Binary Numbers

Computer Science 111
Boston University

Vahid Azadeh Ranjbar, Ph.D.

Bits and Bytes

- Everything stored in a computer is essentially a binary number.
  0110110010100111
- Each digit in a binary number is one *bit.*
  - a single 0 or 1
  - based on two voltages: "low" = 0, "high" = 1
- One *byte* is 8 bits.
  - example: 01101100

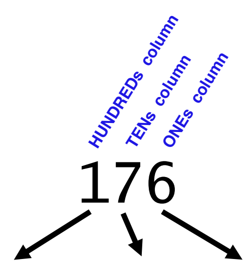
Bits of Data

- A given set of bits can have more than one meaning.

binary	decimal integer	character
01100001	97	'a'
01000110	70	'F'

Representing Integers in Decimal

- In base 10 (decimal), each column represents a power of 10.



1 hundred + 7 tens + 6 ones

$1*10^2 + 7*10^1 + 6*10^0$

Representing Integers in Binary

- In base 2 (binary), each column represents a power of 2.

128's column, SIXTY-FOURs column, THIRTY-TWOs column, SIXTEENs column, EIGHTs column, FOURs column, TWOs column, ONEs column

10110000

$1*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 0*2^0$

128 + 0 + 32 + 16 + 0 + 0 + 0 + 0

also 176!

What Does the Rightmost Bit Tell Us?

128's column, SIXTY-FOURs column, THIRTY-TWOs column, SIXTEENs column, EIGHTs column, FOURs column, TWOs column, ONEs column

10110000

What Does the Rightmost Bit Tell Us?

128's column	SIXTY-FOURs column	THIRTY-TWOs column	SIXTEENs column	EIGHTs column	FOURs column	TWOs column	ONEs column
1	0	1	1	0	0	0	0

- If the rightmost bit is 0, the number is even.
- If the rightmost bit is 1, the number is odd.

Binary to Decimal (On Paper)

- Number the bits from right to left
  - example:

0	1	0	1	1	1	0	1
b7	b6	b5	b4	b3	b2	b1	b0

- For each bit that is 1, add 2^n, where n = the bit number
  - example:

0	1	0	1	1	1	0	1
b7	b6	b5	b4	b3	b2	b1	b0

decimal value = $2^6 + 2^4 + 2^3 + 2^2 + 2^0$

$64 + 16 + 8 + 4 + 1 = 93$

  - another example: what is the integer represented by 1001011?

$2^6 + 2^3 + 2^1 + 2^0$

$= 64 + 8 + 2 + 1 = 75$

Decimal to Binary (On Paper)

- Go in the reverse direction: determine which powers of 2 need to be added together to produce the decimal number.

$$75 = 64 + 8 + 2 + 1$$
$$= 2^6 + 2^3 + 2^1 + 2^0$$
$$= 1001011$$

- Start with the largest power of 2 less than or equal to the number, and work down from there.
  - example: what is 53 in binary?
    - 32 is the largest power of 2 <= 53: 53 = 32 + 21
    - now, break the 21 into powers of 2: 53 = 32 + 16 + 5
    - now, break the 5 into powers of 2: 53 = 32 + 16 + 4 + 1
    - 1 is a power of 2 (2^0), so we're done: 53 = 32 + 16 + 4 + 1 $= 2^5 + 2^4 + 2^2 + 2^0$ = 110101

Which of these is a correct *partial* binary representation of the decimal integer 90?

90 (decimal) → ______ (binary)

A. 101xxx1

B. 111xxx1

C. 101xxx0

D. 111xxx0

E. none of these

an x denotes a "hidden" bit that we aren't revealing

Hint: You shouldn't need to perform the full conversion (i.e., you shouldn't need to determine the hidden bits)!

Which of these is a correct *partial* binary representation of the decimal integer 90?

90 (decimal) → ______ (binary)

A. 101xxx1
B. 111xxx1
C. 101xxx0
D. 111xxx0
E. none of these

Which answers can be ruled out right away?

Which of these is a correct *partial* binary representation of the decimal integer 90?

90 (decimal) → ______ (binary)

A. ~~101xxx1~~
B. ~~111xxx1~~
C. 101xxx0
D. 111xxx0
E. none of these

Which answers can be ruled out right away?
A and B.
90 is even, so the rightmost bit *must* be a 0.

Which of these is a correct *partial*
binary representation of the decimal integer 90?

90 (decimal) → ______ (binary)

A. ~~101xxx1~~

B. ~~111xxx1~~

C. **101xxx0**

D. 111xxx0

E. none of these

$90 = 64 + 26$
$= 64 + 16 + 10$
$= 2^6 + 2^4 + 10$

need a 1 for bits 6 and 4,
but *not* for bit 5.

Which of these is a correct *partial*
binary representation of the decimal integer 90?

90 (decimal) → ______ (binary)

A. ~~101xxx1~~

B. ~~111xxx1~~

C. **101xxx0**

D. 111xxx0

E. none of these

$90 = 64 + 26$
$= 64 + 16 + 10$
$= 2^6 + 2^4 + 10$
$= 2^6 + 2^4 + 8 + 2$
$= 2^6 + 2^4 + 2^3 + 2^1$
$= 1011010$ (binary)

A Binary Address!

Coincidence,
or careful planning?

Introduction to
Computer Science I

Shifting Bits to the Left

- A left-shift:
  - moves every bit of a binary number to the left
  - adds a 0 in the right-most place
- For example: 1011010

Shifting Bits to the Left

- A left-shift:
  - moves every bit of a binary number to the left
  - adds a 0 in the right-most place
- For example: 1011010
  - a left-shift by 1 gives 10110100

Shifting Bits to the Left

- A left-shift:
  - moves every bit of a binary number to the left
  - adds a 0 in the right-most place
- For example: $1011010_2 = 90_{10}$
  - a left-shift by 1 gives $10110100_2 = 180_{10}$

base 2 base 10

- Left-shifting by 1 doubles the value of a number.

Shifting Bits to the Left

- A left-shift:
  - moves every bit of a binary number to the left
  - adds a 0 in the right-most place
- For example: $1011010_2 = 90_{10}$
  - a left-shift by 1 gives $10110100_2 = 180_{10}$

base 2 base 10

- Left-shifting by 1 doubles the value of a number.
- In Python, we can apply the left-shift operator (<<) to any integer:

```
>>> print(75 << 1)
150
```

Shifting Bits to the Left

- A left-shift:
  - moves every bit of a binary number to the left
  - adds a 0 in the right-most place
- For example: $1011010_2 = 90_{10}$
  - a left-shift by 1 gives $10110100_2 = 180_{10}$

base 2 base 10

- Left-shifting by 1 doubles the value of a number.
- In Python, we can apply the left-shift operator (<<) to any integer:

```
>>> print(75 << 1)
150
```


```
>>> print(5 << 2)
???
```

Shifting Bits to the Left

- A left-shift:
  - moves every bit of a binary number to the left
  - adds a 0 in the right-most place
- For example: $1011010_2 = 90_{10}$
  - a left-shift by 1 gives $10110100_2 = 180_{10}$

base 2 base 10

- Left-shifting by 1 doubles the value of a number.
- In Python, we can apply the left-shift operator (<<) to any integer:

```
>>> print(75 << 1)
150
>>> print(5 << 2)      # left-shift by 2 quadruples!
20
```

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!
- For example: 1011010

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!
- For example: 1011010
  - a right-shift by 1 gives 101101

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!
- For example: $1011010_2 = 90_{10}$
  - a right-shift by 1 gives $101101_2 = 45_{10}$
- Right-shifting by 1 halves the value of a number (using integer division).

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!
- For example: $1011010_2 = 90_{10}$
  - a right-shift by 1 gives $101101_2 = 45_{10}$
- Right-shifting by 1 halves the value of a number (using integer division).
- In Python, we can apply the right-shift operator (>>) to any integer:

```
>>> print(15 >> 1)
7
```

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!
- For example: $1011010_2 = 90_{10}$
  - a right-shift by 1 gives $101101_2 = 45_{10}$
- Right-shifting by 1 halves the value of a number (using integer division).
- In Python, we can apply the right-shift operator (>>) to any integer:

```
>>> print(15 >> 1)
7
>>> print(120 >> 2)
???
```

Shifting Bits to the Right

- A right-shift:
  - moves every bit of a binary number to the right
  - the rightmost bit is lost!
- For example: $1011010_2 = 90_{10}$
  - a right-shift by 1 gives $101101_2 = 45_{10}$
- Right-shifting by 1 halves the value of a number (using integer division).
- In Python, we can apply the right-shift operator (>>) to any integer:

```
>>> print(15 >> 1)
7
>>> print(120 >> 2)    # right-shift by 2 quarters!
30
```

Recall: Decimal to Binary (On Paper)

$$
\begin{aligned}
90 &= 64 + 26 \\
&= 64 + 16 + 10 \\
&= 64 + 16 + 8 + 2 \\
&= 2^6 + 2^4 + 2^3 + 2^1 \\
&= 1011010
\end{aligned}
$$

- This is a **left-to-right** conversion.
  - we begin by determining the leftmost digit
- The first step is tricky to perform computationally, because we need to determine the largest power.

Decimal to Binary: Right-to-Left

- We can use a **right-to-left** approach instead.
- For example: let's convert 139 to binary:

```
139   =   ???????1
```

The rightmost bit must be 1. Why?

Decimal to Binary: Right-to-Left

- We can use a **right-to-left** approach instead.
- For example: let's convert 139 to binary:

```
139   =   ???????1
```

The rightmost bit
must be 1. Why?
because 139 is odd

Decimal to Binary: Right-to-Left

- We can use a **right-to-left** approach instead.
- For example: let's convert 139 to binary:


```
139   =   ???????1
```

What the remaining bits
without the rightmost bit,
would represent?

The rightmost bit
must be 1. Why?
because 139 is odd

Decimal to Binary: Right-to-Left

- We can use a **right-to-left** approach instead.
- For example: let's convert 139 to binary:

`139 = ???????1`

The rightmost bit must be 1. Why?
because 139 is odd

What the remaining bits without the rightmost bit, would represent?
139 >> 1 → 69

Decimal to Binary: Right-to-Left

- We can use a **right-to-left** approach instead.
- For example: let's convert 139 to binary:

`139 = ???????1`

The rightmost bit must be 1. Why?
because 139 is odd

What the remaining bits without the rightmost bit, would represent?
139 >> 1 → 69

- To convert 139: recursively convert 69,
  then put a 1 at the end!

Decimal to Binary: Right-to-Left (cont.)

139 = ???????1

Decimal to Binary: Right-to-Left (cont.)

				139	=	???????1
139	>>	1	→	69	=	??????1
69	>>	1	→	34	=	?????0
34	>>	1	→	17	=	????1
17	>>	1	→	8	=	???0
8	>>	1	→	4	=	??0
4	>>	1	→	2	=	?0
2	>>	1	→	1	=	1
				139	=	10001011

`dec_to_bin()` Function

- `dec_to_bin(n)`
  - takes an integer `n`
  - should return a *string* representation of `n`'s binary representation

```
>>> dec_to_bin(139)
'10001011'
>>> dec_to_bin(13)
'1101'
```

How `dec_to_bin()` Should Work...

`dec_to_bin(13)`

How dec_to_bin() Should Work...

dec_to_bin(13)

dec_to_bin(6) + '1'

How dec_to_bin() Should Work...

dec_to_bin(13)

dec_to_bin(6) + '1'

dec_to_bin(3) + '0'

How `dec_to_bin()` Should Work...

```
            dec_to_bin(13)
      dec_to_bin(6) + '1'
   dec_to_bin(3) + '0'
dec_to_bin(1) + '1'
```

How `dec_to_bin()` Should Work...

```
            dec_to_bin(13)
      dec_to_bin(6) + '1'
   dec_to_bin(3) + '0'
dec_to_bin(1) + '1'
    '1'
```

How dec_to_bin() Should Work...

```
dec_to_bin(13)
dec_to_bin(6) + '1'
dec_to_bin(3) + '0'
'1'           + '1'
```

How dec_to_bin() Should Work...

```
dec_to_bin(13)
dec_to_bin(6) + '1'
dec_to_bin(3) + '0'
'1'           + '1'  → '11'
```

How dec_to_bin() Should Work...

dec_to_bin(13)

dec_to_bin(6) + '1'

'11' + '0'

How dec_to_bin() Should Work...

dec_to_bin(13)

dec_to_bin(6) + '1'

'11' + '0' → '110'

How `dec_to_bin()` Should Work...

$$\underbrace{\texttt{'110'} \quad + \texttt{'1'}}_{}$$

`dec_to_bin(13)`

`'110'`   `+ '1'`

How `dec_to_bin()` Should Work...

`dec_to_bin(13)`

`'110'`   `+ '1'` → `'1101'`

How `dec_to_bin()` Should Work...

`dec_to_bin(13)` → '1101'

Binary to Decimal: Right-to-Left

- Here again, we can use a **right-to-left** approach.
- For example:

```
'1101' = ?
```

If we knew the
decimal value
of these bits,
how could we use it?

What should we do
with the rightmost bit?

Binary to Decimal: Right-to-Left

- Here again, we can use a **right-to-left** approach.
- For example:

`'1101' = ?`

If we knew the decimal value of these bits, how could we use it?

What should we do with the rightmost bit?

- To convert `'1101'`:
  - recursively convert `'110'`
  - double the result
  - add 1 for the rightmost bit

Binary to Decimal: Right-to-Left

- Here again, we can use a **right-to-left** approach.
- For example:

`'1101' = 13`

If we knew the decimal value of these bits, how could we use it?

What should we do with the rightmost bit?

- To convert `'1101'`: 13
  - recursively convert `'110'` : 6
  - double the result: 2 * 6 = 12
  - add 1 for the rightmost bit: 12 + 1 = 13

How `bin_to_dec()` Should Work...

```
bin_to_dec('1101')
```

How `bin_to_dec()` Should Work...

```
bin_to_dec('1101')
2 * bin_to_dec('110') + 1
```

How `bin_to_dec()` Should Work...

```
        bin_to_dec('1101')
2 * bin_to_dec('110') + 1
2 * bin_to_dec('11') + 0
```

How `bin_to_dec()` Should Work...

```
        bin_to_dec('1101')
2 * bin_to_dec('110') + 1
2 * bin_to_dec('11') + 0
2 * bin_to_dec('1') + 1
```

How bin_to_dec() Should Work...

```
bin_to_dec('1101')
2 * bin_to_dec('110') + 1
2 * bin_to_dec('11') + 0
2 * bin_to_dec('1') + 1
1
```

How bin_to_dec() Should Work...

```
bin_to_dec('1101')
2 * bin_to_dec('110') + 1
2 * bin_to_dec('11') + 0
2 *        1        + 1
```

How `bin_to_dec()` Should Work...

```
        bin_to_dec('1101')
   2 * bin_to_dec('110') + 1
  2 * bin_to_dec('11') + 0
 2 *      1       + 1    → 3
```

How `bin_to_dec()` Should Work...

```
        bin_to_dec('1101')
   2 * bin_to_dec('110') + 1
  2 *      3       + 0
```

How bin_to_dec() Should Work...

```
       bin_to_dec('1101')
2 * bin_to_dec('110') + 1
2 *       3       + 0   → 6
```

How bin_to_dec() Should Work...

```
       bin_to_dec('1101')
2 *      6        + 1
```

How `bin_to_dec()` Should Work...

$$\underbrace{\texttt{2 * 6 + 1}}_{}$$

```
          bin_to_dec('1101')
2 *       6                  + 1   → 13
```

How `bin_to_dec()` Should Work...

```
bin_to_dec('1101')  → 13
```

Binary Addition Fundamentals

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 10
- 1 + 1 + 1 = 11

Adding Decimal Numbers

```
      1 1
      12537
+      9272
-----------
      21809
```

Adding Binary Numbers

$$
\begin{array}{r}
01110 \\
+ \quad 11100 \\
\hline
\end{array}
$$

Adding Binary Numbers

$$
\begin{array}{r}
01110 \\
+ \quad 11100 \\
\hline
0
\end{array}
$$

$0 + 0 = 0$

Adding Binary Numbers

$$
\begin{array}{rr}
 & 01110 \\
+ & 11100 \\
\hline
 & 10
\end{array}
\qquad 1 + 0 = 1
$$

Adding Binary Numbers

$$
\begin{array}{rr}
 & \overset{1}{0}\,1110 \\
+ & 11100 \\
\hline
 & 010
\end{array}
\qquad 1 + 1 = 10
$$

Adding Binary Numbers

```
      1 1
     01110
+    11100
----------
      1010
```

1 + 1 + 1 = 11

Adding Binary Numbers

```
    1 1 1
     01110
+    11100
----------
     01010
```

1 + 0 + 1 = 10

Adding Binary Numbers

```
  1 1 1
    01110
+   11100
---------
   101010
```

Adding Binary Numbers

```
  1 1 1
    01110
+   11100
---------
   101010
```

Add these two binary numbers ***WITHOUT*** converting to decimal!

$$\begin{array}{r} 101101 \\ +\quad 1110 \\ \hline \end{array}$$

Hint:

$$\begin{array}{r} \overset{1}{5}29 \\ +\ 742 \\ \hline 1271 \end{array}$$

Do you remember this algorithm? It's the same!

A. 100011

B. 111011

C. 110111

D. 110011

E. none of these

Add these two binary numbers ***WITHOUT*** converting to decimal!

$$\begin{array}{r} 101101 \\ +\quad 1110 \\ \hline \end{array}$$

Hint:

$$\begin{array}{r} \overset{1}{5}29 \\ +\ 742 \\ \hline 1271 \end{array}$$

Do you remember this algorithm? It's the same!

A. 100011

B. 111011

C. 110111

D. 110011

E. none of these

Add these two binary numbers
WITHOUT converting to decimal!

$$\begin{array}{r} 101101 \\ +\quad 1110 \\ \hline 1 \end{array}$$

$$\begin{array}{r} {}^{1} \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

Hint:
Do you remember this algorithm? It's the same!

A. 100011
B. 111011
C. 110111
D. 110011
E. none of these

Add these two binary numbers
WITHOUT converting to decimal!

$$\begin{array}{r} 101101 \\ +\quad 1110 \\ \hline 11 \end{array}$$

$$\begin{array}{r} {}^{1} \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

Hint:
Do you remember this algorithm? It's the same!

A. 100011
B. 111011
C. 110111
D. 110011
E. none of these

Add these two binary numbers ***WITHOUT*** converting to decimal!

$$\begin{array}{r} {\scriptstyle 1} \\ 101101 \\ +\quad 1110 \\ \hline 011 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1} \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

Hint:
Do you remember this algorithm? It's the same!

A. 100011

B. **111011**

C. 110111

D. 110011

E. none of these

Add these two binary numbers ***WITHOUT*** converting to decimal!

$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 101101 \\ +\quad 1110 \\ \hline 1011 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1} \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

Hint:
Do you remember this algorithm? It's the same!

A. 100011

B. **111011**

C. 110111

D. 110011

E. none of these

Add these two binary numbers
WITHOUT converting to decimal!

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ } \\ 101101 \\ +\quad 1110 \\ \hline 11011 \end{array}$$

Hint: Do you remember this algorithm? It's the same!

$$\begin{array}{r} {\scriptstyle 1\ \ } \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

A. 100011

B. **111011**

C. 110111

D. 110011

E. none of these

Add these two binary numbers
WITHOUT converting to decimal!

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ } \\ 101101 \\ +\quad 1110 \\ \hline 111011 \end{array}$$

Hint: Do you remember this algorithm? It's the same!

$$\begin{array}{r} {\scriptstyle 1\ \ } \\ 529 \\ +\ 742 \\ \hline 1271 \end{array}$$

A. 100011

B. **111011**

C. 110111

D. 110011

E. none of these

Multiply these binary numbers
WITHOUT converting to decimal!

```
   101101
*    1110
---------
```

Hint: Do you remember this algorithm? It's the same!

```
   529
*   42
------
  1058
+2116
------
 22218
```

Multiply these binary numbers
WITHOUT converting to decimal!

```
   101101
*    1110
---------
   000000
```

Hint: Do you remember this algorithm? It's the same!

```
   529
*   42
------
  1058
+2116
------
 22218
```

Multiply these binary numbers ***WITHOUT*** converting to decimal!

```
   101101
*    1110
---------
   000000
  1011010
```

Hint: Do you remember this algorithm? It's the same!

```
   529
*   42
------
  1058
+2116
------
 22218
```

Multiply these binary numbers ***WITHOUT*** converting to decimal!

```
   101101
*    1110
---------
   000000
  1011010
 10110100
```

Hint: Do you remember this algorithm? It's the same!


```
   529
*   42
------
  1058
+2116
------
 22218
```

Multiply these binary numbers
WITHOUT converting to decimal!

```
   101101
*    1110
---------
   000000
  1011010
 10110100
101101000
```


Multiply these binary numbers
WITHOUT converting to decimal!

```
    101101
*     1110
----------
    000000
   1011010
  10110100
+101101000
----------
1001110110
```


```
   529
*   42
------
  1058
+2116
------
 22218
```
Hint:
Do you remember this algorithm? It's the same!

It's All Bits!

- Another example: text

`'terriers'`

`0111010001100101011100100111001001101001011001010111001001110011`

8 ASCII characters, 8 bits each → 64 bits

It's All Bits!

- Another example: text

`'terriers'`

`0111010001100101011100100111001001101001011001010111001001110011`

8 ASCII characters, 8 bits each → 64 bits

- *All* types of data are represented in binary.
    - images, sounds, movies, floating-point numbers, etc...

- ***All computation*** involves manipulating bits!