BINARY ARITHMETIC

David Kauchak
CS 51 – Spring 2026

BBICS

Get connected with BBICS!

Email list

Instagram

Mentee Application

Administrative

Assignment 1 graded

Assignment 2

Academic honesty

- Working with other students
- AI tools

Adding numbers base 10

Add: 456 and 735

Adding numbers base 10

$$
\begin{array}{r} 456 \\ +\ 735 \\ \hline ? \end{array}
$$

Adding numbers base 10

$$
\begin{array}{r}
{\scriptstyle 1\ 0\ 1} \\
456 \\
+\ 735 \\
\hline
1191
\end{array}
$$

Adding numbers base 5

Add: 223_5 and 414_5

Adding numbers base 5

$$\begin{array}{r} 223_5 \\ +\ 414_5 \\ \hline ? \end{array}$$

Adding numbers base 5

$$
\begin{array}{r}
{\scriptstyle 1\ 0\ 1} \\
223_5 \\
+\ \ 414_5 \\
\hline
1142_5
\end{array}
$$

Adding numbers base 5

$$\begin{array}{r} {\scriptstyle 1\ 0\ 1\ \ } \\ 223_5 \\ +\ \ 414_5 \\ \hline 1142_5 \end{array}$$

Is this correct?

What numbers are these?

Adding numbers base 5

$$
\begin{array}{r}
{\scriptstyle 1\,0\,1\ \ } \\
223_5 \\
+\ \ 414_5 \\
\hline
1142_5
\end{array}
$$

2*25 + 2*5 + 3

4*25 + 1*5 + 4

1*125 + 1*25+4*5+2

Adding numbers base 5

$$
\begin{array}{rr}
 {\scriptstyle 1\,0\,1\;\;} & \\
 223_5 & 63_{10} \\
 +\ \ 414_5 & 109_{10} \\
 \hline
 1142_5 & 172_{10}
\end{array}
$$

Adding numbers base 2

Add: 0001_2 and 0101_2

Adding numbers base 2

$$
\begin{array}{r}
0001_2 \\
+\ 0101_2 \\
\hline
?
\end{array}
$$

Adding numbers base 2

$$\begin{array}{r} {\scriptstyle 0\ \ 0\ \ 1\ \ \ } \\ 0001_2 \\ +\ 0101_2 \\ \hline 0110_2 \end{array}$$

Adding numbers base 2

$$
\begin{array}{r}
{\scriptstyle 0\ 0\ 1\ \ } \\
0001_2 \\
+\ 0101_2 \\
\hline
0110_2
\end{array}
$$

Is this correct?

What numbers are these?

Adding numbers base 2

$$
\begin{array}{rl}
 & {\scriptstyle 0\;\;0\;\;1} \\
 & 0001_2 \qquad 1_{10} \\
+ & 0101_2 \qquad 5_{10} \\
\hline
 & 0110_2 \qquad 6_{10}
\end{array}
$$

PRACTICE TIME - Addition

Compute the following sums

- $0111_2 + 0101_2$
- $9B9_{16} + 38_{16}$

ANSWER - Addition

Compute the following sums

- $0111_2 + 0101_2$
- $9B9_{16} + 38_{16}$

$0111_2 + 0101_2$

0 1 1 1

0 1 1 1

+ 0 1 0 1

1 1 0 0

← carry →

$9B3_{16} + 38_{16}$

0 0 1

9 B 9

+ 3 8

9 F 1

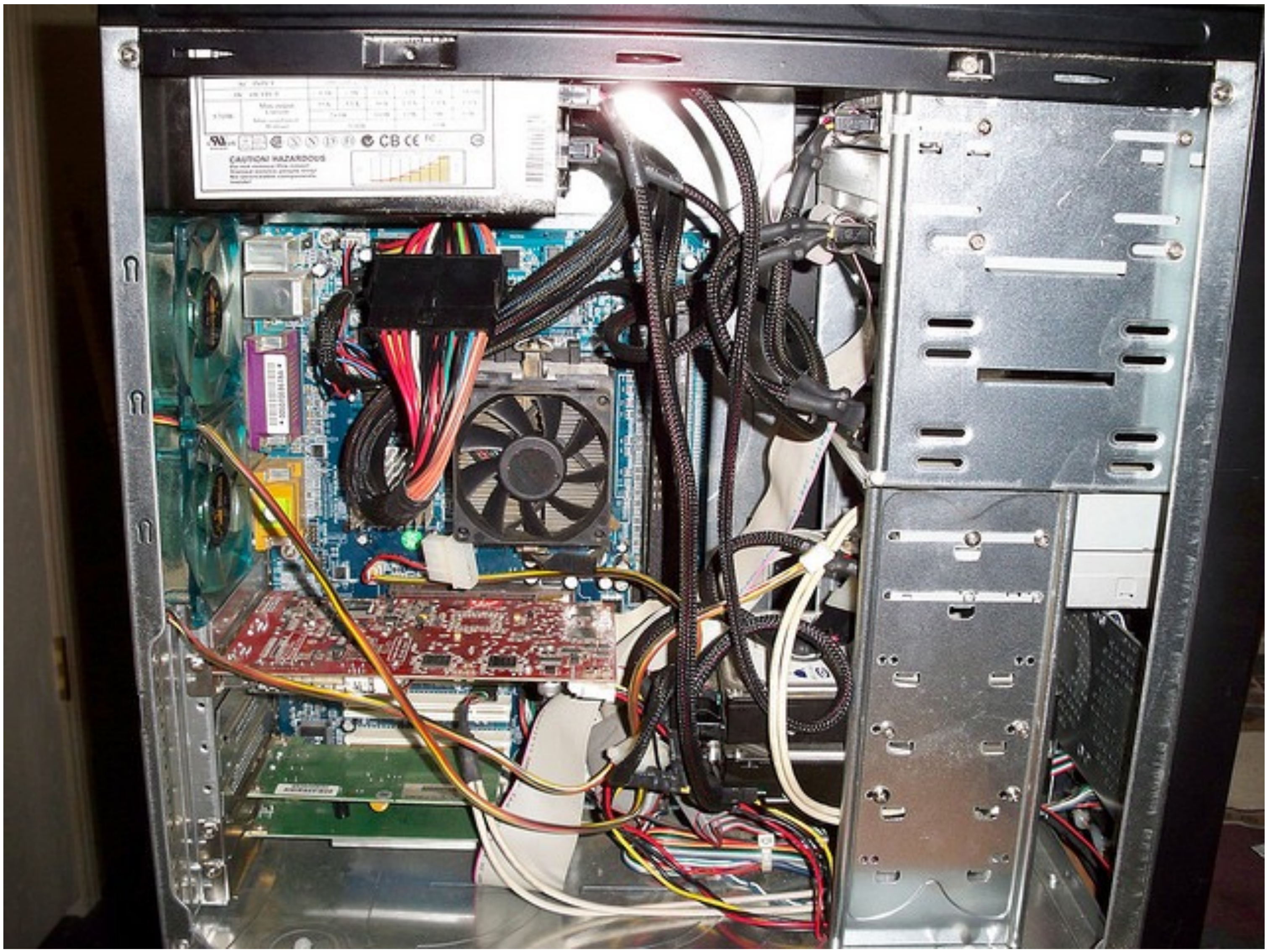
Adding numbers base 2

$$
\begin{array}{r}
1001_2 \\
+\ 1101_2 \\
\hline
?
\end{array}
$$

Adding numbers base 2

$$\begin{array}{r} {}^{1\;0\;0\;1} \\ 1001_2 \\ +\ 1101_2 \\ \hline 10110_2 \end{array}$$

Computer internals



Computer internals simplified

CPU

RAM

What does it stand for?

What does it do?

Computer internals simplified

CPU

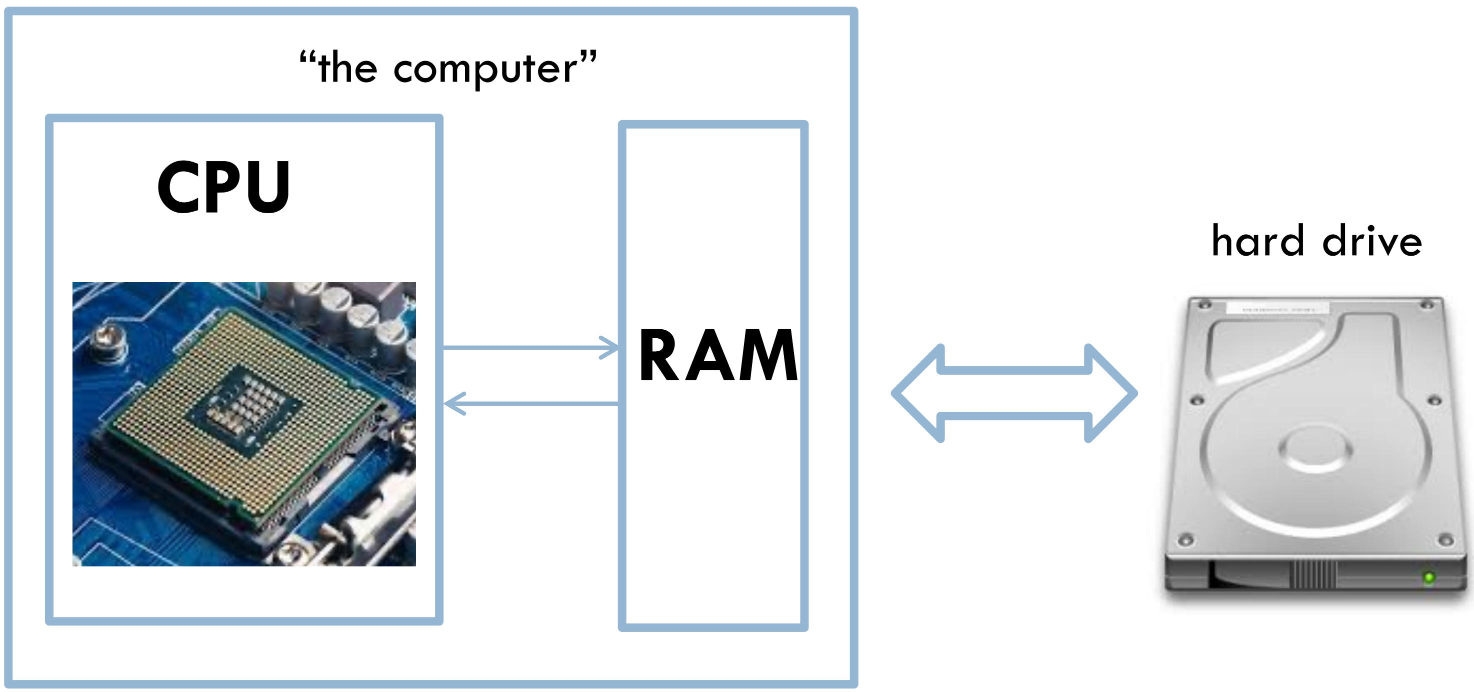
RAM

(**R**andom **A**ccess **M**emory,
aka "*memory*"
or "*main memory*")

Does all the work!

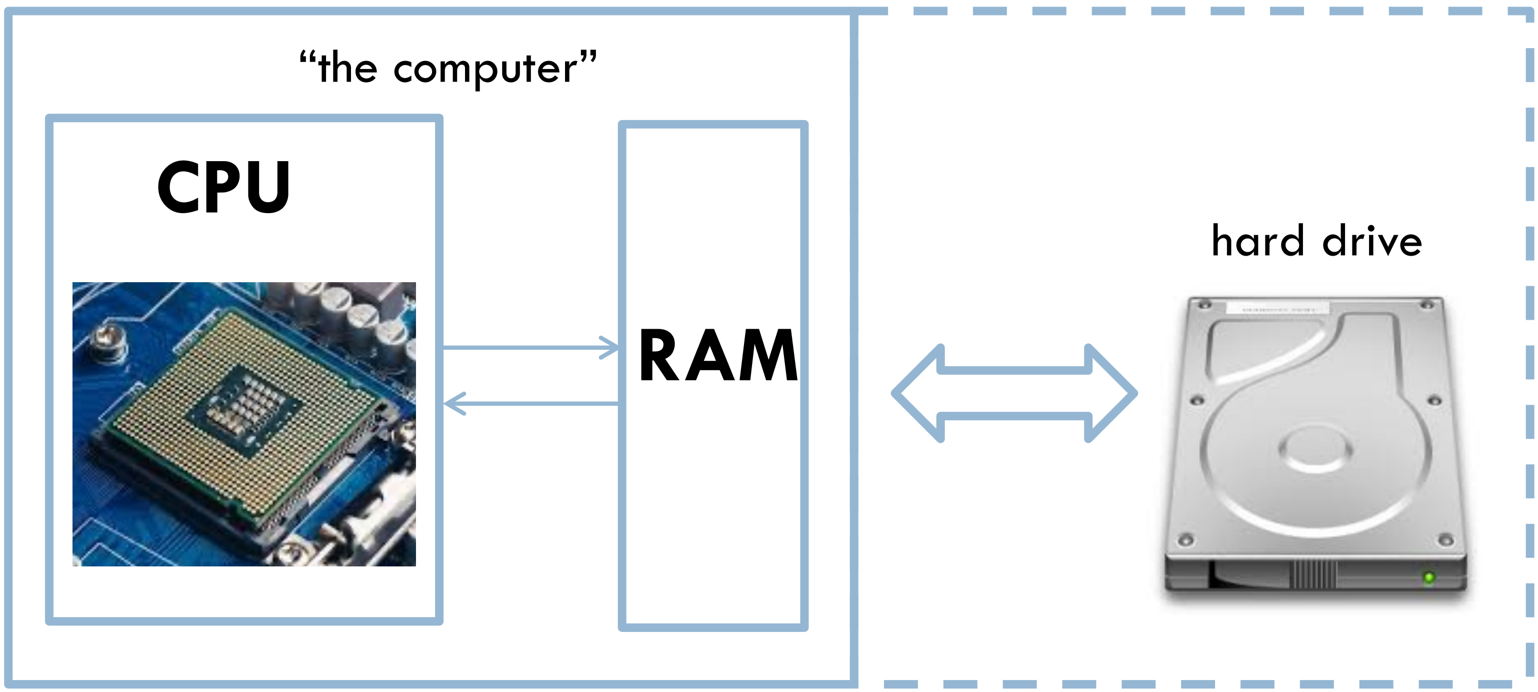
Temporary storage

Computer internals simplified



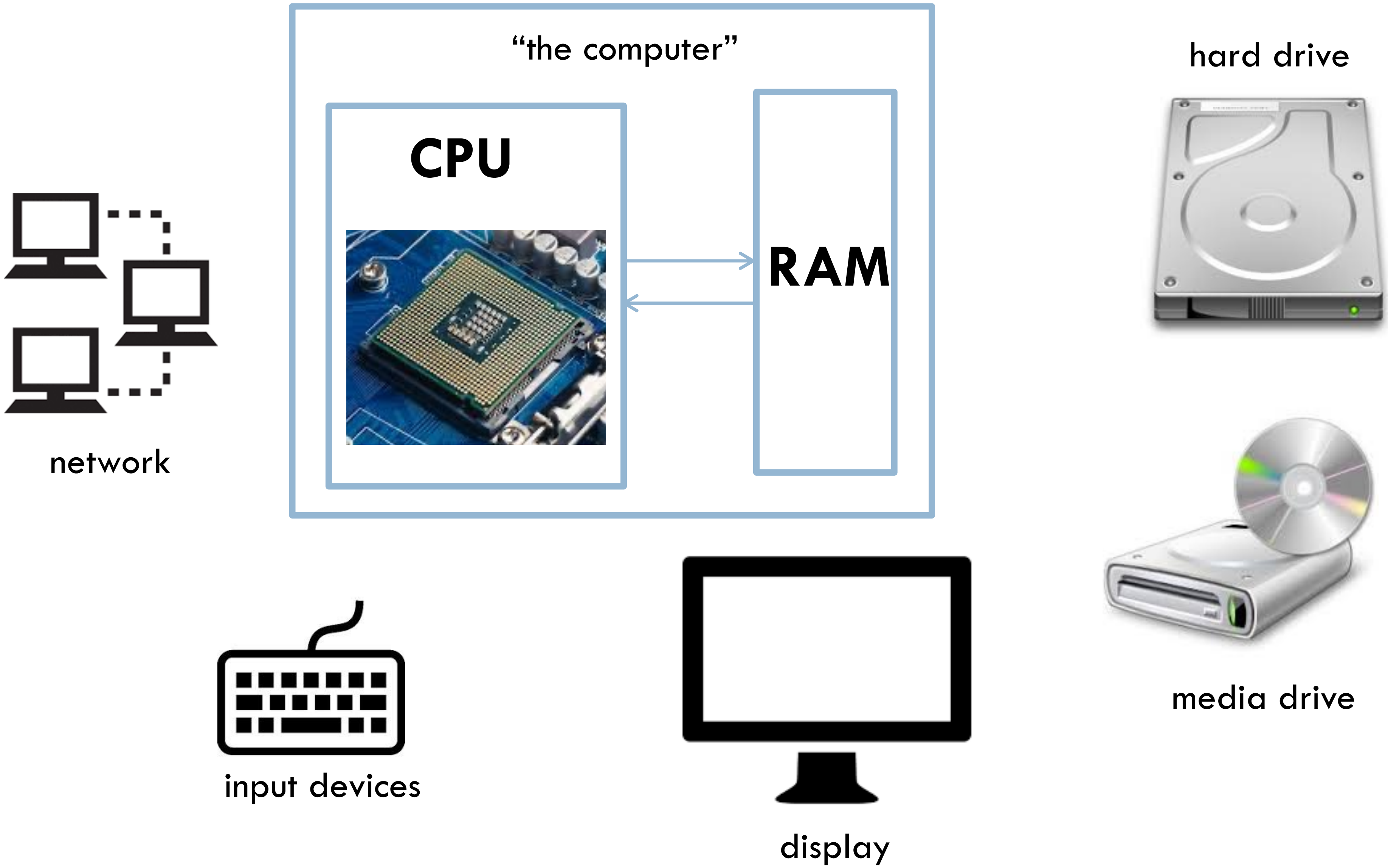
Why do we need a hard drive?

Computer internals simplified



- Persistent memory
- RAM only stores data while it has power

Computer simplified

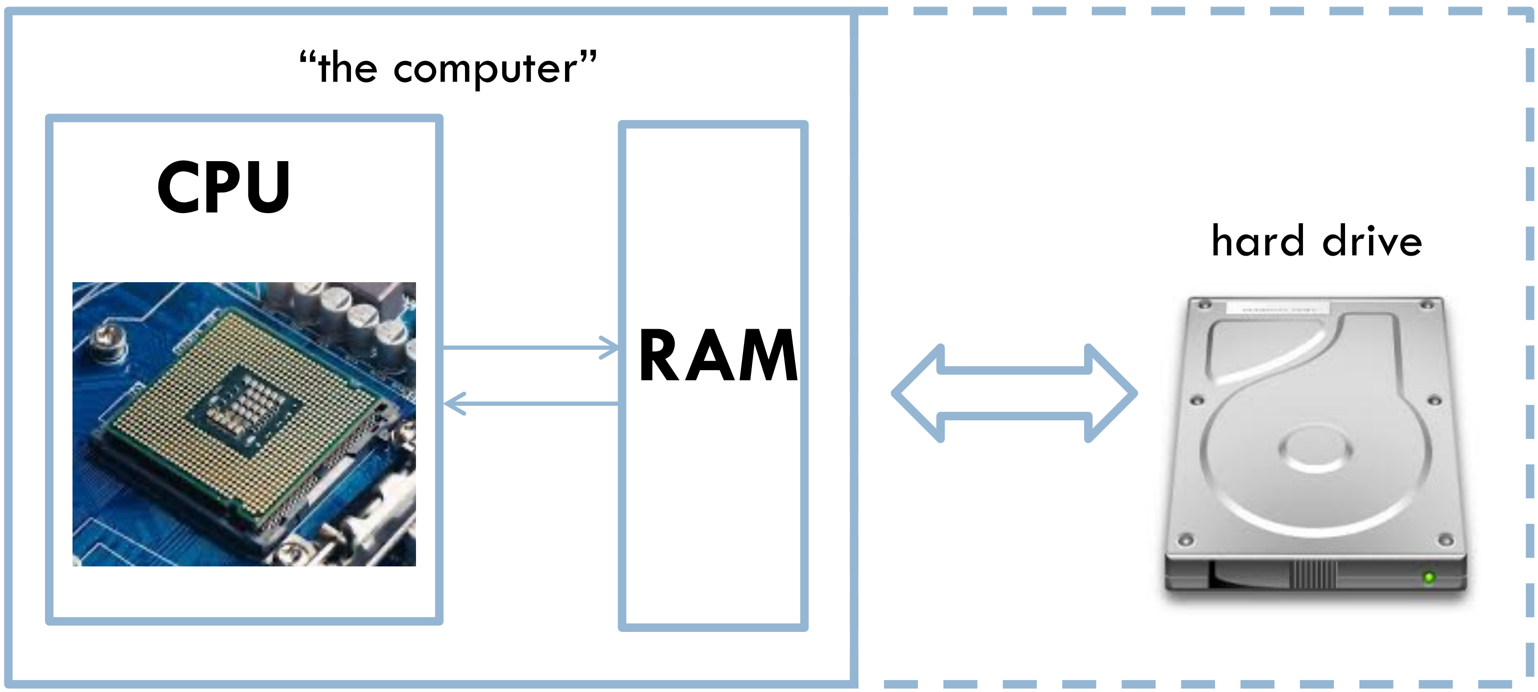


Fixed length numbers

0100100100101001000111100010001010

This is more than one number. What are the numbers?

Memory



Both RAM and the hard drive stores a sequence of bits

To simplify hardware, these bits are grouped into fixed length chunks called "words"

Fixed length numbers

0100 1001 0011 0100 1001 0111 1000 1000 1010

For example, we could have a 4-bit word

Fixed length numbers

0100 1001 0011 0100 1001 0111 1000 1000 1010

For example, we could have a 4-bit word

Most computers these days have 64-bit words

Fixed length words

With 4 bits, how many values/numbers can we represent?

Fixed length words

With 4 bits, how many values/numbers can we represent?

2^4 = 16 values from 0000 up to 1111

Fixed length words

1 0 0 1

$$
\begin{array}{r}
1001_2 \\
+\ 1101_2 \\
\hline
10110_2
\end{array}
$$

What happens here?

Overflow

$$\begin{array}{rr} & {}^{1\ 0\ 0\ 1} \\ & 1001_2 \\ + & 1101_2 \\ \hline & 10110_2 \end{array} \qquad \begin{array}{r} 9_{10} \\ 13_{10} \\ 22_{10} \end{array}$$

Overflow: the result is too large to represent

Overflow

$$
\begin{array}{r}
{\scriptstyle 1\ 0\ 0\ 1}\quad \\
1001_2 \\
+\ 1101_2 \\
\hline
10110_2
\end{array}
$$

How do we detect overflow?

Overflow

1 0 0 1

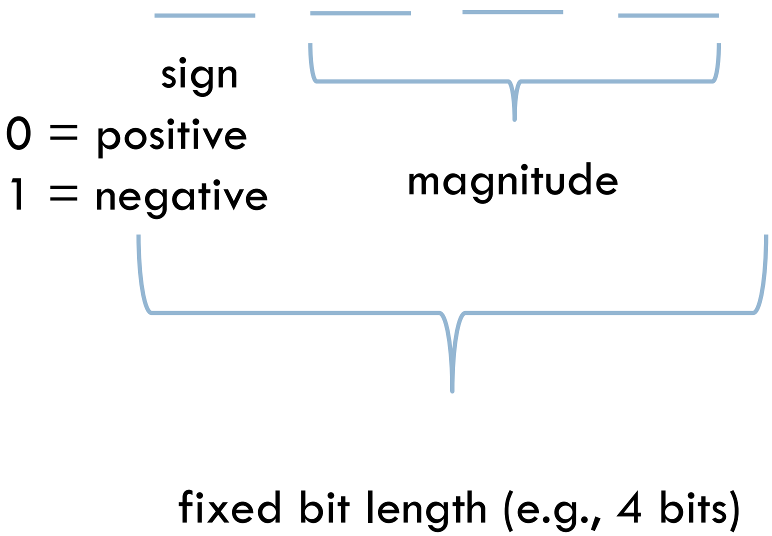
$$
\begin{array}{r}
1001_2 \\
+\ 1101_2 \\
\hline
10110_2
\end{array}
$$

Check the carry bit for the most significant bit

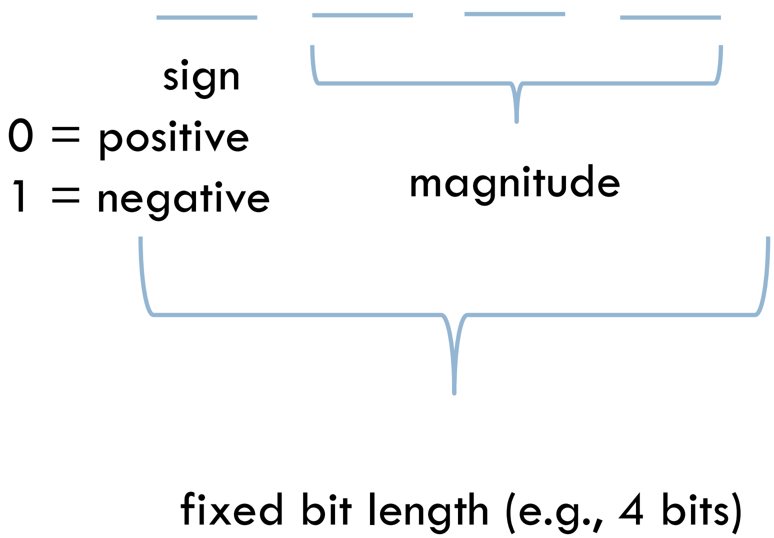
Binary numbers revisited

What is -6_{10} in binary?

One option: sign/magnitude

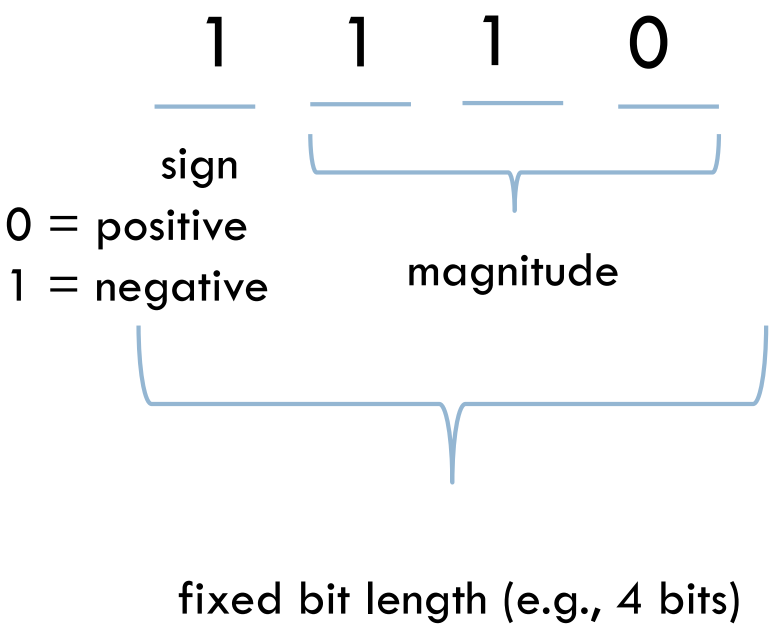


One option: sign/magnitude



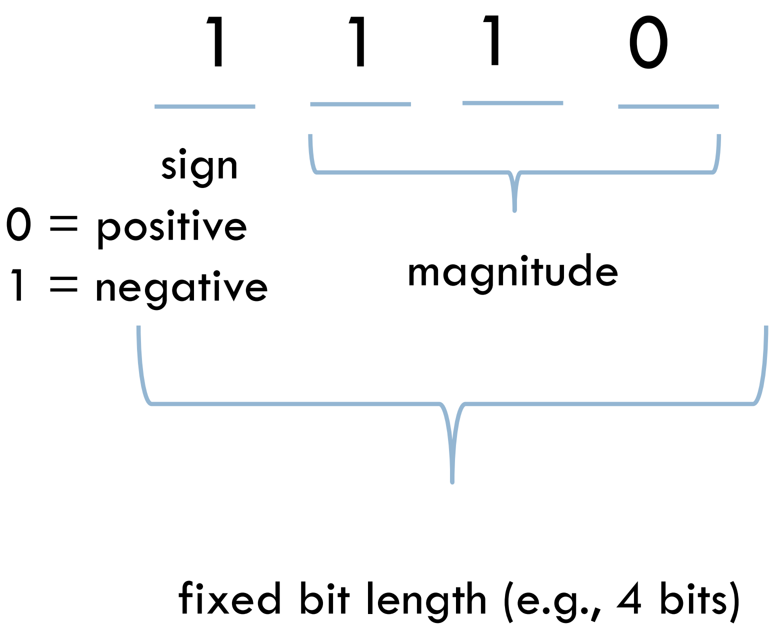
What is -6_{10} in sign/magnitude?

One option: sign/magnitude



What is -6_{10} in sign/magnitude?

One option: sign/magnitude



What are the range of values?

One option: sign/magnitude

1 1 1 0

sign
0 = positive
1 = negative

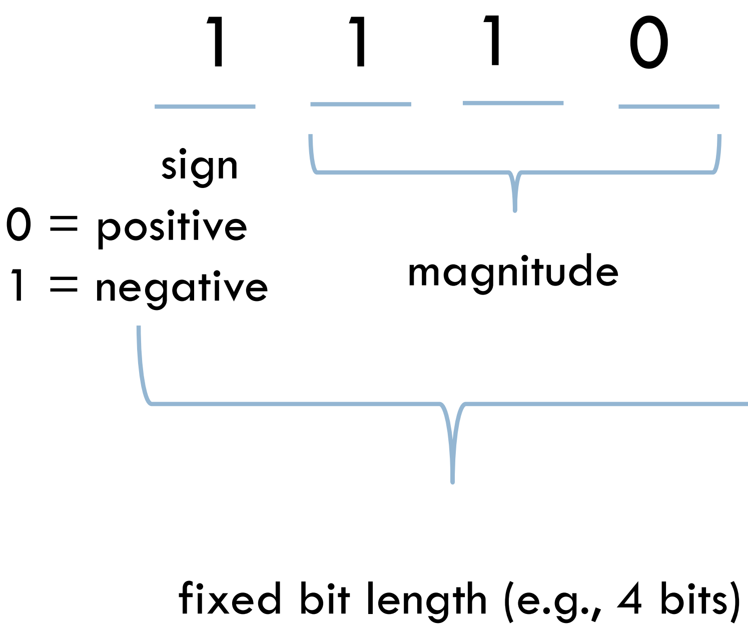
magnitude

fixed bit length (e.g., 4 bits)

What are the range of values?

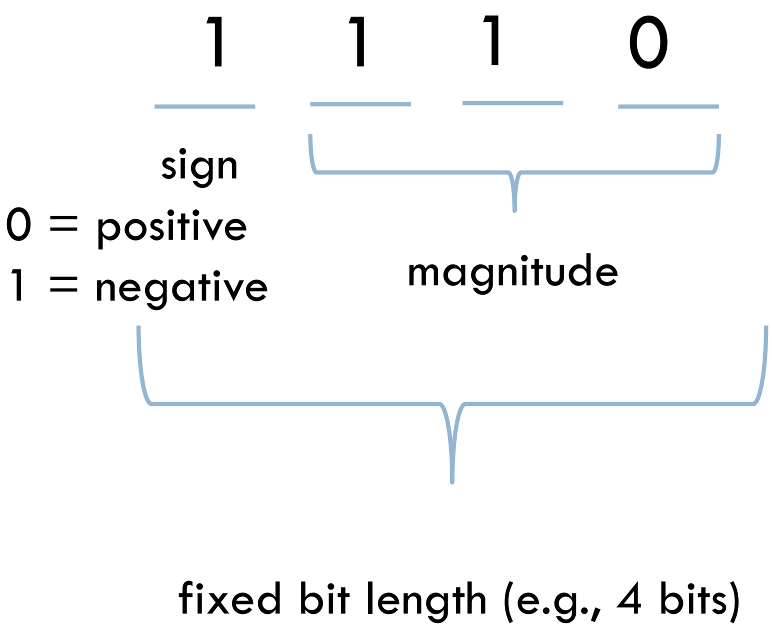
1111 = -7 to
0111 = 7

One option: sign/magnitude



Any problems?

One option: sign/magnitude



What are the range of values?

1000 = 0000

two zeros

hardware-wise, can be more difficult

Another option: twos complement

For a number with *n* digits the high order bit represents -2^{n-1}

unsigned	2^3	2^2	2^1	2^0
signed (twos complement)	-2^3	2^2	2^1	2^0

Twos complement

What number is it?

unsigned	1	0	0	1	9
	2^3	2^2	2^1	2^0	
signed (twos complement)	1	0	0	1	-7
	-2^3	2^2	2^1	2^0	

Twos complement

What number is it?

unsigned	1	1	1	1	15
	2^3	2^2	2^1	2^0	
signed (twos complement)	1	1	1	1	-1
	-2^3	2^2	2^1	2^0	

Twos complement

What number is it?

	2^3	2^2	2^1	2^0	
unsigned	1	1	0	0	12

	-2^3	2^2	2^1	2^0	
signed (twos complement)	1	1	0	0	-4

Twos complement

How many numbers can we represent with each approach using 4 bits?

16 (2^4) numbers, 0000, 0001, …., 1111

Doesn't matter the representation!

unsigned	2^3	2^2	2^1	2^0
signed (twos complement)	-2^3	2^2	2^1	2^0

Twos complement

How many numbers can we represent with each approach using 32 bits?

$2^{32} \approx$ 4 billion numbers

unsigned	2^3	2^2	2^1	2^0
signed (twos complement)	-2^3	2^2	2^1	2^0

Twos complement

What is the range of numbers that we can represent for each approach with 4 bits?

unsigned: 0, 1, … 15
signed: -8, -7, …, 7

unsigned	2^3	2^2	2^1	2^0
signed (twos complement)	-2^3	2^2	2^1	2^0

binary representation	unsigned	
0000	0	
0001	1	
0010	?	
0011		
0100		
0101		
0110		
0111		
1000		
1001		
1010		
1011		
1100		
1101		
1110		
1111		

binary representation	unsigned	twos complement
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	

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	?
1001	9	
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

binary representation	unsigned	twos complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	?
1010	10	
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

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

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

How can you tell if a number is negative?

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

High order bit!

A two's complement trick

You can also calculate the value of a negative number represented as twos complement as follows:

- Flip all of the bits (0 → 1 and 1 → 0)
- Add 1
- The resulting number is the magnitude of the original negative number

flip the bits: 1101 → 0010

add 1: 0010 → 0011

0011 → -3

A two's complement trick

You can also calculate the value of a negative number represented as twos complement as follows:

- Flip all of the bits (0 → 1 and 1→ 0)
- Add 1
- The resulting number is the magnitude of the original negative number

flip the bits add 1

1110 → 0001 → 0010 → -2

Addition with 4-bit *twos complement* numbers

$$
\begin{array}{r} 0001_2 \\ +\ 0101_2 \\ \hline ? \end{array}
$$

Addition with 4-bit *twos complement* numbers

$$\begin{array}{r} {}^{0\;0\;1} \\ 0001_2 \\ +\ 0101_2 \\ \hline 0110_2 \end{array}$$

Addition with 4-bit *twos complement* numbers

```
  0110
+ 0101
------
   ?
```

(Note: I'm going to stop writing the base 2 ☺)

Addition with 4-bit *twos complement* numbers

```
   1
   0110
 + 0101
 ------
   1011?
```

Addition with 4-bit *twos complement* numbers

	$\overset{1}{0}110$	6
+	0101	5
	1011?	-5? (11 unsigned)

Overflow! We cannot represent this number (it's too large)

Addition with 4-bit *twos complement* numbers

```
  0110
+ 1101
------
   ?
```

Addition with 4-bit *twos complement* numbers

```
 1  1
  0110
+ 1101
------
  0011
```

Addition with 4-bit *twos complement* numbers

ignore the last carry

```
  1  1
   0110     6
 + 1101    -3
 ------
   0011     3
```

Overflow in twos complement

How do you know that overflow has occurred with twos complement numbers?

Overflow in twos complement

How do you know that overflow has occurred with twos complement numbers?

Can only happen when adding two numbers of the same sign

Add the numbers and discard the carry

Check the sign of the resulting number: if it differs than the number added = overflow

Subtraction

Ideas?

Subtraction

Negate the 2nd number (flip the bits and add 1)

Add them!

PRACTICE TIME – Subtracting two's complements

Calculate $3_{10} - 5_{10}$ using 4-bit two's complement numbers.

ANSWER – Subtracting two's complements

Calculate $3_{10} - 5_{10}$ using 4-bit two's complement numbers.

3_{10} is 0011_2

Take the two's complement of -$5_{10} = 1011_2$:

- $5_{10} = 0101_2$
- Invert 1010_2
- Add 1: 1011_2

Add them: $0011_2 + 1011_2 = 1110_2 = -2_{10}$

- Invert and then 1: $0001 + 1 = 0010 = 2$
- So $1110_2 = -2_{10}$

Shifting: variable length numbers

Shifting shifts the binary representation of the number right or left

Shifting: variable length numbers

Shifting shifts the binary representation of the number right or left

37 >> 2 ?

number to be shifted

right shift

number of positions to shift

Shifting: variable length numbers

Shifting shifts the binary representation of the number right or left

37 >> 2

37 → 100101 → 1001 → 9

number in binary

shift right two positions
(discard bits shifting off)

decimal form

Shifting: variable length numbers

Shifting shifts the binary representation of the number right or left

37 >> 3 ?

Shifting: variable length numbers

Shifting shifts the binary representation of the number right or left

37 >> 3

37 → 100101 → 100 → 4

number in binary

shift right three positions (discard bits shifting off)

decimal form

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

37 >> 2

What is 37 as an 8-bit binary number?

37 → 00100101

pad with 0s

number in binary

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

37 >> 2

How do we fill in the leftmost bits?

37 → 00100101 →

number in binary

shift right two positions (discard bits shifting off)

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

37 >> 2

How do we fill in the leftmost bits?

37 → 00100101 → 00001001

number in binary

shift right two positions (discard bits shifting off)

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

37 >> 2

37 → 00100101 → 00001001 → 9

number in binary

shift right two positions
(discard away bits shifting off)

decimal form

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

?

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

15 → ?

number in binary

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

15 → 00001111

number in binary

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

15 → 00001111 → ?

number in binary

shift left two positions (discard bits shifting off)

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

15 → 00001111 → 001111??

number in binary

shift left two positions
(discard bits shifting off)

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

15 → 00001111 → 00111100

number in binary

shift left two positions
(discard bits shifting off)

Shifting 8-bit numbers

Shifting shifts the binary representation of the number right or left

15 << 2

15 → 00001111 → 00111100 → 60

number in binary

shift left two positions (discard bits shifting off)

decimal form

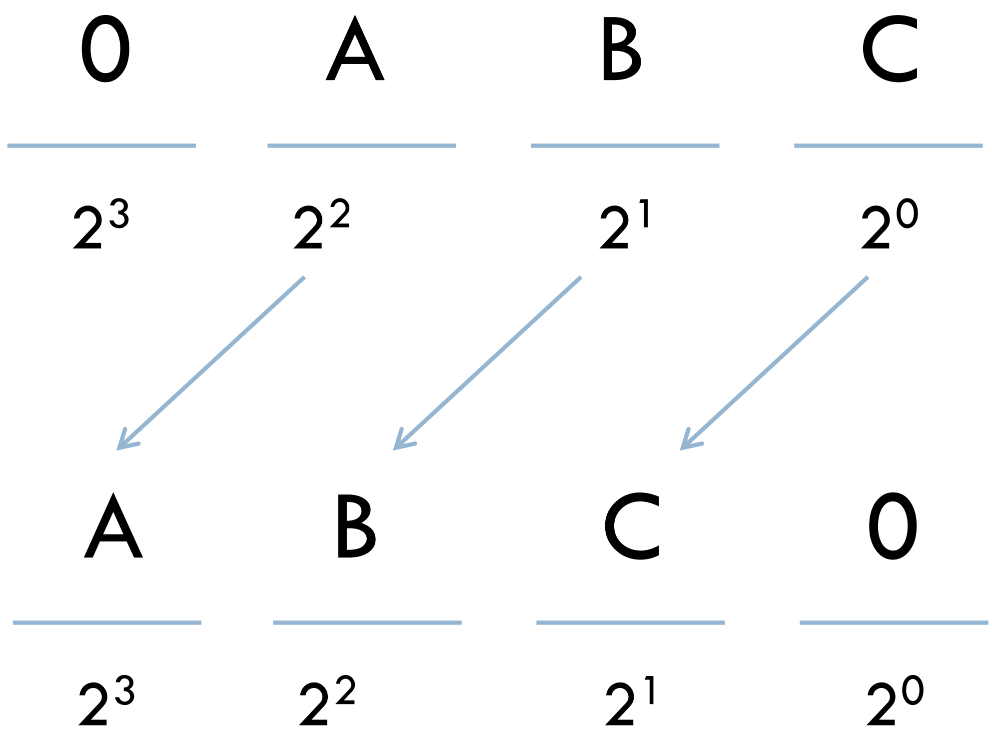
Shifting mathematically: unsigned

What does **left** shifting by one position do mathematically?

0	A	B	C
2^3	2^2	2^1	2^0

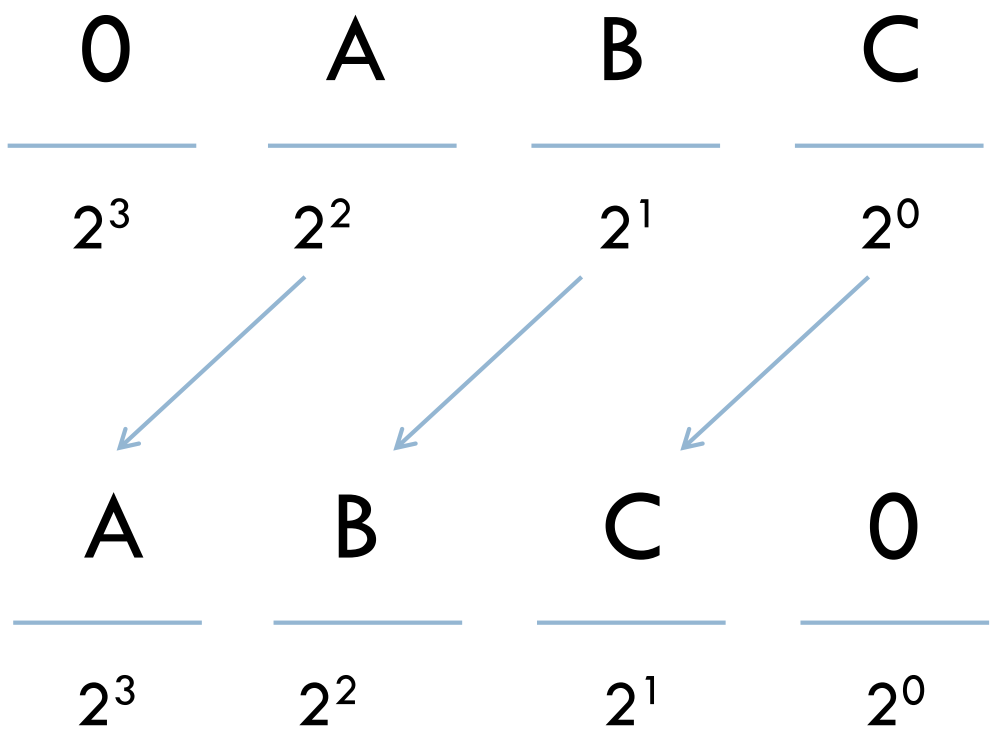
Shifting mathematically: unsigned

What does **left** shifting by one position do mathematically?



Shifting mathematically: unsigned

What does **left** shifting by one position do mathematically?



$$= A*2^2 + B*2^1 + C*2^0$$

$$= A*2^3 + B*2^2 + C*2^1$$

$$= 2*(A*2^2 + B*2^1 + C*2^0)$$

Shifting mathematically: unsigned

What does **left** shifting by one position do mathematically?

0	A	B	C
2^3	2^2	2^1	2^0

$$= A*2^2 + B*2^1 + C*2^0$$

A	B	C	0
2^3	2^2	2^1	2^0

$$= A*2^3 + B*2^2 + C*2^1$$

$$= 2*(A*2^2 + B*2^1 + C*2^0)$$

Doubles the number!

Shifting mathematically: unsigned

What does **left** shifting by ***n*** positions do mathematically?

Multiply by 2^n (double n times)

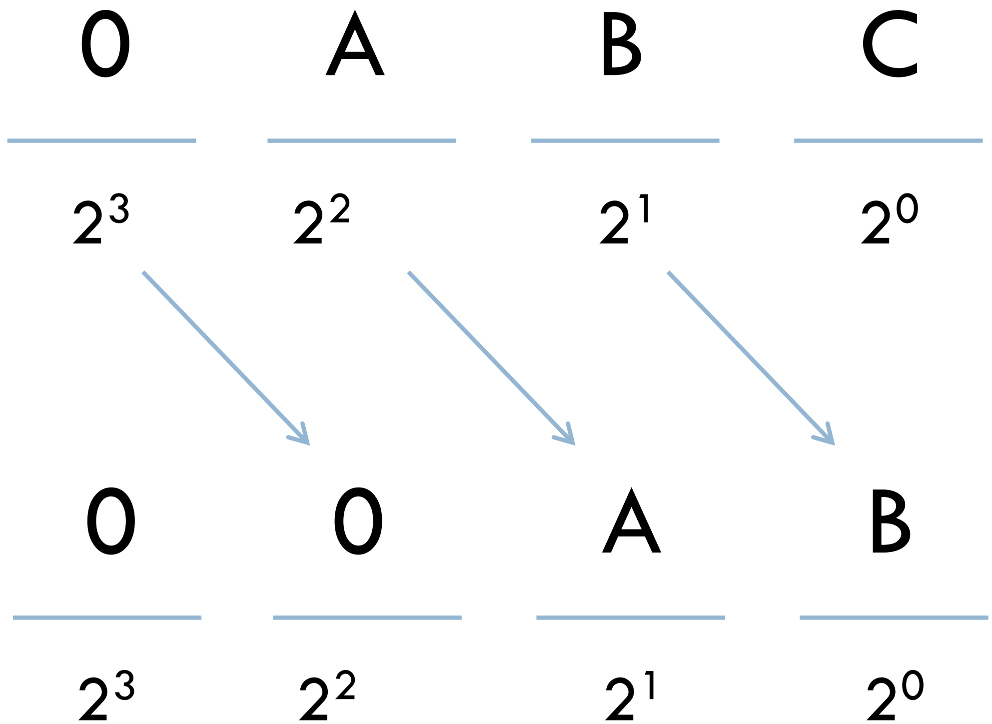
Shifting mathematically: unsigned

What does **right** shifting by one position do mathematically?

0	A	B	C
2^3	2^2	2^1	2^0

Shifting mathematically: unsigned

What does **right** shifting by one position do mathematically?



Shifting mathematically: unsigned

What does **right** shifting by one position do mathematically?

0	A	B	C
2^3	2^2	2^1	2^0

$$= A * 2^2 + B * 2^1 + C * 2^0$$

0	0	A	B
2^3	2^2	2^1	2^0

$$= A * 2^1 + B * 2^0$$

$$= (A * 2^2 + B * 2^1 + C * 2^0) \text{ div } 2$$

Shifting mathematically: unsigned

What does right shifting by one position do mathematically?

0	A	B	C
2^3	2^2	2^1	2^0

$= A*2^2 + B*2^1 + C*2^0$

0	0	A	B
2^3	2^2	2^1	2^0

$= A*2^1 + B*2^0$

$= (A*2^2 + B*2^1 + C*2^0) \text{ div } 2$

Integer divide by 2
(// in Python)

Shifting mathematically: unsigned

What does **right** shifting by ***n*** positions do mathematically?

Integer division by 2^n (halve n times)

Shifting 4-bit numbers

Shifting shifts the binary representation of the number right or left

-4 >> 1

?

Shifting 4-bit numbers

Shifting shifts the binary representation of the number right or left

-4 >> 1

What is -4 as a 4-bit binary number?

-4 → 1100

number in binary

Shifting 4-bit numbers

Shifting shifts the binary representation of the number right or left

-4 >> 1

How do we fill in the leftmost bit?

-4 → 1100 → ?110

number in binary

shift right one position (discard bits shifting off)

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 → 1100 → ?110

number in binary

shift right one position (discard bits shifting off)

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 → 1100 → ?110

number in binary

shift right one position (discard bits shifting off)

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 → 1100 → 1110

number in binary

shift right one position (discard bits shifting off)

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 >> 1

-4 → 1100 → 1110 → ?

number in binary

shift right one position (discard bits shifting off)

decimal form

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 >> 1

-4 → 1100 → 1110 → -2

number in binary

shift right one position (discard bits shifting off)

decimal form

Shifting 4-bit numbers

Shifting shifts the binary representation of the number right or left

-4 >> 2

?

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 >> 2

-4 → 1100 → 1111 → -1

number in binary

shift right two positions (discard bits shifting off)

decimal form

Arithmetic shifting mathematically

What does **right** arithmetic shifting by ***n*** positions do mathematically for **signed numbers**?

Integer division by 2^n (halve n times)

Same thing!!

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 → 1100 → ?110

number in binary

shift right one position (discard bits shifting off)

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 → 1100 → 0110

number in binary

shift right one position
(discard bits shifting off)

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 >>> 1

-4 → 1100 → 0110 → ?

number in binary

shift right one position (discard bits shifting off)

decimal form

Shifting 4-bit numbers

Two types of right shifts:

- logical shift: always shift in 0s
- arithmetic shift: shift in the same as the high-order bit

-4 >>> 1

-4 ➡ 1100 ➡ 0110 ➡ 6

number in binary

shift right one position (discard bits shifting off)

decimal form

Left shifts

Two types of left shifts?

- logical shift: always shift in 0s
- arithmetic shift: ?

arithmetic

-3 << 1 ?

logical

-3 <<< 1 ?

Left shifts

Two types of left shifts?

- logical shift: always shift in 0s
- arithmetic shift: ?

arithmetic

-3 << 1 1101 → 101? (double the number)

logical

-3 <<< 1 1101 → 1010

Left shifts

Two types of left shifts?

- logical shift: always shift in 0s
- arithmetic shift: ?

arithmetic

-3 << 1 1101 1010 (double the number)

logical

-3 <<< 1 1101 1010

Only one type of left shift

Shifting summarized

Arithmetic shift:

- Right shift n
  - shift n bits to the right
  - discard right n bits
  - left n bits match high-order bits of original number
  - Effect: Integer division by 2^n (halve n times)
- Left shift
  - shift n bits to the left
  - discard left n bits
  - right n bits are 0s
  - Effect: multiply by 2^n (double n times)

Logical shift right:

- left n bits are 0s (no mathematical guarantees for negative numbers)