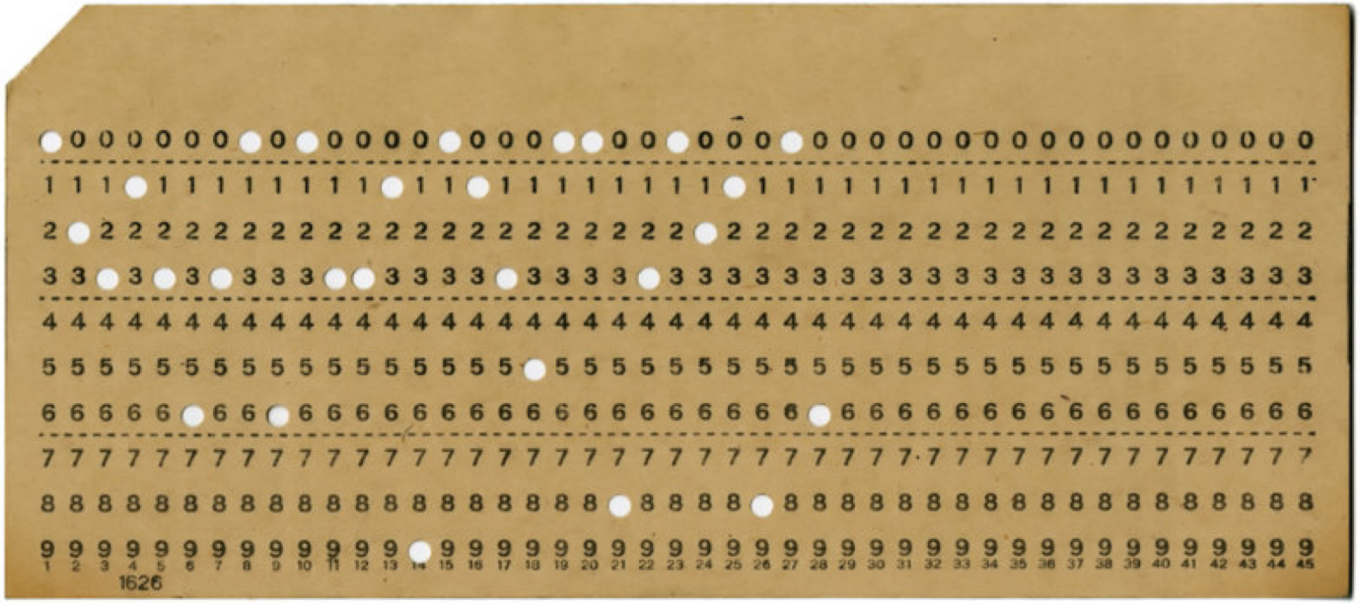
Computer Architecture, Fall 2019

Course Introduction, Number Representation



Agenda

- Course Overview
- Number Representation
  - Number Bases
  - Signed Representations
  - Overflow
  - Sign Extension

My Computers

Late 1980's

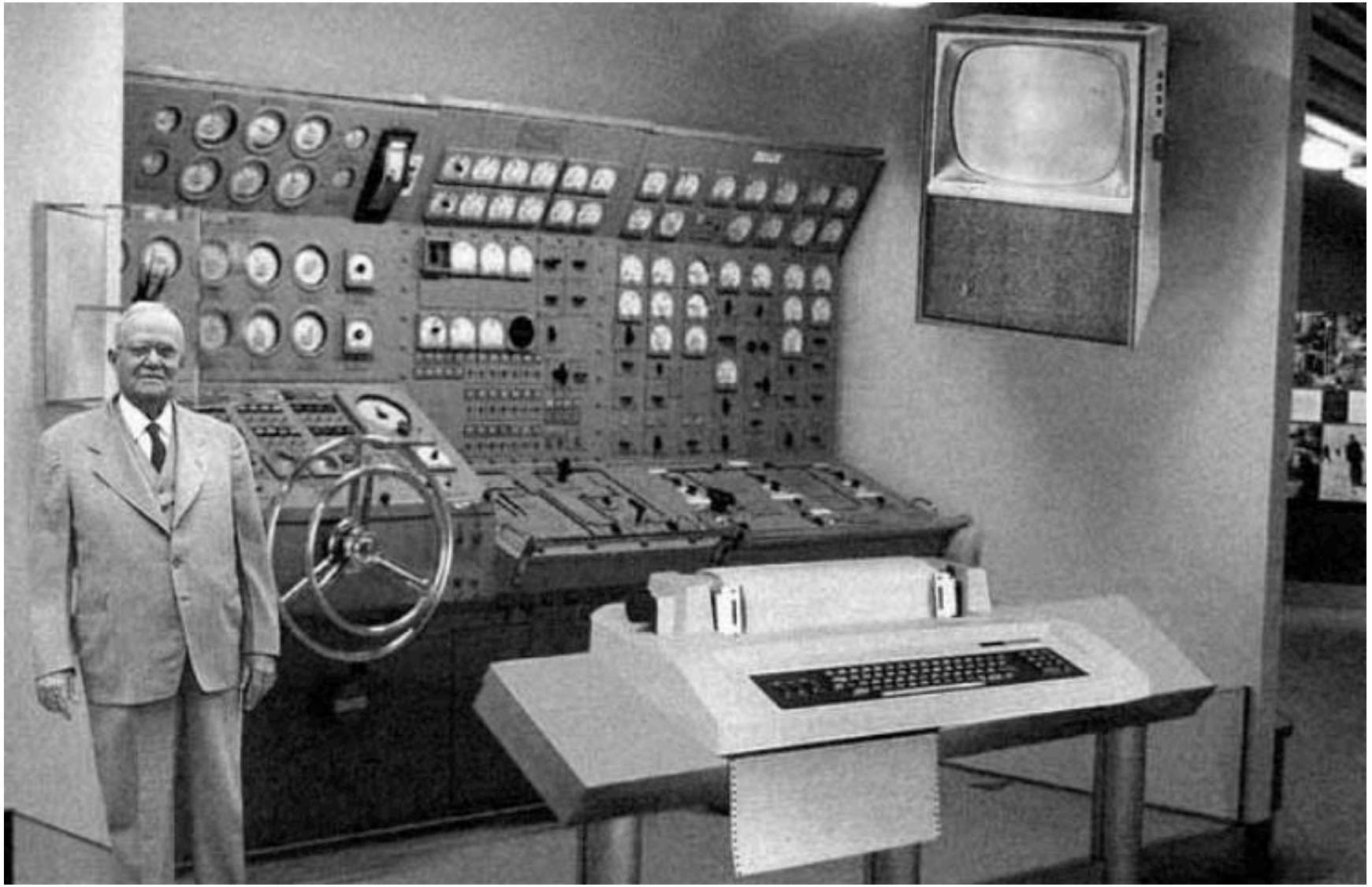
- CPU: Zilog Z80 3.58 MHz
- ROM: 32KB
  (BIOS 16KB + MSX Basic v1.0 16KB)
- RAM: 8KB

Fast forward ~30 years:

Present 2018

- CPU: Intel Xeon 40 CPUs 2.4 GHz
- ROM: SSD 8TB
- RAM: 256 GB
- 10 NVIDIA GTX Titan XP GPUs
- 10 Gbps network

Computer Architecture Course in Old School

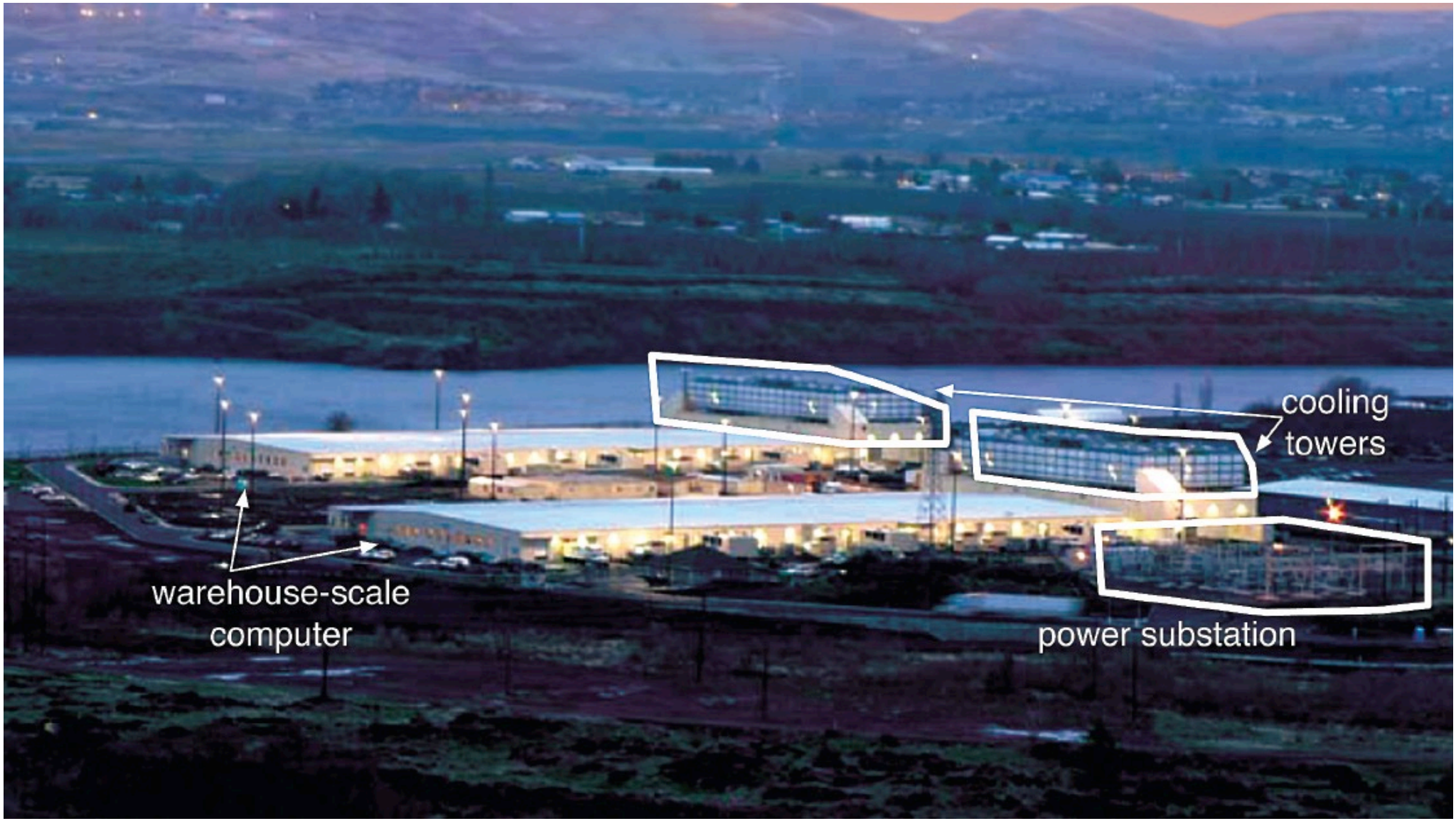


Computer Architecture Course in New School (1/3)

Personal Mobile Devices

Network Edge Devices

Computer Architecture Course in New School (2/3)



Computer Architecture Course in New School (3/3)

New-School Machine Structures

- Parallel Requests
  - Assigned to computer
  - e.g., Search “cats”
- Parallel Threads
  - Assigned to core
  - e.g., Lookup, Ads
- Parallel Instructions
  - >1 instruction @ one time
  - e.g., 5 pipelined instructions
- Parallel Data
  - >1 data item @ one time
  - e.g., Add of 4 pairs of words
- Hardware descriptions

All gates functioning in parallel at same time

Software

Hardware

Harness Parallelism & Achieve High Performance

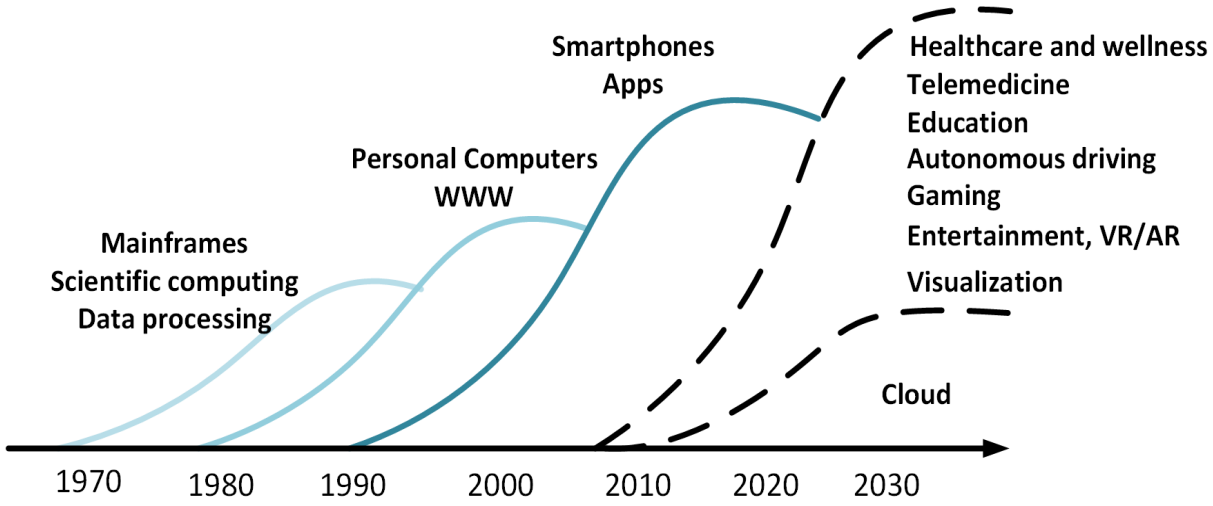
Warehouse-Scale Computer

Smart Phone

Six Great Ideas in Computer Architecture

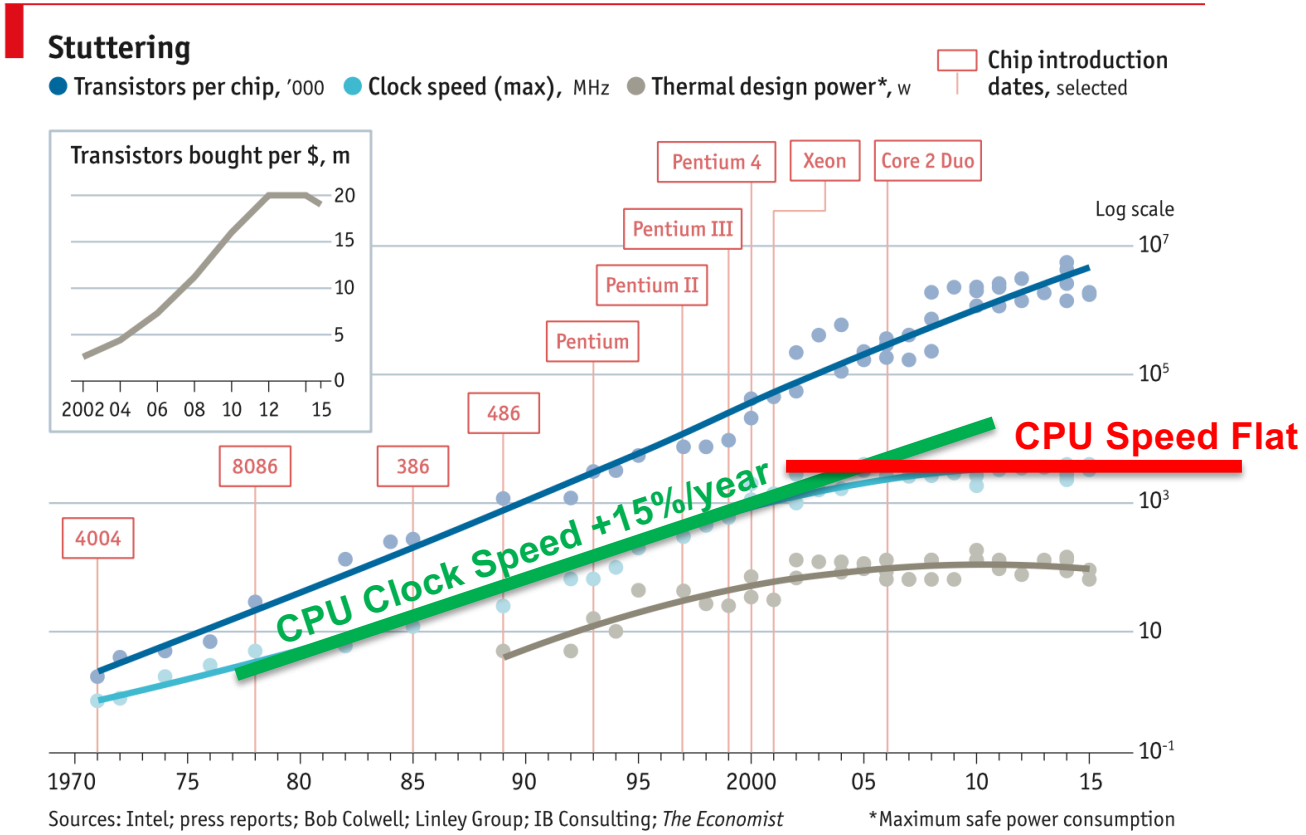
1) Abstraction
2) Technology Trends
3) Principle of Locality/Memory Hierarchy
4) Parallelism
5) Performance Measurement & Improvement
6) Dependability via Redundancy

Why is Architecture Exciting Today?



- Number of deployed devices continues growing, but no single killer app
  - –Diversification of needs, architectures

Why is Architecture Exciting Today?

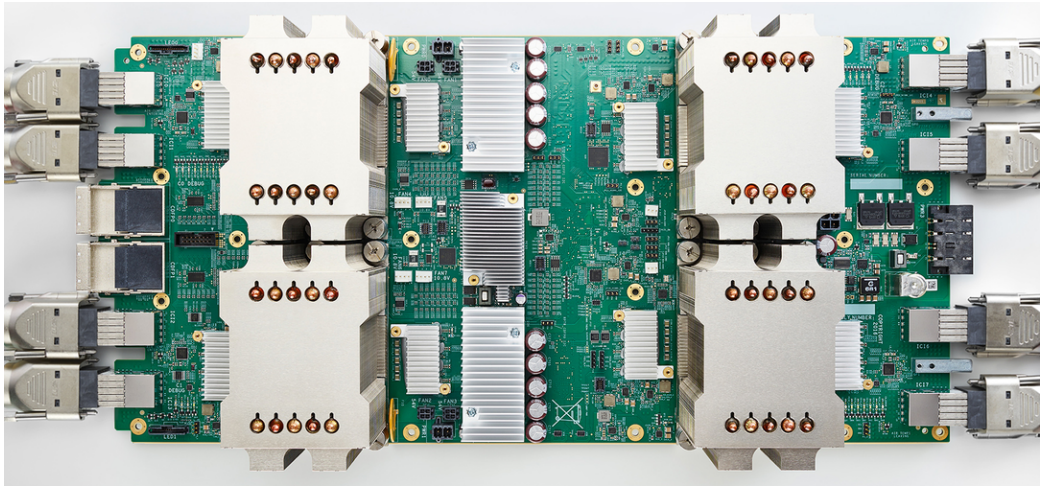


Old Conventional Wisdom

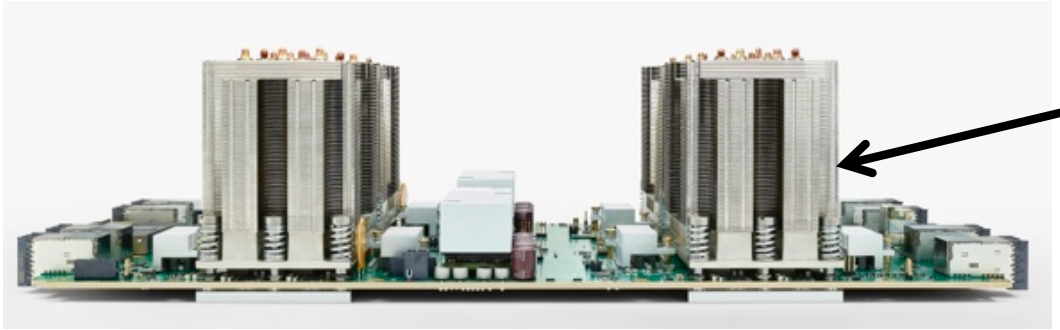
- Faster, cheaper, lower-power general-purpose computers each year
- In glory days, 1%/week performance improvement!

- Dumb to compete by designing parallel or specialized computers
- By time you've finished design, next generation of general-purpose will beat you

New Conventional Wisdom



Google TPU2
Specialized Engine for neural network training
Deployed in cloud
45 TFLOPS/chip

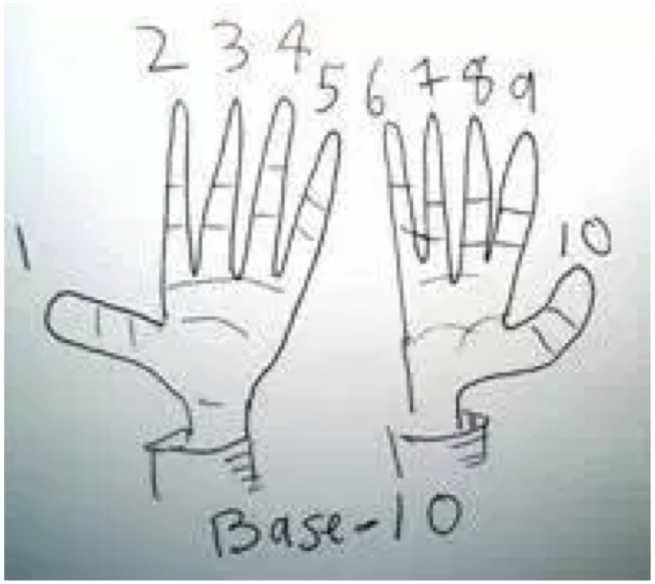


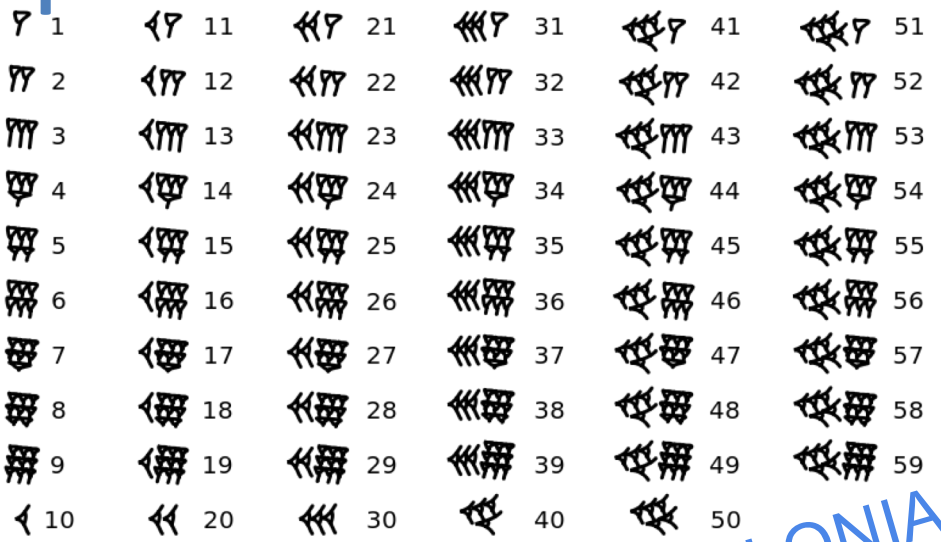
Serious heatsinks!

Agenda

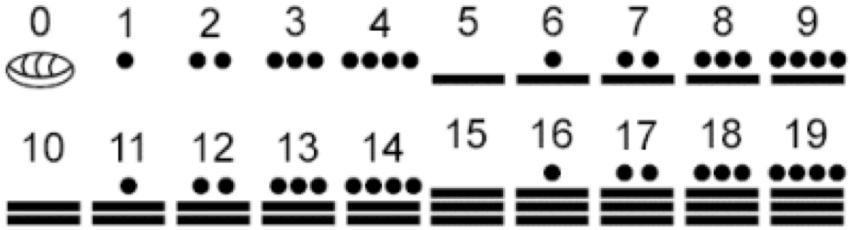
- Course Overview
- Number Representation
  - Number Bases
  - Signed Representations
  - Overflow
  - Sign Extension

Number Representation





BABYLONIANS



Example:

28 = (1 x 20) + 8 =

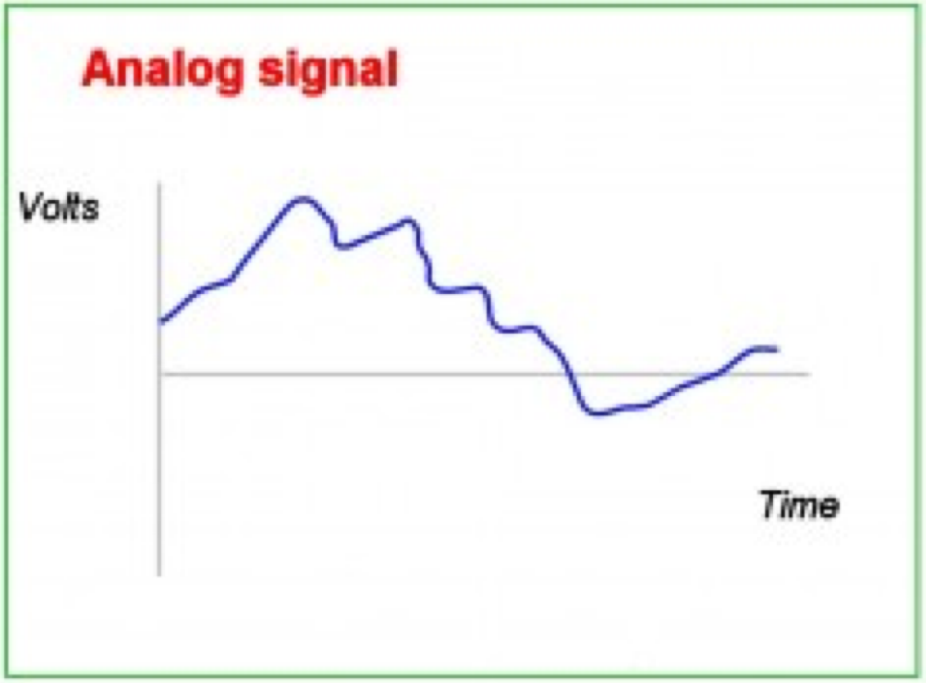
433 = (1 x 400) + (1 x 20) + 13 =

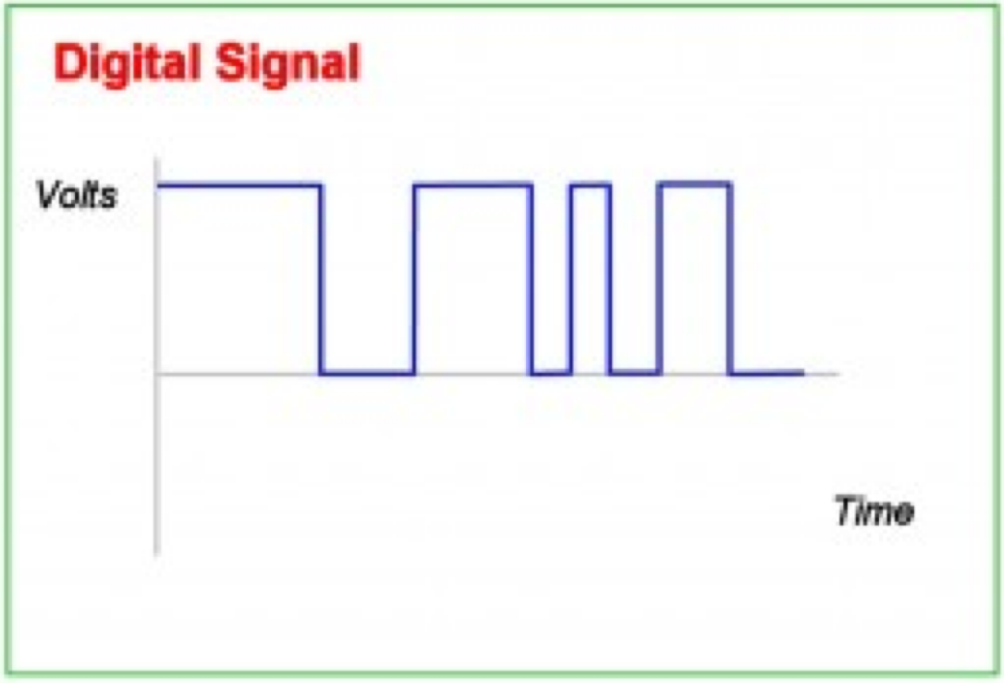
MAYANS

Number Representation

- Numbers are an abstract concept!
  - Recognize that these are *arbitrary* symbols
- Inside a computer, everything stored as a sequence of 0's and 1's (bits)
  - Even this is an abstraction!
- How do we represent numbers in this format?
  - Let's start with integers

Computers: Base 2





Number Bases

- **Key terminology:** digit (*d*) and base (*B*)
- In base B, each digit is one of B possible symbols
- Value of *i*-th digit is $d \times B^i$ where *i* starts at 0 and increases from right to left
  - *n* digit number $d_{n-1}d_{n-2} \ldots d_1d_0$
  - value = $d_{n-1} \times B^{n-1} + d_{n-2} \times B^{n-2} + \ldots + d_1 \times B^1 + d_0 \times B^0$
- Notation: Base is indicated using either a prefix or a subscript

Commonly Used Number Bases

- **Decimal** (base 10)
  - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Notation: 9472_{ten} = 9472
- **Binary** (base 2)
  - Symbols: 0, 1
  - Notation: 101011_{two} = 0b101011
- **Hexadecimal** (base 16)
  - Symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - Notation: $2A5D_{hex}$ = 0x2A5D

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

Number Base Examples

- Examples:

3 2 1 0

$\mathbf{9472}_{ten}$ = 9000 + 400 + 70 + 2

9x1000 + 4x100 + 7x10 + 2x1

$9x10^3 + 4x10^2 + 7x10^1 + 2x10^0$

$\mathbf{9472}_{ten} = 2x16^3 + 5x16^2 + 0x16^1 + 0x16^0$

$= \mathbf{2500}_{hex}$

0xA15 = 0b 1010 0001 0101

Bits Can Represent Anything

- *n* digits in base *B* can represent at most B^n things!
  - Each of the *n* digits is one of *B* possible symbols
  - Have more things? Add more digits!
- Example: Logical values (1 bit) – 0 is False, 1 is True
- Example: Characters
  - 26 letters require 5 bits ($2^5 = 32 > 26$)
- Example: Students in this class (6 bits)
- For convenience, can group into *nibbles* (4 bits) and *bytes* (8 bits)

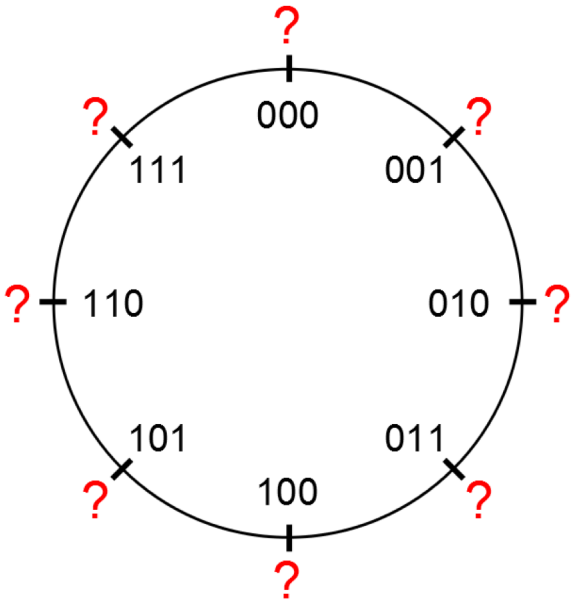
So What Number Is It Anyway?

- Here we are using binary bit patterns to represent numbers
  - Strictly speaking they are called *numerals* and have no meaning until you interpret them
  - Is CAB a word (taxi) or a number (3243_{ten})?
  - Is 0x999999 a number or a color (RGB)?
- Keep in mind that the same bit pattern will mean different things depending on how you choose to interpret it

Numbers in a Computer

- Numbers really have ∞ digits, but hardware can only store a finite number of them (fixed)
  - Usually ignore *leading zeros*

- "Circle" of binary numerals:
  (3-bit example)



Unsigned Integers

Represent only non-negative (unsigned) integers:

$0000_{two} = 0_{ten}$

$0001_{two} = 1_{ten}$

. . .

$0111_{two} = 7_{ten}$

$1000_{two} = 8_{ten}$

. . .

$1110_{two} = 14_{ten}$

$1111_{two} = 15_{ten}$

Zero?

$0…0_{two} = 0_{ten}$

Most neg number?

$0…0_{two} = 0_{ten}$

Most pos number?

$1…1_{two} = (2^n-1)_{ten}$

Increment?

Signed Integers

- *n* bits can represent 2^n different things
  - Ideally, want the range evenly split between positive and negative
- How do we want to encode zero?
- Can we encode them in such a way that we can use the same hardware regardless of whether the numbers are signed or unsigned?
  - e.g. incrementing pos/neg numbers

Sign and Magnitude

"first" bit gives sign, rest treated as unsigned (magnitude):

$000_{two} = +0_{ten}$
$001_{two} = +1_{ten}$
$010_{two} = +2_{ten}$
$011_{two} = +3_{ten}$

$100_{two} = -0_{ten}$
$101_{two} = -1_{ten}$
$110_{two} = -2_{ten}$
$111_{two} = -3_{ten}$

Zero?

$0\ldots0_{two}$ and $10\ldots0_{two} = \pm 0_{ten}$

TWO ZEROS?!

Most pos number?

$01\ldots1_{two} = (2^{(n-1)}-1)_{ten}$

Most neg number?

$1\ldots1_{two} = -(2^{(n-1)}-1)_{ten}$

Increment?

Biased Notation

Like unsigned, but "shifted" so zero is (roughly) in the middle: (value = "unsigned value" - bias)

PRO $000_{two} = -3_{ten}$

$001_{two} = -2_{ten}$

$010_{two} = -1_{ten}$

CON $011_{two} = 0_{ten}$ ← Conventional Bias: $(2^{n-1}-1)$

$100_{two} = +1_{ten}$

$101_{two} = +2_{ten}$

$110_{two} = +3_{ten}$

$111_{two} = +4_{ten}$ PRO

Zero?

$01\ldots1_{two} = 0_{ten}$

Most neg number?

$0\ldots0_{two} = -(2^{(n-1)}-1)_{ten}$

Most pos number?

$1\ldots1_{two} = 2^{(n-1)}{}_{ten}$

Increment?

PRO just like unsigned (sorta)

One's Complement

New negation procedure – complement the bits:

$000_{two} = +0_{ten}$
$001_{two} = +1_{ten}$
$010_{two} = +2_{ten}$
$011_{two} = +3_{ten}$

$100_{two} = -3_{ten}$
$101_{two} = -2_{ten}$
$110_{two} = -1_{ten}$
$111_{two} = -0_{ten}$

Zero?

$0\ldots0_{two}$ and $1\ldots1_{two} = \pm0_{ten}$

TWO ZEROS AGAIN……

Most neg number?

$10\ldots0_{two} = -(2^{(n-1)}-1)_{ten}$

Most pos number?

$01\ldots1_{two} = (2^{(n-1)}-1)_{ten}$

Increment?

Two's Complement

Like One's Complement, but "shift" negative #s by 1:

$000_{two} = +0_{ten}$
$001_{two} = +1_{ten}$
$010_{two} = +2_{ten}$
$011_{two} = +3_{ten}$

$100_{two} = -4_{ten}$
$101_{two} = -3_{ten}$
$110_{two} = -2_{ten}$
$111_{two} = -1_{ten}$

Zero?

$0\ldots0_{two} = 0_{ten}$

Most neg number?

$10\ldots0_{two} = -2^{(n-1)}{}_{ten}$

Most pos number?

$01\ldots1_{two} = (2^{(n-1)}-1)_{ten}$

Increment?

Two's Complement Summary

- Used by all modern hardware
- Roughly evenly split between positive and negative
  - One more negative # because positive side has 0
- Can still use MSB as sign bit
- To negate: Flip the bits and add one
  - Example: +7 = 0b 0000 0111, -7 = 0b 1111 1001

Two's Complement Review

- Suppose we had 5 bits. What integers can be represented in two's complement?

  (A) -31 to +31 ← need 6 bits

  (B) -15 to +15 ← one's complement

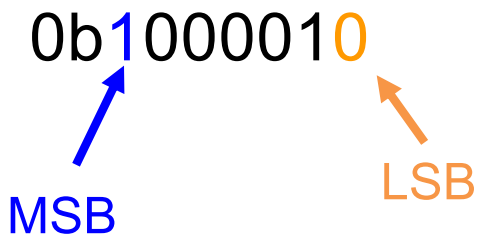
  (C) 0 to +31 ← unsigned

  (D) -16 to +15 ← two's complement

  (E) -32 to +31 ← need 6 bits

Numbers in a Computer

- Numbers really have ∞ digits, but hardware can only store a finite number of them (fixed)
  - Usually ignore *leading zeros*
  - Leftmost is *most significant bit* (MSB)
  - Rightmost is *least significant bit* (LSB)



Overflow

- Overflow is when the result of an arithmetic operation can't be represented by the hardware bits
  - i.e. the result is mathematically incorrect
- Examples:
  - Unsigned: $0b1\ldots1 + 1_{10} = 0b0\ldots0 = 0$?
  - Two's: $0b01\ldots1 + 1_{10} = 0b10\ldots0 = -2^{(n-1)}{}_{ten}$?

Sign Extension

- Want to represent the same number using more bits than before
  - Easy for positive #s (add leading 0's), more complicated for negative #s
  - Sign and magnitude: add 0's *after* the sign bit
  - One's complement: copy MSB
  - Two's complement: copy MSB
- Example:
  - Sign and magnitude: 0b 11 = 0b 1001
  - One's/Two's complement: 0b 11 = 0b 1111

Summary

- Number Representation: How to represent positive and negative integers using binary
  - Unsigned: Interpret numeral in base 2
  - Signed: Two's Complement
  - Biased: Subtract bias
  - Sign extension must preserve signed number