Memory and the CPU

Given an instruction: C = A + B

- **Memory:** Stores add instruction, and variables A, B, and C.
- **CPU:** reads the instruction and values, performs the addition operation on the values and writes the result to C in memory.

We can think of memory as an array of bytes; we will access memory **4-bytes** at a time. Each byte is 8 bits (1 or 0)

How does the CPU interact with memory?

Typically the CPU ______ a number of bytes from memory (to access a value), or ______ to memory (to initialize or update a variable value).

When working with a sequence (multiple) bytes, we need 2 things:

1.

2.

Variables and memory

In this class, we will use 32-bit (4-byte) integers.

Range of values in 1 byte:

Last day we saw two different types of variables:

1. Variables: have an address, size, and a value.
2. Pointers: whose value is a memory address.

Addr	Value
...	
200	8
204	
208	200
...	

Binary <-> Decimal <-> Hexidecimal

A hexit is a hexadecimal digit.
Typically, we write a hexadecimal value with a preceding 0x.

Convert 0x4c3 to binary (base-2) and decimal (base-10):

Subtracting Hexadecimal Numbers

We can subtract hexits the same way we subtract numbers in decimal. Remember there are 16-digits in hex, so the carry is 16 instead of 10.

```
  D E A 9
 -4 F B D
```

Twos compliment notation: *-x = ~x + 1*

Negate a number by complimenting it and incrementing it by 1.
What is the negative value in binary for 6?

Big and Little Endian

Associated with how processors interpret data.
Big endian eat numbers from big part of number first.
Little endian eat numbers starting at little end.

Draw the memory contents for value `0x12345678`

Big Endian:

Memory

Little Endian:

Memory

Address Alignment
We will align our addresses so that each address is aligned to the size of the data type mod zero.

This means that a 4-byte value will be aligned to every memory location with an address divisible by 4 (0, 4, 8, 12, 16, etc), whereas a 10-byte value will be aligned to only memory addresses divisible by 10.

Bit Shifts
Shifting **left** *b* bits is the same as ______________ by 2^b.
Shifting **right** *b* bits is the same as ______________ by 2^b.

`00101100 (44)` shifted right one bit: __________________
`00101100 (44)` shifted left one bits: __________________

For signed values, the sign bit (far-left) remains unchanged.

`-6 == 11111010` shifted right: `01111101 =` ______
`-6 == 11111010` shifted right: `11111101 =` ______

`Why?`

Extension and Truncation

- Extension:
- Truncation:

What value will be printed in each of the following print statements:

```
...
int i = 0x1234; //4660
byte b = (byte) i;
out.printf("%x\n", b);
...
```

```
...
int i = 0x87; //135
byte b = (byte) i;
out.printf("%x\n", b);
...
```

What value will be printed in each of the following print statements:

```
...
byte b = 0x12; //18
int i = b;
out.printf("%x\n", i);

int i = 0x8B << 16;
out.printf("%x\n", i);
```

```
...
int i = ((byte) 0x8b)<< 16;
 out.printf("%x\n", i);

i = 0xff8b0000 & 0x00ff0000;
 out.printf("%x\n", i);
```