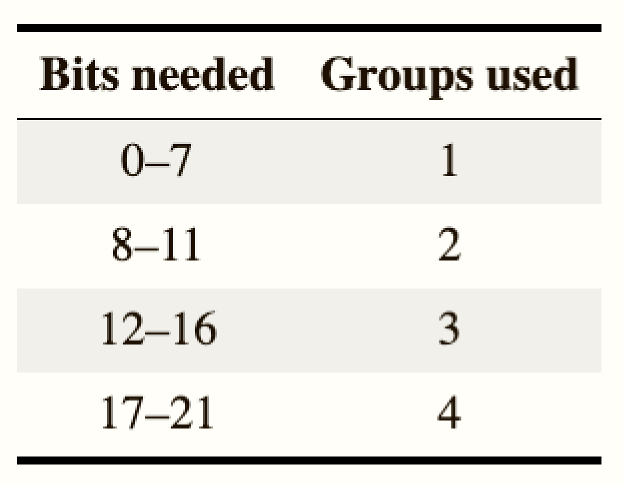
group2 = 110x xxxx 10xx xxxx

group3 = 1110 xxxx 10xx xxxx 10xx xxxx

group4 = 1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx



Byte	Meaning
0xxxxxxx	only byte of character
10xxxxxx	second, third, or fourth byte of a character
110xxxxx	first byte of a two-byte character
1110xxxx	first byte of a three-byte character
11110xxx	first byte of a four-byte character
11111xxx	invalid



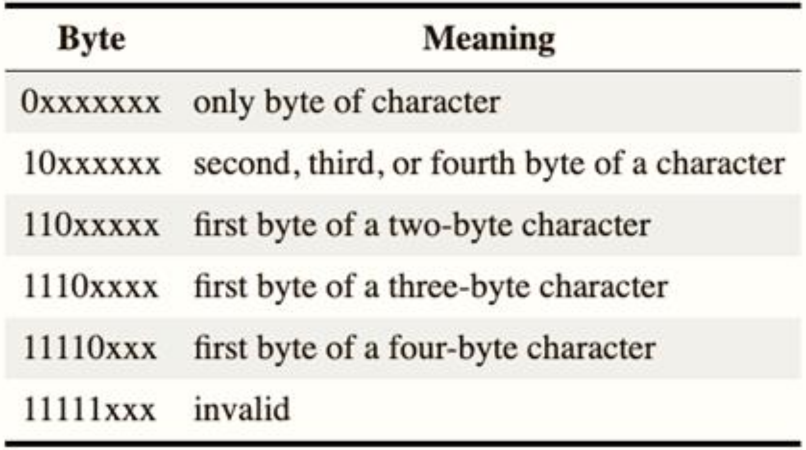
Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

(Encoding) Code point -> UTF-8

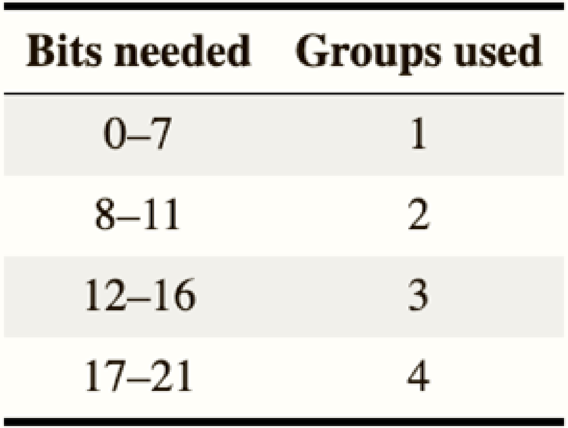
'a' = 0x61 = 0110 0001 = 7 bits = group1

0 1 1 0 0 0 0 1

↑
header



Byte	Meaning
0xxxxxxx	only byte of character
10xxxxxx	second, third, or fourth byte of a character
110xxxxx	first byte of a two-byte character
1110xxxx	first byte of a three-byte character
11110xxx	first byte of a four-byte character
11111xxx	invalid



Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

(Encoding) Code point -> UTF-8

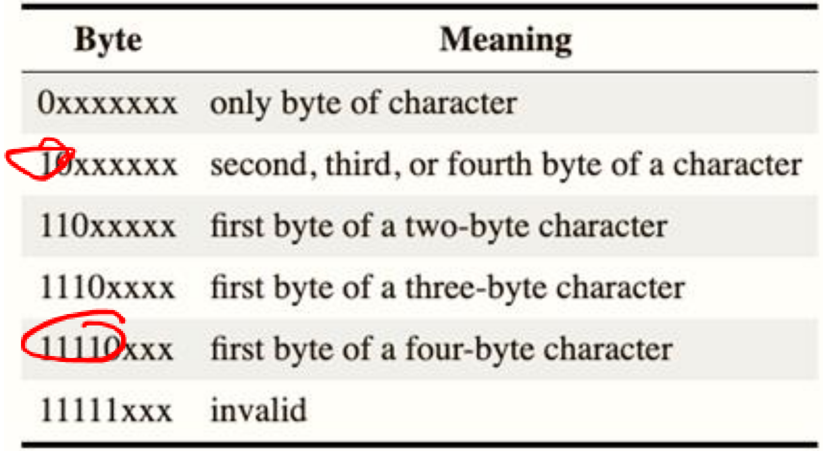
ஹ

1011 1011 1001 = 3001

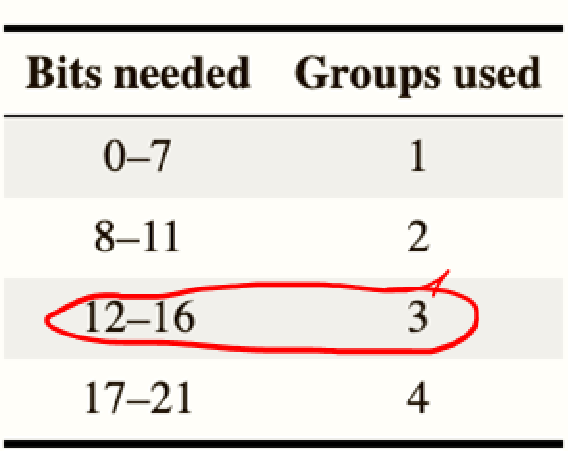
12 bits

1110 0000 10 101110 10 111001

fill in the rest w/ 0s



Byte	Meaning
0xxxxxxx	only byte of character
10xxxxxx	second, third, or fourth byte of a character
110xxxxx	first byte of a two-byte character
1110xxxx	first byte of a three-byte character
11110xxx	first byte of a four-byte character
11111xxx	invalid



Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

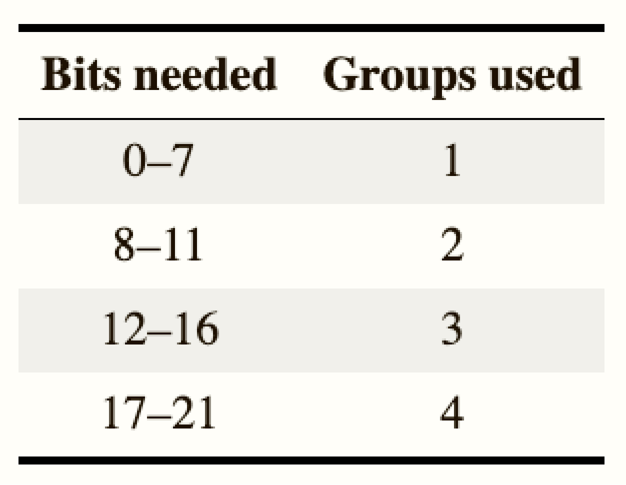
If I need 9 bits to represent a code point, how many bytes will I need to encode it to UTF-8?

2

clicker.cs.illinois.edu

Q9

~Code~
340



Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

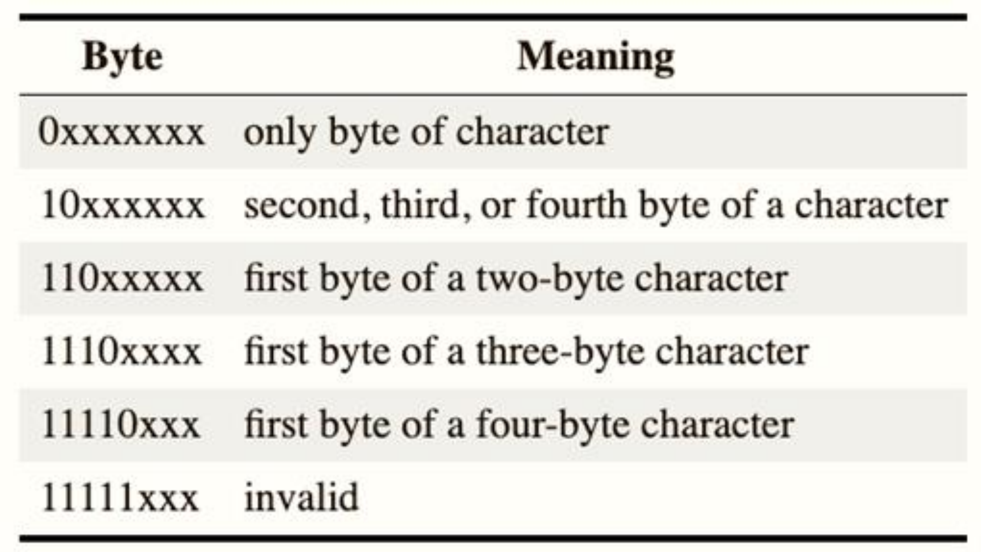
How many X slots are there in a 4-group UTF-8 character?

21

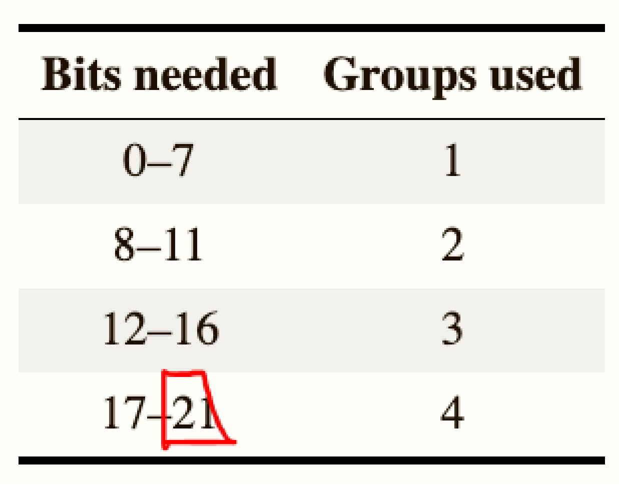
clicker.cs.illinois.edu

Q10

~Code~
340

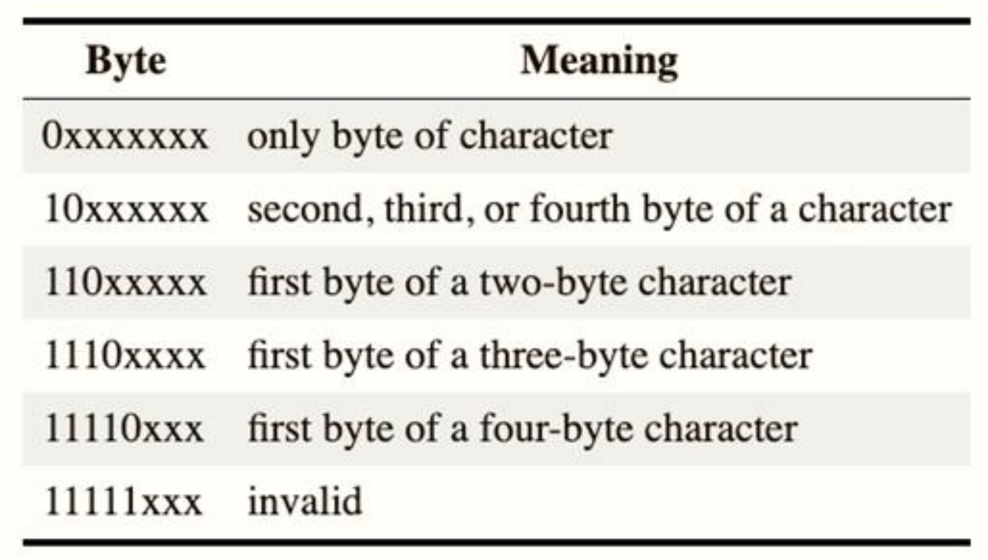


Byte	Meaning
0xxxxxxx	only byte of character
10xxxxxx	second, third, or fourth byte of a character
110xxxxx	first byte of a two-byte character
1110xxxx	first byte of a three-byte character
11110xxx	first byte of a four-byte character
11111xxx	invalid

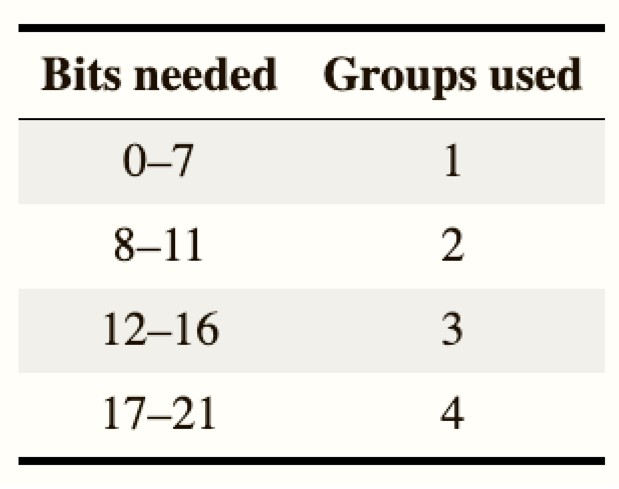


Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

(Decoding) Code point -> UTF-8



Byte	Meaning
0xxxxxxx	only byte of character
10xxxxxx	second, third, or fourth byte of a character
110xxxxx	first byte of a two-byte character
1110xxxx	first byte of a three-byte character
11110xxx	first byte of a four-byte character
11111xxx	invalid



Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

(Decoding) Code point -> UTF-8

~~10000~~

✓

1000 0000

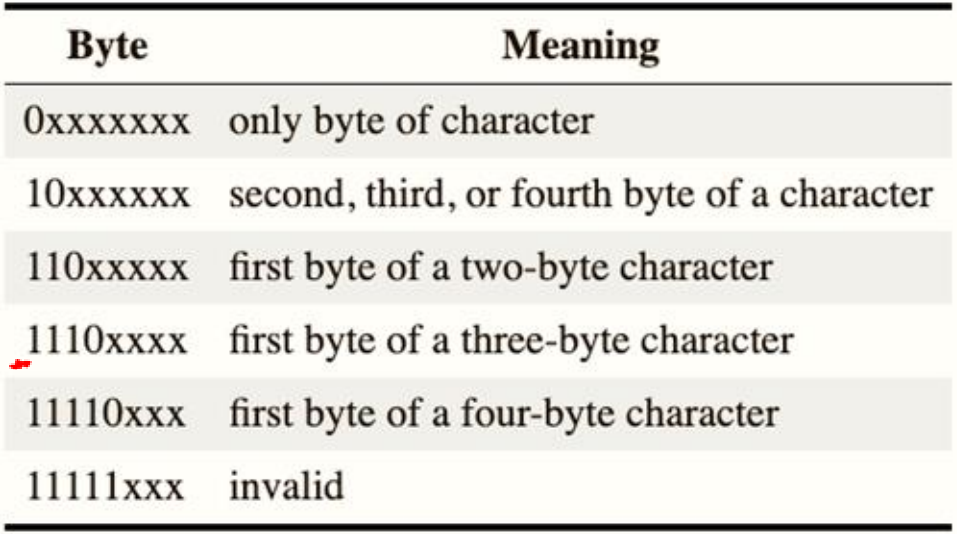
1111 1111

0000 1000 1110 0100 1000 0001 1001 0000

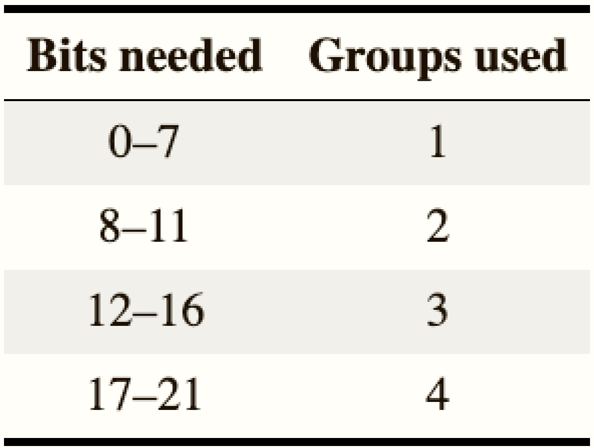
✓

1 character

2nd character



Byte	Meaning
0xxxxxxx	only byte of character
10xxxxxx	second, third, or fourth byte of a character
110xxxxx	first byte of a two-byte character
1110xxxx	first byte of a three-byte character
11110xxx	first byte of a four-byte character
11111xxx	invalid



Bits needed	Groups used
0–7	1
8–11	2
12–16	3
17–21	4

MP4 - UTF-8

~ 11110

~ ~~11111~~

$\boxed{\sim}$ 000001

Bitwise Operations

Today's LGs - Build on your mental model of how data is stored and interpreted on a computer (1's and 0's).

Be able to converse about bits

Be able to shift bits

Be able to apply logical operations at a bit level

Be able to use bit operations to isolate the bits you need

Be able to convert between code points and UTF-8