IEEE 754 Format (from last time)

- Single precision format

This is effectively a signed magnitude fixed-point number with a "hidden" 1.

The 1 is hidden because it provides no information after the number is "normalized"

The exponent is represented in bias 127 notation. Why?

$$v = -1^{s} \times 1.\text{Significand} \times 2^{\text{Exponent}-127}$$

- Example: $52.25 = 00110100.01000000_2$

Normalize: $001.1010001000000_2 \times 2^5$

(127+5)

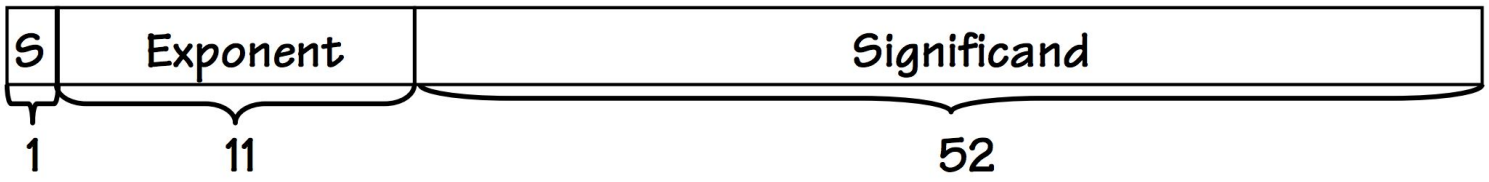
0 10000100 10100010000000000000000

0100 0010 0101 0001 0000 0000 0000 0000

52.25 = 0x42510000

IEEE 754 Limits and Features

- Single precision limitations
  - A little more than 7 decimal digits of precision
  - Minimum positive normalized value: $\sim 1.18 \times 10^{-38}$
  - Maximum positive normalized value: $\sim 3.4 \times 10^{38}$
- Inaccuracies become evident after multiple single precision operations
- Double precision format



$$v = -1^{s} \times 1.\text{Significand} \times 2^{\text{Exponent}-1023}$$

IEEE 754 Special Numbers

- Zero - ±0

  A floating point number is considered zero when its exponent and significand are both zero. This is an exception to our "hidden 1" normalization trick. There are also a positive and negative zeros.

S	000 0000 0	000 0000 0000 0000 0000 0000

- Infinity - ±∞

  A floating point number with a maximum exponent (all ones) is considered infinity which can also be positive or negative.

S	111 1111 1	000 0000 0000 0000 0000 0000

- Not a Number - NaN for ±0/±0, ±∞/±∞, log(-42), etc.

S	111 1111 1	non-zero

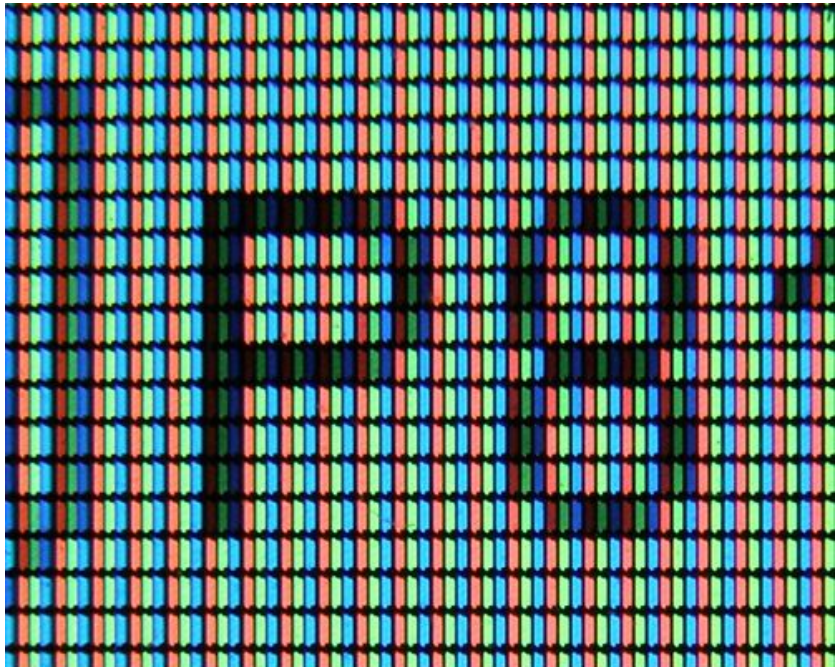
A Quick Wake-up exercise

What decimal value is represented by 0x3f800000, when interpreted as an IEEE 754 single precision floating point number?

Bits You can See

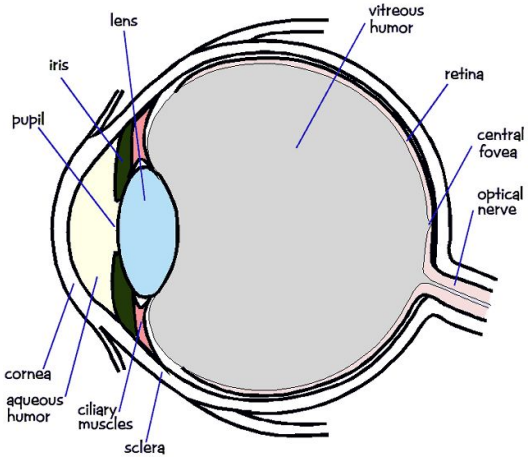
The smallest element of a visual display is called a "pixel". Pixels have three independent color components that generate most of the ***perceivable*** color range.

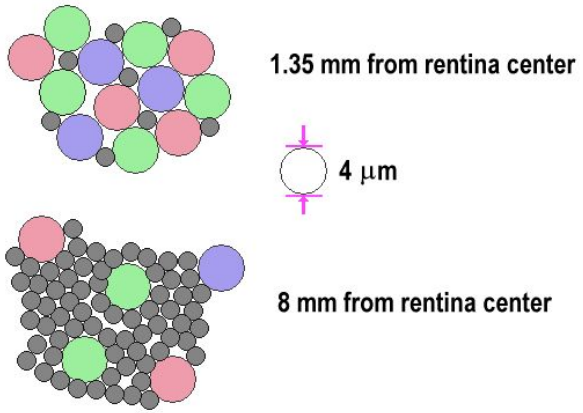
- Why three and what are they
- How are they represented in A computer?
- First, let's discuss this notion of perceivable



It starts with the Eye

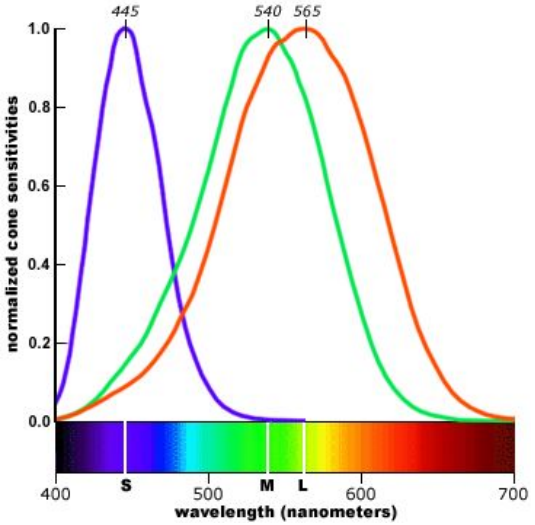
- The photosensitive part of the eye is called the retina.
- The retina is largely composed of two cell types, called rods and cones.
- Cones are responsible for color perception.
- Cones are most dense within the fovea.
- There are three types of cones, referred to as S, M, and L whose spectral sensitivity varies with wavelength.

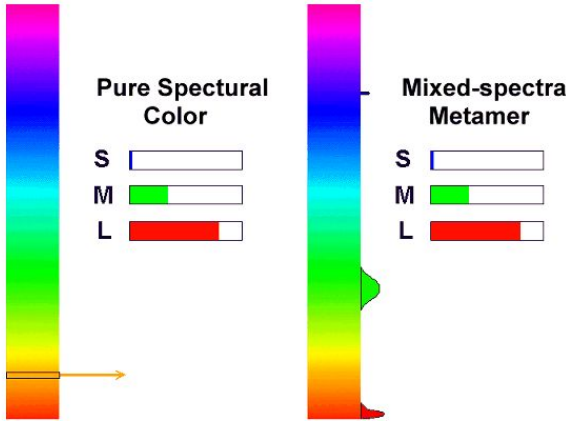




Why we see in color

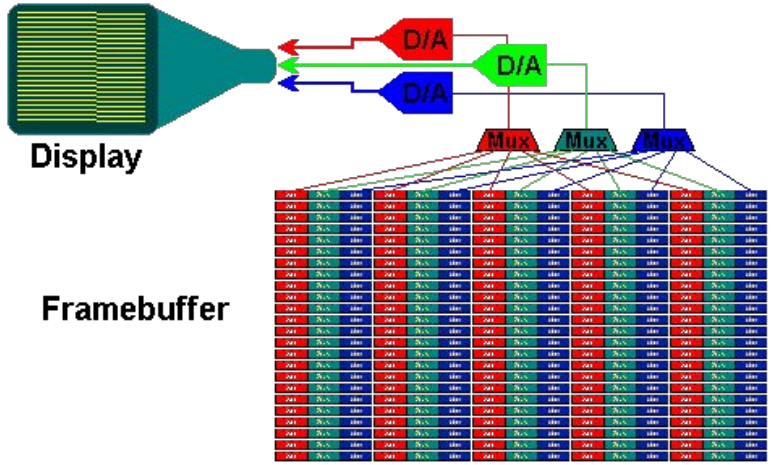
- Pure spectral colors simulate all cones to some extent.
- Mixing multiple colors can stimulate the cones to respond in a way that is indistinguishable from a pure color.
- Perceptually identical sensations are called metamers.
- This allows us to use just three colors to generate all others.

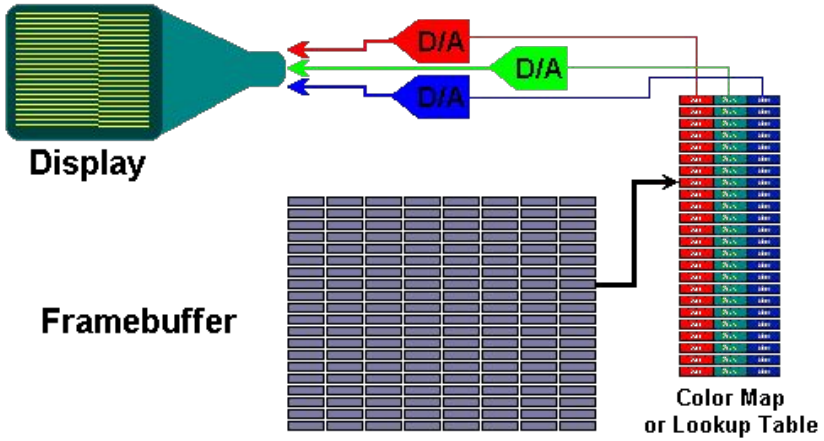




How colors Are Represented

- Each pixel is stored as three primary parts
- Red, green, and blue
- Usually around 8-bits per channel
- Pixels can have individual R,G,B components or they can be stored indirectly via a "look-up table"





3 - 8-bit unsigned binary integers (0,255)
-OR-
3 - fixed point 8-bit values (0-1.0)

Color Specifications

Web colors:

Name	Hex	Decimal Integer	Fractional
Orange	#FFA500	(255, 165, 0)	(1.0, 0.65, 0.0)
Sky Blue	#87CEEB	(135, 206, 235)	(0.52, 0.80, 0.92)
Thistle	#D8BFD8	(216, 191, 216)	(0.84, 0.75, 0.84)

Colors are stored as binary too. You'll commonly see them in Hex, decimal, and fractional formats.

Summary

- **ALL** modern computers represent signed integers using a two's-complement representation
- Two's-complement integer representations eliminate the need for separate addition and subtraction units
- Addition is **identical** using either unsigned and two's-complement numbers
- Finite representations of numbers on computers leads to anomalies
- Floating point numbers have separate fractional and exponent components.

Behind the Curtain

1. Computer organization
2. Computer Instructions
3. Memory concepts
4. Where should code go?
5. Computers as systems

On Friday (8/31) we'll have a lecture from 9:05-10:20.

Labs start next Friday (9/7)

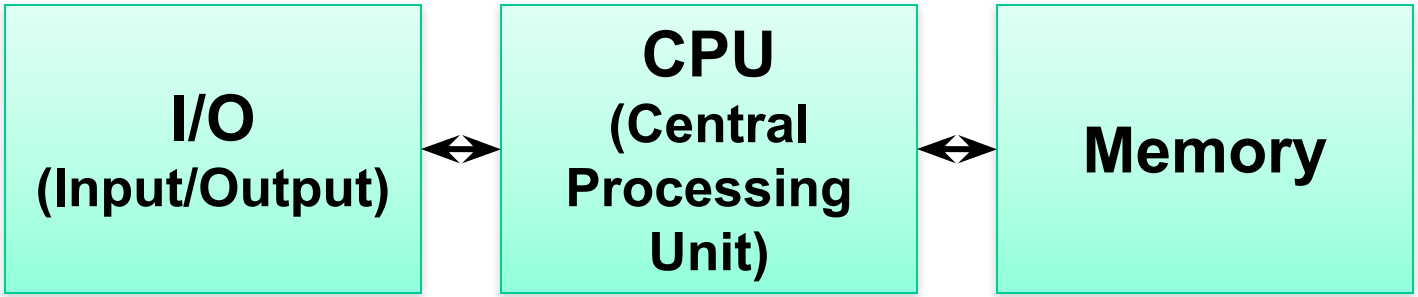
Computers Everywhere

The computers we're used to

- Desktops
- Laptops
- Tablets

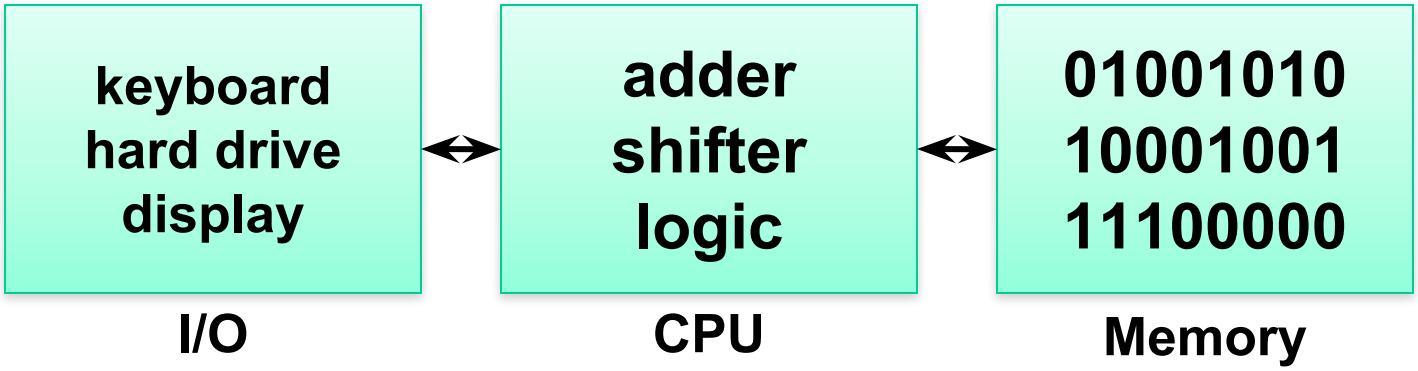
- Embedded processors
  - Cars
  - Light bulbs
  - Mobile phones
  - Toasters, irons, wristwatches, happy-meal toys

Computer Organization



- Every computer has at least three basic units
  - Input/Output
    - where data arrives from the outside world
    - where data is sent to the outside world
    - where data is archived for the long term (i.e. when the lights go out)
  - Memory
    - where data is stored (numbers, text, lists, arrays, data structures)
  - Central Processing Unit
    - where data is manipulated, analyzed, etc.

Computer Organization (cont)



- Properties of units
  - Input/Output
    - converts symbols to bits and vice versa
    - where the analog "real world" meets the digital "computer world"
    - must somehow synchronize to the CPU's clock
  - Memory
    - stores bits that represent information
    - every unit of memory has an "address" and "contents",
  - Central Processing Unit
    - besides processing, it also coordinates data's movements between units

What sort of "Processing"

A CPU performs low-level operations called **INSTRUCTIONS**

Arithmetic

- ADD X to Y then put the result in Z
- SUBTRACT X from Y then put the result back in Y

Logical

- Set Z to 1 if X AND Y are 1, otherwise set Z to 0
  (AND X with Y then put the result in Z)
- Set Z to 1 if X OR Y are 1, otherwise set Z to 0
  (OR X with Y then put the result in Z)

Comparison

- Set Z to 1 if X is EQUAL to Y, otherwise set Z to 0
- Set Z to 1 if X is GREATER THAN OR EQUAL to Y, otherwise set Z to 0

Control

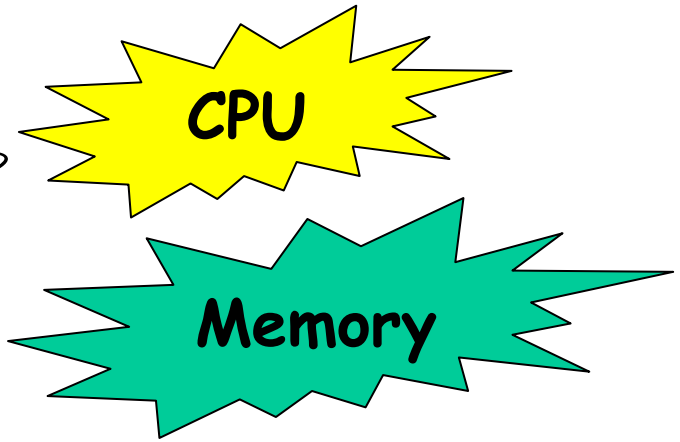
- Skip the next INSTRUCTION if Z is EQUAL to 0

Anatomy of an Instruction

Nearly all instructions can be made to fit a common template

Issues remaining …

- Which operations to include?
- Where to get variables and constants?
- Where to store the results?

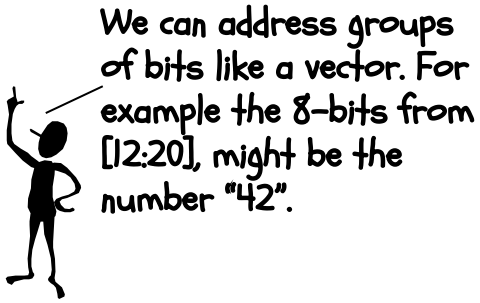


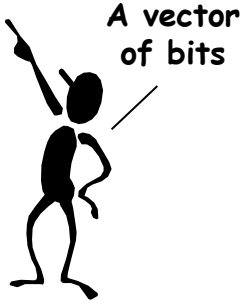
How is Memory Organized

- By now you know memory is a vast collection of bits
- **Groups of bits** can represent various types of data
  - Integers, Signed integers. Floating-point values, Strings, Pixels
- How do bits get "Grouped"?
- Memory is organized as a vector of bits with indices called "addresses"

0 1 0 0 1 0 1 0 1 0 1 0 [0 0 1 0 1 0 1 0] 0 0 1 0 1 1 1 1 ...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27





Addresses are Key!

The need to "address" bits is one of the most important factors of a computer's design.

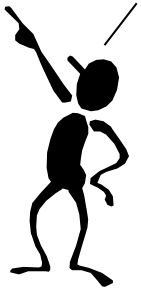
- How many bits will I ever need?
  (remember computer representations are finite)
- The size of scratch variables (registers), is more determined by the need to address bits than the size of the data-types needed..
- Should we squander address space by giving "every" bit a distinct address?
- Perhaps we could address bits in more manageble units

Memory Concepts

- Memory is divided into "addressable" units, each with an address (like an array with indices)
- Addressable units are usually larger than a bit, typically 8 (byte), 16 (halfword), 32 (word), or 64 (long) bits
- Each address has variable "contents"
- Memory contents might be:
  - Integers in 2's complement
  - Floats in IEEE format
  - Strings in ASCII or Unicode
  - Data structure de jour
  - ADDRESSES
  - Nothing distinguishes the difference

Address	Contents
0	42
1	3.141592
2	"Lee "
3	"Hart"
4	"Bud "
5	"Levi"
6	"le "
7	2
8	0xe3a00000
9	0xe3a0100a
10	0xe0800001
11	0xe2511001
12	0x1affffc
13	0xeaffffe
14	0x00004020
15	0x20090001

Here we assume a 32-bit "Word" address-able machine



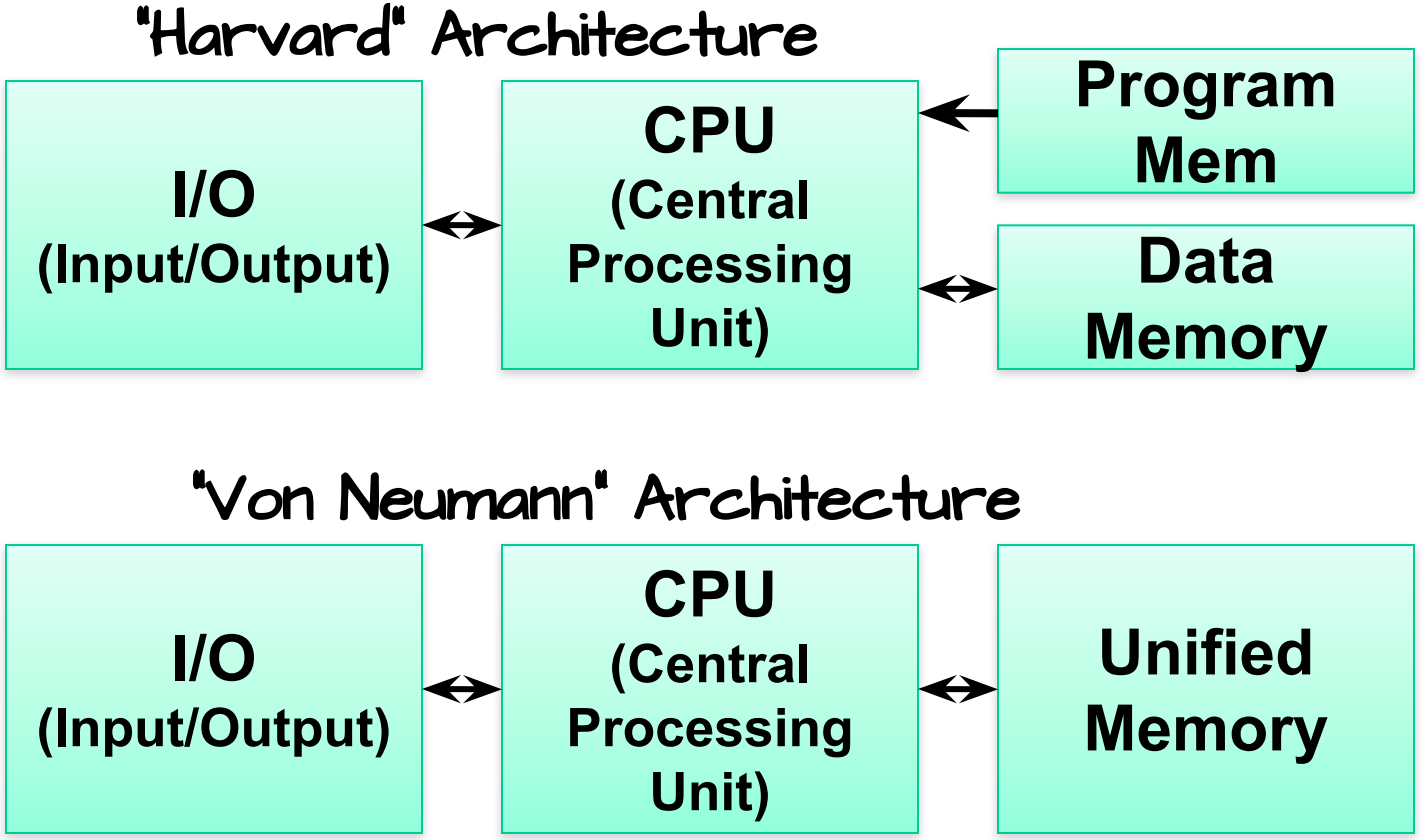
One More Thing

- INSTRUCTIONS for the CPU are stored in memory along with data
- CPU fetches instructions, decodes them and then performs their implied operation
- Mechanism inside the CPU directs which instruction to get next.
- They appear in memory as a string of bits that are typically uniform in size
- Their encoding as "bits" is called "machine language." ex: 0c3c1d7fff
- We assign "mnemonics" to particular bit patterns to indicate meanings.
- These mnemonics are called **Assembly language**. ex: mov r1, #10

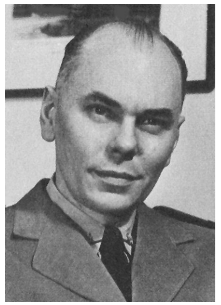
Address	Contents
0	42
1	3.141592
2	"Lee "
3	"Hart"
4	"Bud "
5	"Levi"
6	"le "
7	2
8	mov r0, #0
9	mov r1, #10
10	add r0, r0, r1
11	subs r1, r1, #1
12	bne .-2
13	b .
14	0x00004020
15	0x20090001

A Bit of History

There is a commonly recurring debate over whether "data" and "instructions" should be mixed. Leads to two common flavors of computer architectures



Harvard Architecture



Howard Aiken: Architect of the Harvard Mark 1

Instructions and data do not/should not interact. They can have different "word sizes" and exist in different "address spaces"

- **Advantages**:
  - No self-modifying code (a common hacker trick)
  - Optimize word-lengths of instructions for control and data for applications
  - Higher Throughput (i.e. you can fetch data and instructions from their memories simultaneously)
- **Disadvantages**:
  - The H/W designer decides the trade-off between how big of a program and how large are data
  - Hard to write "Native" programs that generate new programs (i.e. assemblers, compilers, etc.)
  - Hard to write "Operating Systems" which are programs that at various points treat other programs as data (i.e. loading them from disk into memory, swapping out processes that are idle)

Von Neumann Architecture

John Von Neumann: Proponent of unified memory architecture

Instructions are just a type of data that share a common "word size" and "address space" with other types.

- Most common model used today, and what we assume in 411
- **Advantages**:
  - S/W designer decides how to allocate memory between data and programs
  - Can write programs to create new programs (assemblers and compilers)
  - Programs and subroutines can be loaded, relocated, and modified by other programs (dangerous, but powerful)
- **Disadvantages**:
  - Word size must suit both common data types and instructions
  - Slightly lower performance due to memory bottleneck (mediated in modern computers by the use of separate program and data caches)
  - We need to be very careful when treading on memory. Folks have taken advantage of the program-data unification to introduce viruses.

Next Time

- We examine an instruction set in depth
  - Assembly language
  - Machine language