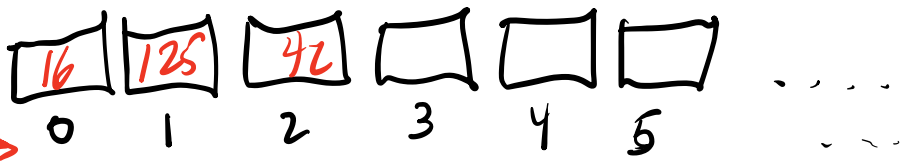
# Memory - "RAM"

- random access memory.
- think of it as an array of bytes.
- where an index is used to specify a particular byte in memory.
- the index is called an "address".

each of these is a byte




addresses: not stored, the memory hardware uses the address to directly access a specified byte.

How many bits (digits) must an address contain in order to specify any byte in memory?

Since n binary digits gives us $2^n$ different numbers (addresses), where those numbers are $0 \ldots 2^n - 1$, gives a memory with $2^n$ bytes, we need n binary digits (bits) in the address.

- Given a memory with M bytes, we need log M bits in the address.
  - memory size is always a power of 2.

- If an address is n bits, it can access every byte of a memory with $2^n$ bytes.
  - the addresses are $0 \ldots 2^n - 1$.

Sample quiz question:

How big a memory can be accessed using 32 bit addresses?

Answer: $2^{32} = 2^{30} * 2^2 = 4G$

How many bits do you need in an address to access a 64T memory?

Ans: $\log 64T = \log 64 + \log 1T$

$= 6 + 40 = 46$

---

Computer Scientists often want to specify a particular bit sequence in their program.

- How would they write it out?

- You could specify each bit explicitly:

```
int x = 0011010101100111000110016
```

↳ some compilers won't let you do this, and it's error prone.

- Or, you could write the number in decimal:

```
int x = 6492345;
```

↳ this requires converting between decimal & binary.

- Instead, people write in Hexadecimal ("Hex")
  - base 16 number system
  - digits are 0, 1, 2, 3, 4, 5, 6, 7
    8, 9, A, B, C, D, E, F

these represent ten through fifteen in decimal.

  - because $2^4 = 16$, each hex digit corresponds to exactly 4 binary digits (bits).

this is in the handout.

| Decimal | Hex | Binary |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

Know this for the quiz

So, converting from hex to binary is easy: Just 4 bits (1 hex digit) at a time.

hex → 4AC8 = 0100101011001000 (binary)

4 → 0100, A → 1010, C → 1100, 8 → 1000

binary → 1111001011010110 = F2D6 (hex)

F 2 D 6

In C:

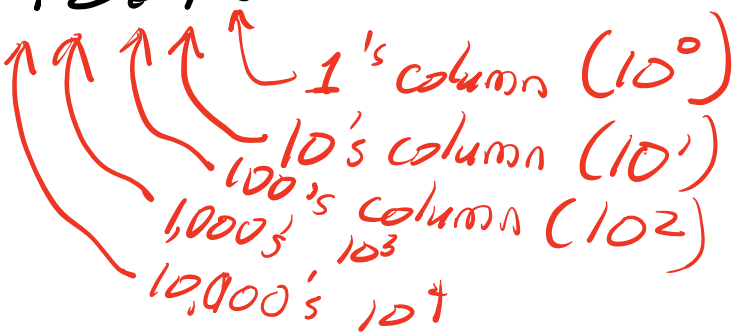
```
int x = 0xBC34;
```

0x tells the compiler you are writing a hex number

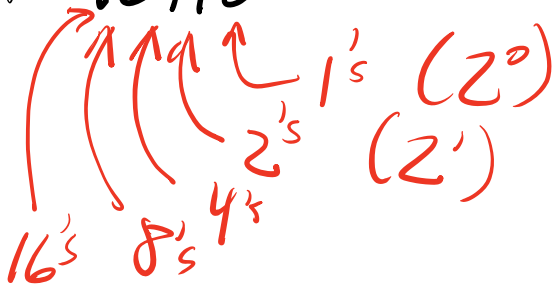
BC34: hex number

# Number systems:

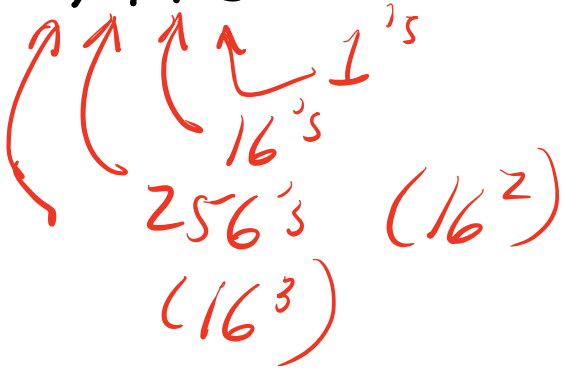
Decimal: 12645




Binary: 10110



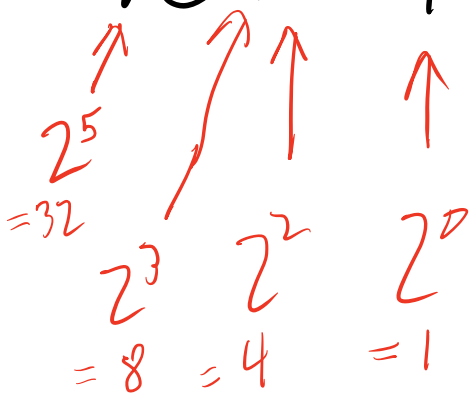

Hexadecimal: 4A9C



# Converting binary to decimal:

$101101 = 1 + 4 + 8 + 32$

$= 45$



# Decimal to binary:

$654 = 512 + 128 + 8 + 4 + 2$

512

142

128

14

8

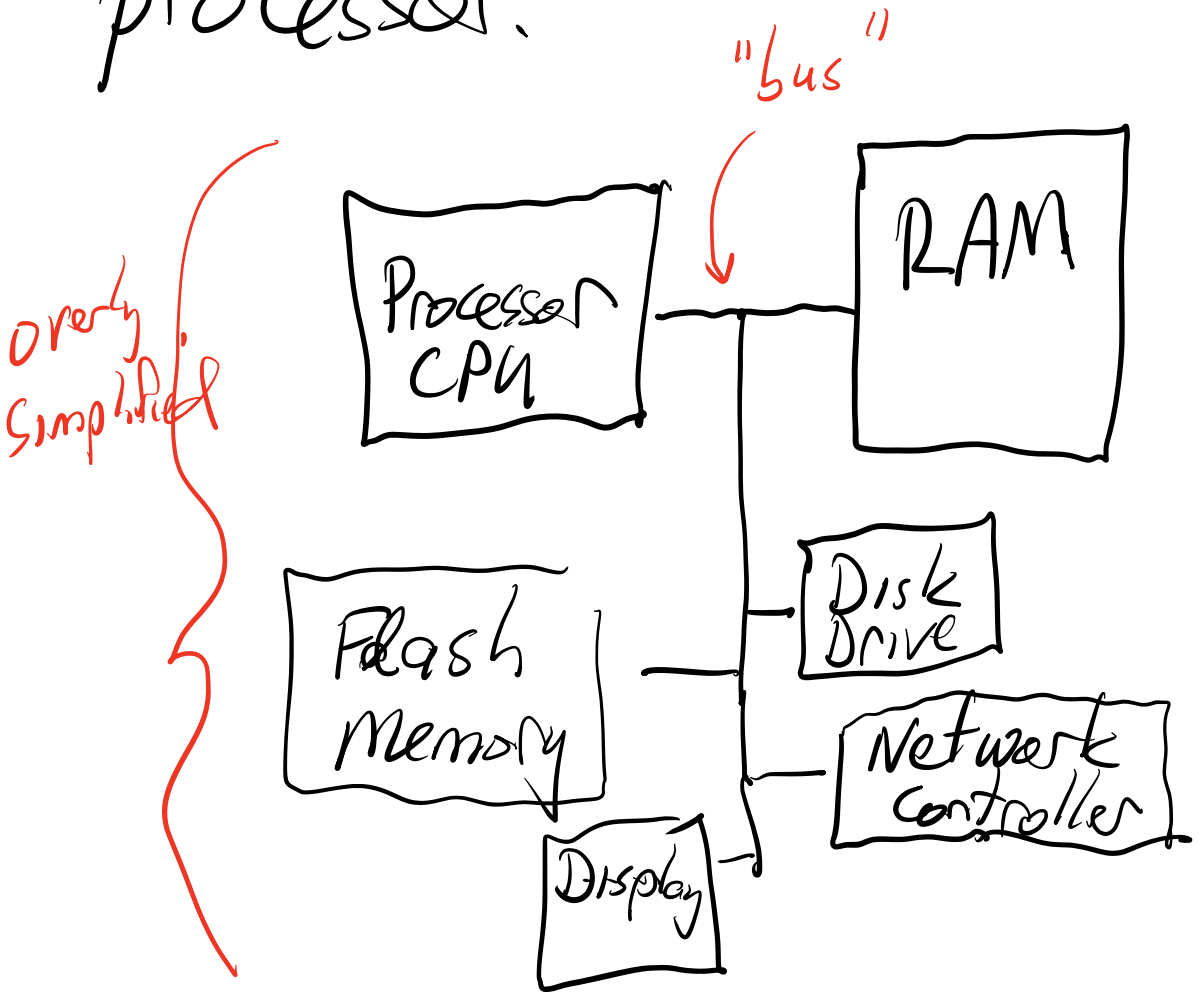
6

$= 2^9 + 2^7 + 2^3 + 2^2 + 2^1$

$= 1010001110$

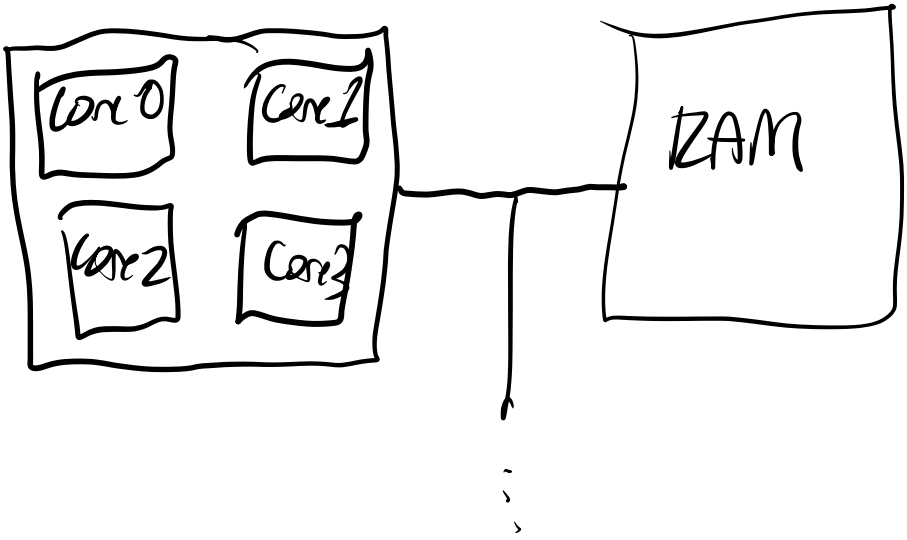
What a pain!

# Going back to memory & processor:

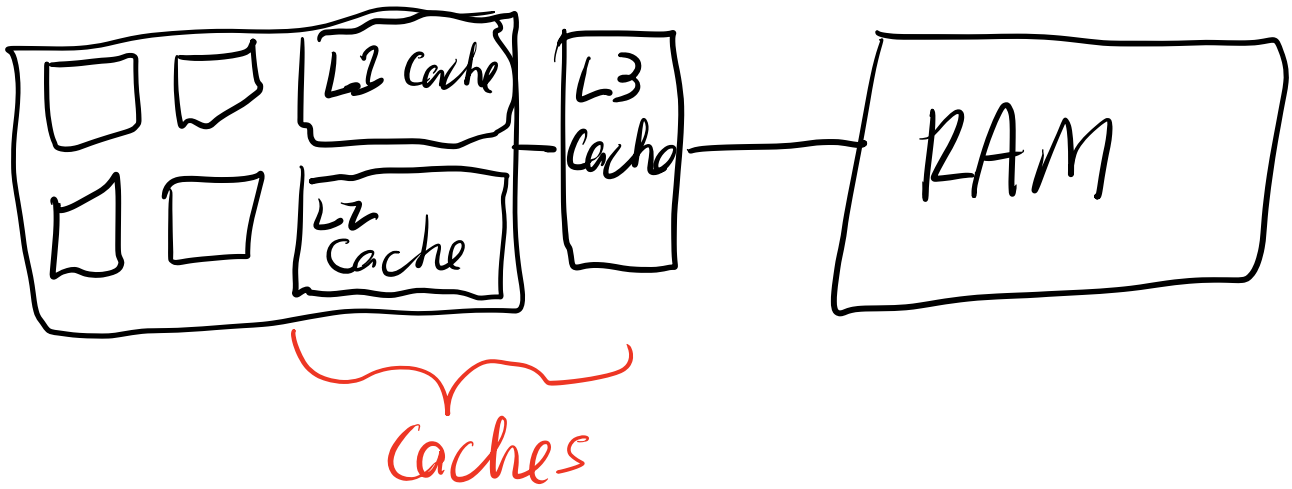


Modern Processors have multiple "cores".

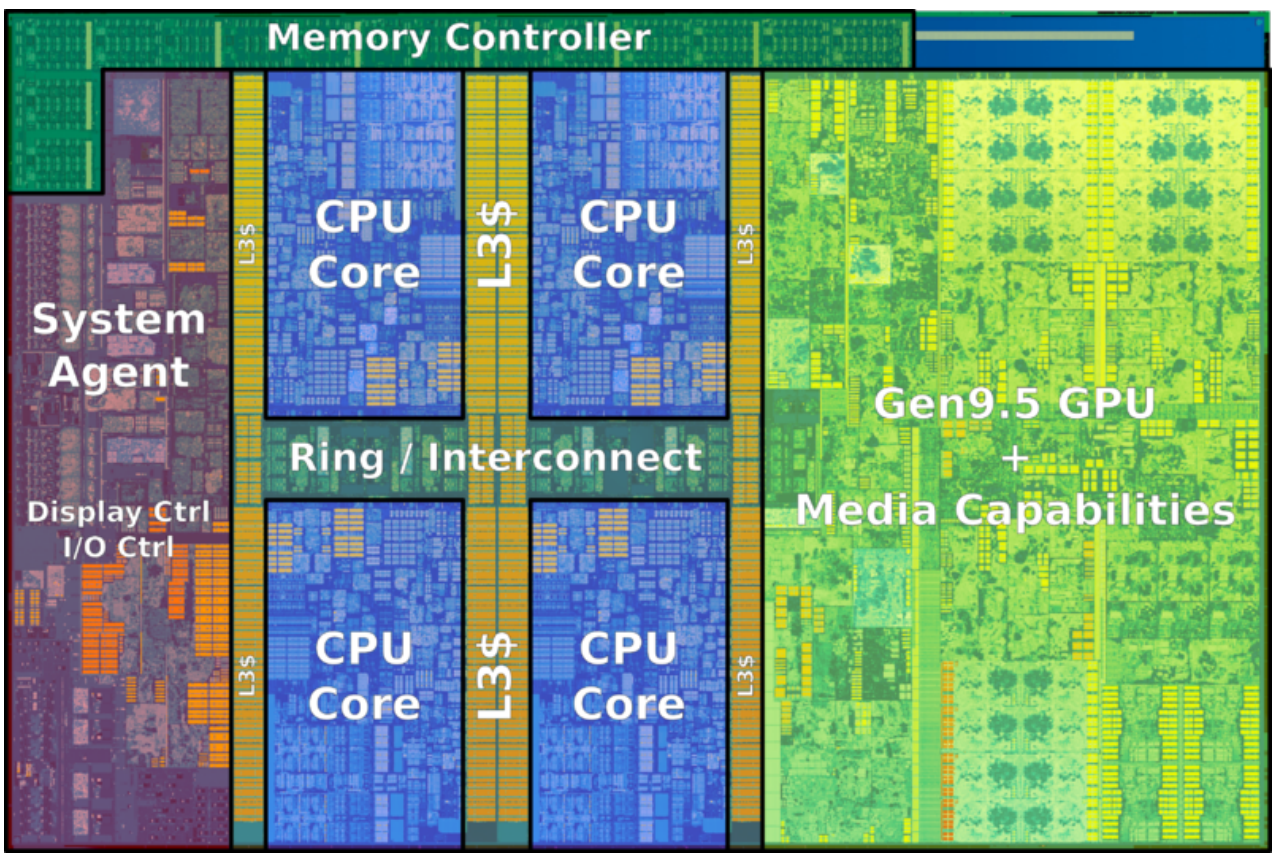
- each core is a little processor that executes instructions.



To reduce the time retrieving data and instructions from RAM, a processor contains small memories called "caches"



Here is what it really looks like on a 4-core Intel Processor ("Kaby Lake"):



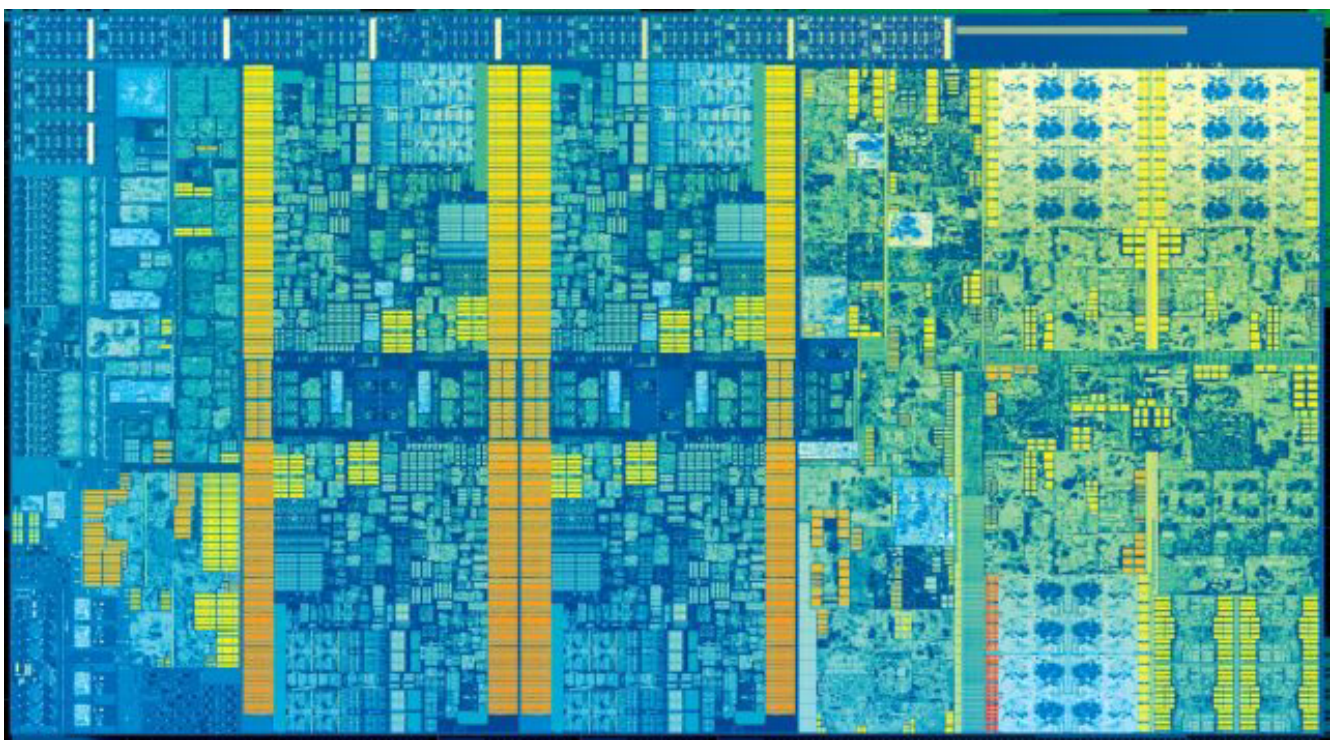

4 cores

L3 cache

Graphics processor (GPU)

I/O interface

Under a microscope:



5 - 10 billion transistors!