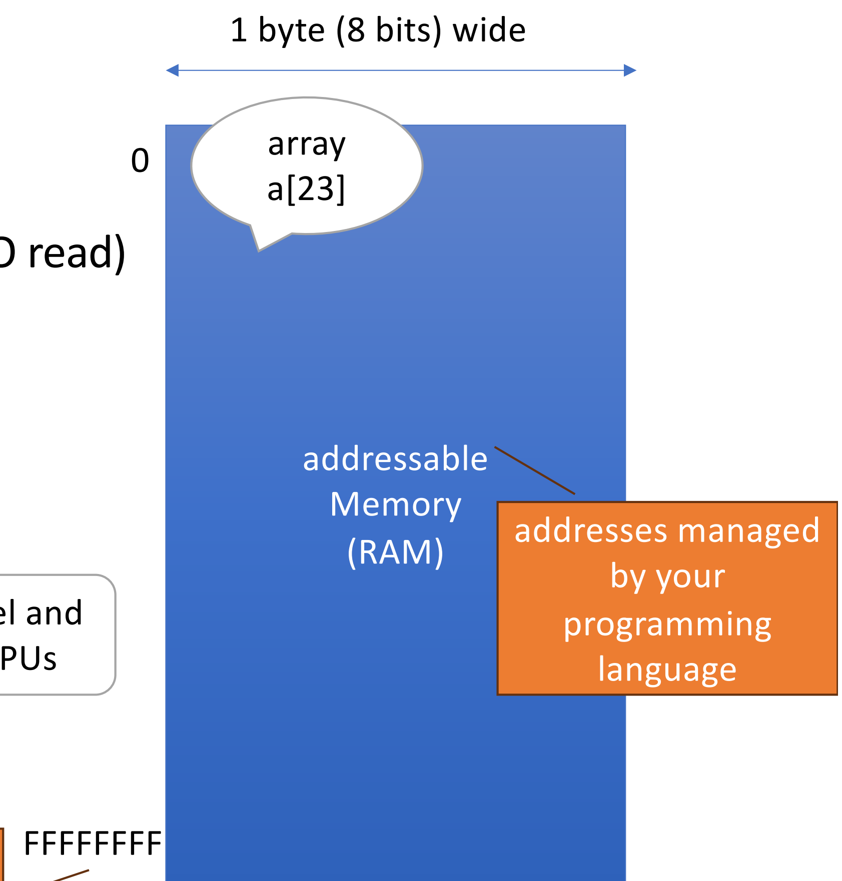
Lecture 2

408/508 *Computational Techniques for Linguists*

Introduction

- RAM memory (your program's view)
  - the area to store your program and data (after I/O read)
- Underlying representation
  - binary: zeros and ones (1 bit)
  - organized into **bytes** (8 bits) and **nibbles** (4 bits)
    - memory is byte-addressable
  - **word** (32 bits)
    - e.g. integer
    - (64 bits: floating point number)
  - big-endian/little-endian
    - most significant byte first or least significant byte
    - *matters for communication …*

your Intel and ARM CPUs



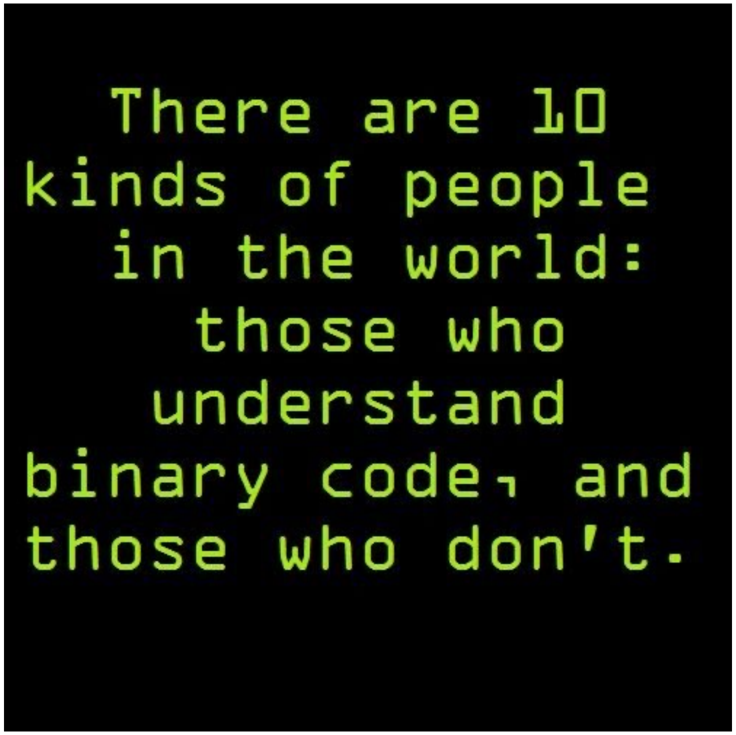
hexadecimal notation

Joke

- Continue with the introduction
  - binary representation (base 2) and arithmetic

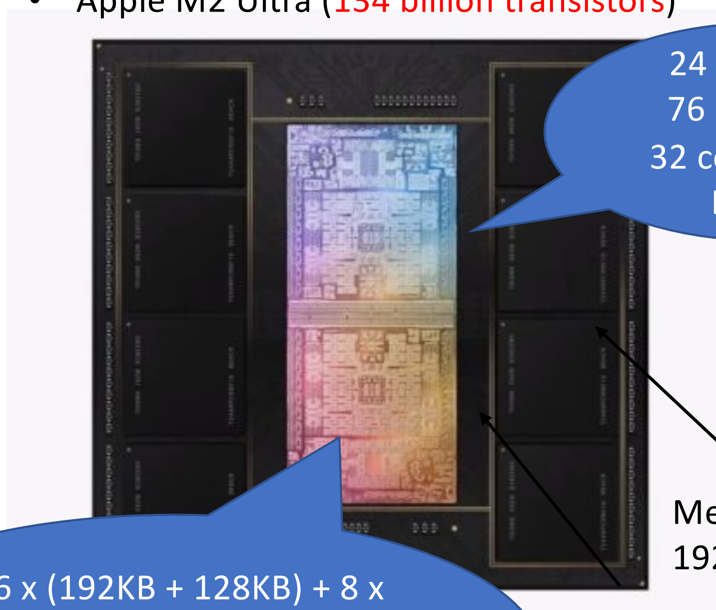
2^1	2^0	Total (decimal)
1	0	2

10^1	10^0	Total
1	0	10



Background

- **TDP 62W** power (250W max?)
- Apple M2 Ultra (134 billion transistors)



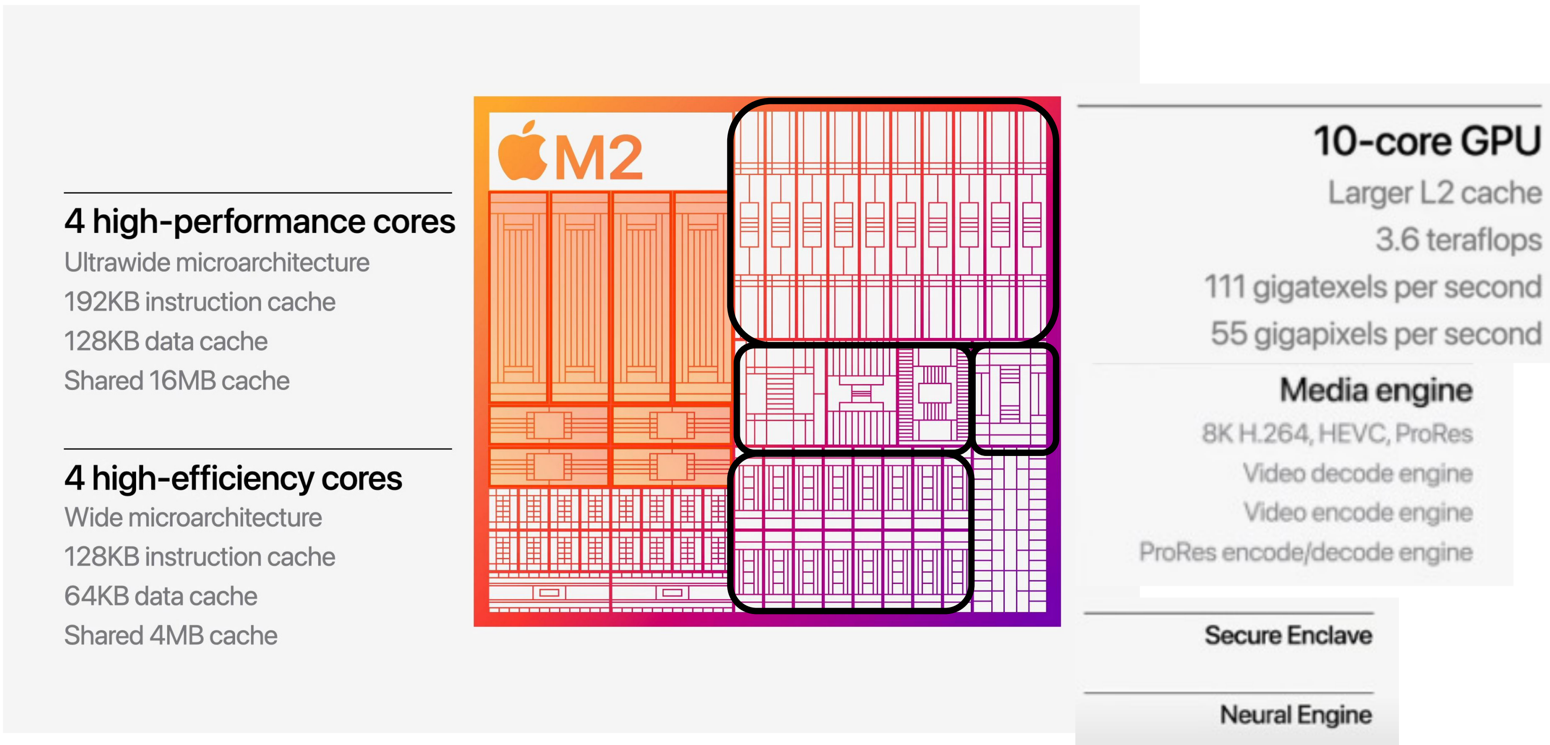
- **Human brain:** 86 billion neurons[1]

- **20W** of power (oft-cited figure)

[1]Herculano-Houzel, S. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain . *Frontiers in Human Neuroscience*. 2009;3:31. doi:10.3389/neuro.09.031.2009.

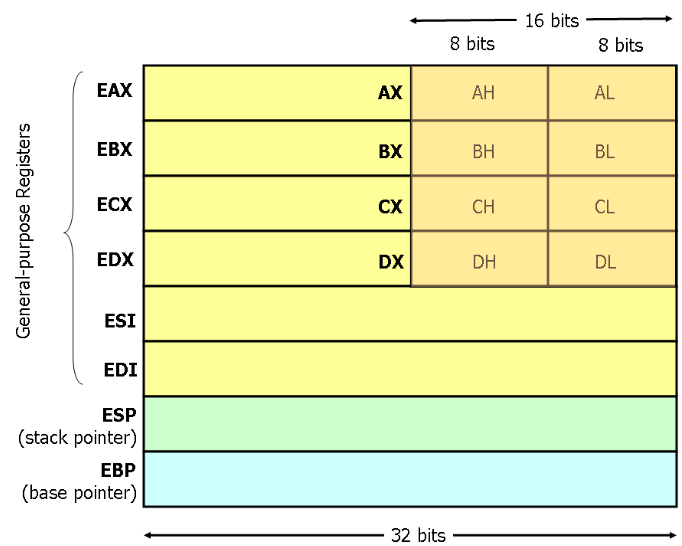
Introduction

A 8 core machine



Introduction

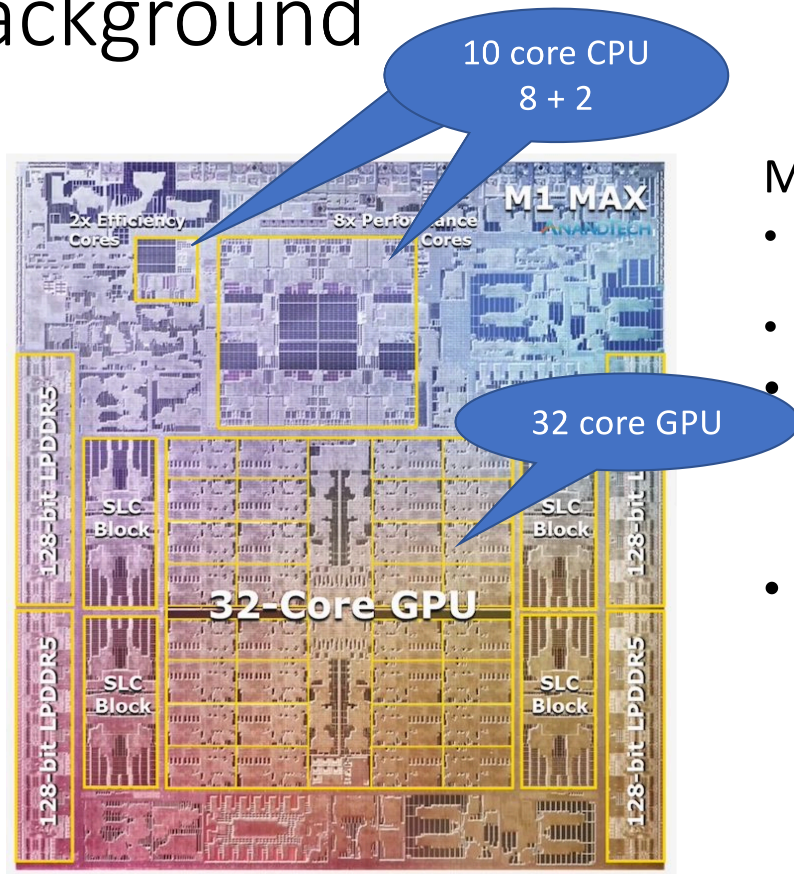
- Machine Language
  - A CPU understands only one language: machine language (Intel/AMD x86)
    - **all other languages** must be translated into machine language
  - Primitive instructions include:
    - MOV
    - PUSH
    - POP
    - ADD / SUB
    - INC / DEC
    - IMUL / IDIV
    - AND / OR / XOR / NOT
    - NEG
    - SHL / SHR
    - JMP
    - CMP
    - JE / JNE / JZ / JG / JGE / JL / JLE
    - CALL / RET



Assembly Language: (this notation)
by definition, nothing built on it is more powerful

http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

Background



Machine Language (ML)

- A CPU understands only one language: machine language (ARM64)
- **all other languages** must be translated into ML
- Each core is an independent CPU itself:
  - 31 general purpose registers X0..X31
  - 32 floating point/vector registers
  - SP
- Primitive instructions include:

```
• ADD X0, X1, X2
• MOV X0, #1
• LDR X0, [<address>]
• STRW X0, [<address>]
• CBZ <Xn> <label> and CBNZ <Xn> <label>
• BL <label> / RET
```

https://developer.arm.com/documentation/ddi0602/2022-12/?lang=en

Fugaku supercomputer

- World's fastest computer (using the ARM instruction set) until May 2022
- Power consumption of 30 (to 40) megawatts (#1)
- https://www.fujitsu.com/global/about/innovation/fugaku/

Number of Nodes

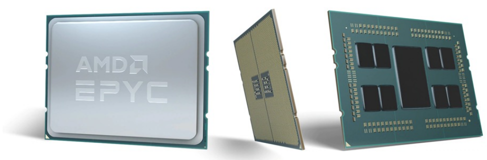
Number of Nodes	158,976 nodes

Node

Architecture	Armv8.2-A SVE 512 bit With the following Fujitsu's extensions: Hardware barrier, Sector cache, and Prefetch
Number of computational cores	48 cores

Supercomputers rely on parallelism

- each chip may have 64 cores each



- Not all tasks are easily parallelizable.
  - Can you use 8 million cores?
  - e.g. *analyze 8 million sentences at a time*?

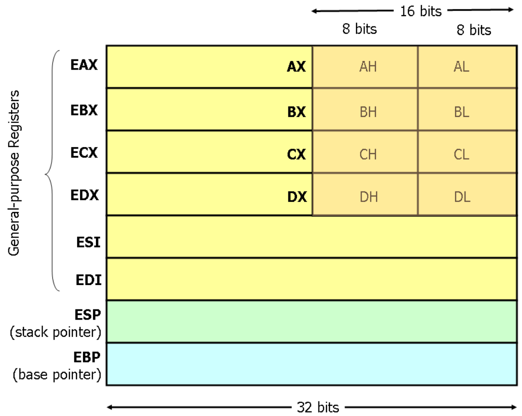
https://www.top500.org

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1,463,616	174.70	255.75	5,610

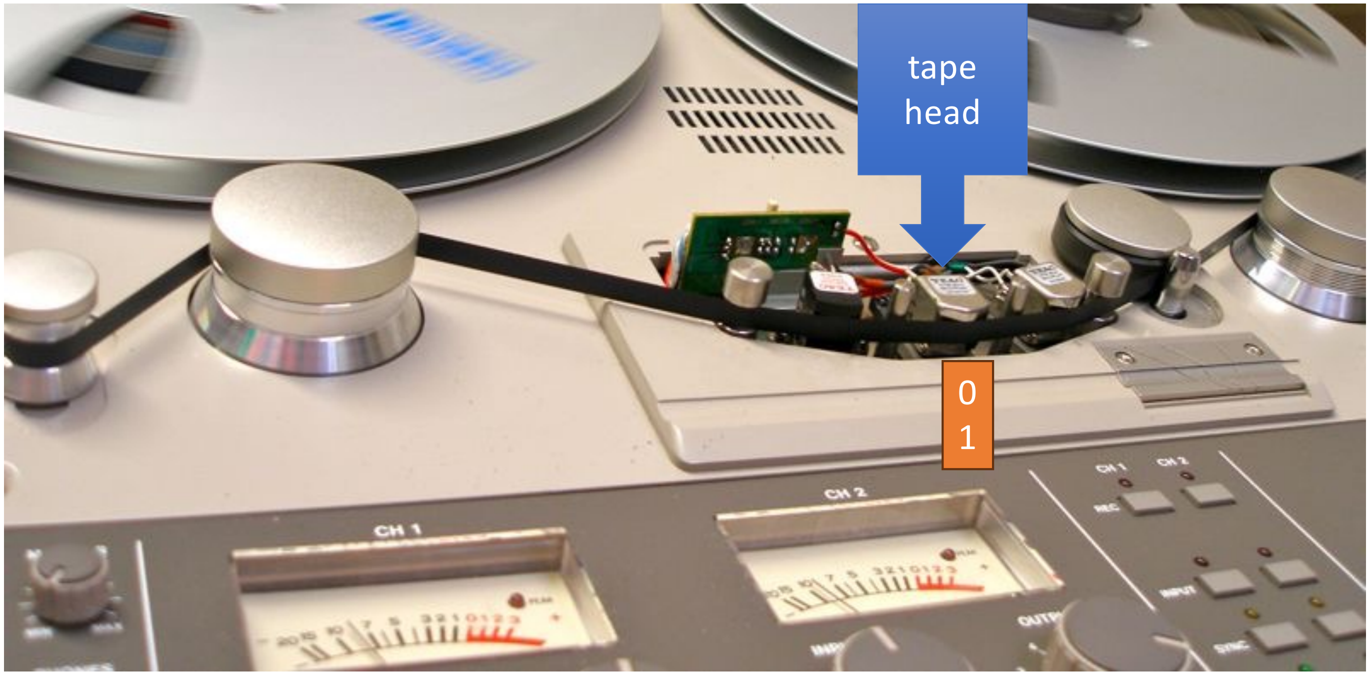
Introduction

- Not all machine instructions are conceptually necessary
  - many provided for speed/efficiency
- Theoretical Computer Science
  - All mechanical computation can be carried out using a TURING MACHINE (a-machine; Turing, 1936)
  - Finite state table + (infinite) tape
  - Tape instructions:
    - at the tape head: Erase, Write, Move (Left/Right/NoMove)
  - Finite state table:
    - Current state x Tape symbol --> new state x New Tape symbol x Move

Assembly Language instructions:
MOV/ADD/JMP/JZ/CALL/RET



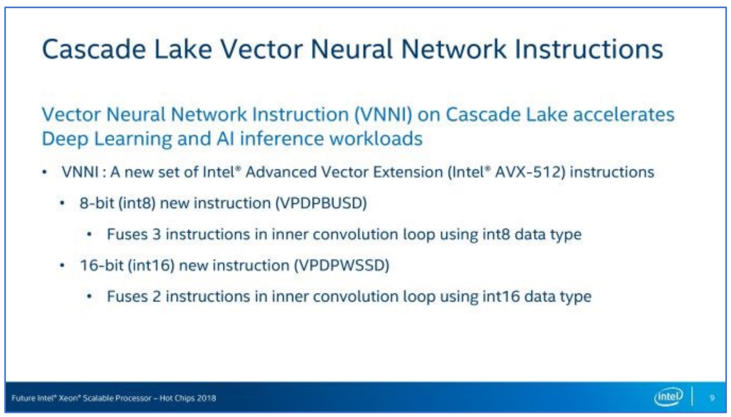
Introduction

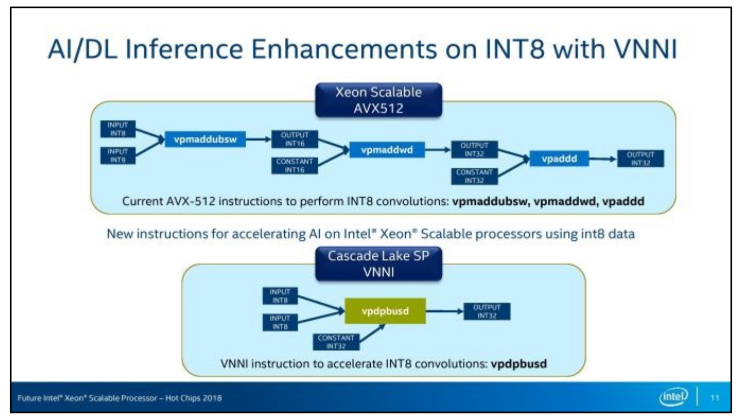


http://www.thegreatbear.net/audio-tape-transfer/quarter-inch-reel-to-reel-transfer/

Introduction

- Not all machine instructions are conceptually necessary
  - many provided for speed/efficiency
- **Example (Hot Chips 2018):**





Introduction

- Storage:
  - based on digital logic
  - binary (base 2) – everything is a power of 2
  - Byte: 8 bits
    - 01011011
    - $= 2^6+2^4+2^3+2^1+2^0$
    - = 64 + 16 + 8 + 2 + 1
    - = 91 (in decimal)
  - Hexadecimal (base 16)
    - 0-9,A,B,C,D,E,F (need 4 bits)
    - 5B (= 1 byte)
    - $= 5*16^1 + 11$
    - = 80 + 11
    - = 91

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	1	1	0	1	1

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
0	1	0	1	1	0	1	1

5 B

16^1	16^0
5	B

Introduction: data types

- Integers
  - In one byte (= 8 bits), what's the largest and smallest number, we can represent?
  - 00000000 = 0
  - 01111111 = 127
  - 10000000 = -128
  - 11111111 = -1
  - Number line:

00000000 (all zeros) ... 11111111 (all ones)

0	**…**						**127**	**-128**	**-127**						**-1**

2's complement representation

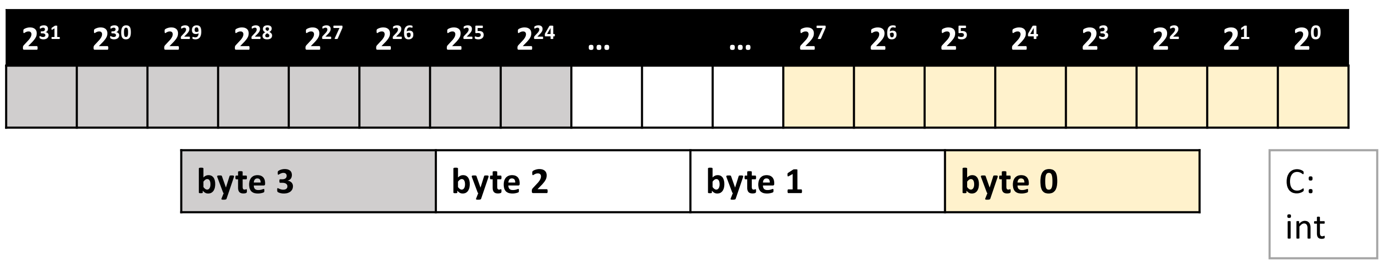
Introduction: data types

- Integers
  - In one byte, what's the largest and smallest number, we can represent?
  - **Answer**: -128 .. 0 .. 127 using the **2's complement representation**
  - Why? super-convenient for arithmetic operations
  - "to convert a positive integer X to its negative counterpart, flip all the bits, and add 1"
  - **Example**:
  - 00001010 = $2^3 + 2^1$ = 10 (decimal)
  - 11110101 + 1 = 11110110 = -10 (decimal)
  - 11110110 flip + 1 = 00001001 + 1 = 00001010

Addition:
-10 + 10
= 11110110
+ 00001010 = 0 (ignore overflow)

Introduction: data types

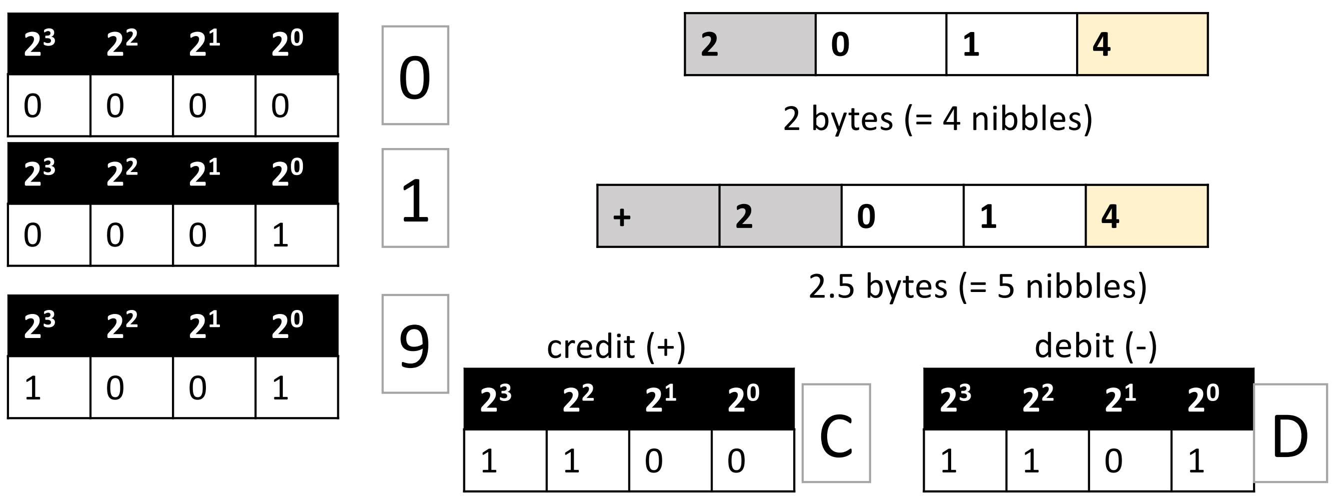
- Typically 32 bits (4 bytes) are used to store an integer
  - range: -2,147,483,648 ($2^{(31-1)}$ -1) to 2,147,483,647 ($2^{(32-1)}$ -1)



- what if you want to store even larger numbers?
  - Binary Coded Decimal (BCD)
  - code each decimal digit separately, use a string (sequence) of decimal digits …

Introduction: data types

- what if you want to store even larger numbers?
  - Binary Coded Decimal (BCD)
  - 1 byte can code two digits (0-9 requires 4 bits)
  - 1 nibble (4 bits) codes the sign (+/-), e.g. hex C/D



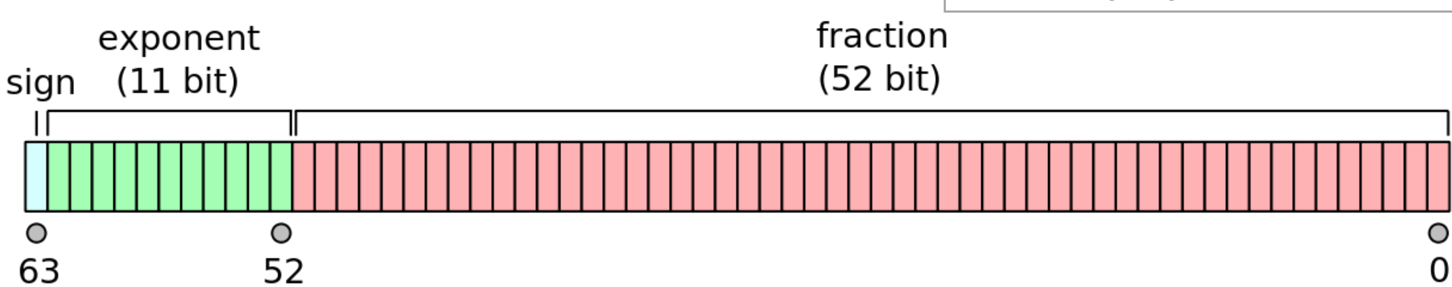
Introduction: data types

e.g. probabilities

- Typically, 64 bits (8 bytes) are used to represent floating point numbers (double precision)
  - c = 2.99792458 x 10^8 (m/s)
  - coefficient: 52 bits (**implied 1**, therefore treat as 53)
  - exponent: 11 bits (usually not 2's complement, unsigned with bias $2^{(10-1)}$-1 = 511)
  - sign: 1 bit (+/-)

C:
float
double

x86 CPUs have a built-in floating point coprocessor (x87)
80 bit long registers



wikipedia

Example 1

- The speed of light:
    - c = 2.99792458 x 10^8 (m/s)

1. Can a 4 byte integer be used to represent *c* exactly?
    - 4 bytes = 32 bits
    - 32 bits in 2's complement format
    - Largest positive number is
    - 2^{31}-1 = 2,147,483,647
    - c = 299,792,458

Example 2

- Recall the speed of light:
  - *c* = 2.99792458 x 10^8 (m/s)

2. How much memory would you need to encode *c* using BCD notation?
   - 9 digits
   - each digit requires 4 bits (a nibble)
   - BCD notation includes a sign nibble
   - total is 5 bytes

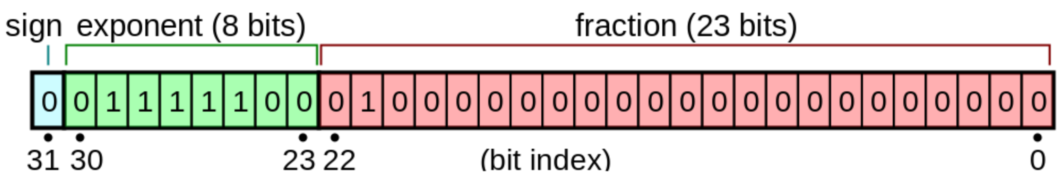
Example 3

- Recall the speed of light:
  - *c* = 2.99792458 x 10^8 (m/s)

3. Can the 64 bit floating point representation (`double`) encode *c* without loss of precision?
   - Recall significand precision: 53 bits (52 explicitly stored)
   - 2^{53}-1 = 9,007,199,254,740,991
   - almost 16 digits
   - (we only need 9 digits of precision)

Example 4

- Recall the speed of light:
  - $c = 2.99792458 \times 10^8$ (m/s)
- The 32 bit floating point representation (`float`) – sometimes called single precision - is composed of 1 bit sign, 8 bits exponent (unsigned with bias $2^{(8-1)}-1$), and 23 bits coefficient (24 bits effective).



- Can it represent *c* without loss of precision?
  - $2^{24}-1 = 16{,}777{,}215$
  - Nope (7 digits of precision)

Homework 1: Binary Exercise

- What would the integer representation of the speed of light (in m/s) look like in binary representation as a 32 bit number?

2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	…		…	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

byte 3	**byte 2**	**byte 1**	**byte 0**

Instructions

- Email to sandiway@arizona.edu
- By Friday midnight
- SUBJECT: 408/508 Homework 1: YOUR NAME
- Send me the binary bit pattern
- Either Plain Text or PDF accepted (no Word files please)