

CS61C – Machine Structures

Lecture 1.1.1

Introduction and Number Representation

2004-06-21

Kurt Meinz

CS61C www page
`www-inst.eecs.berkeley.edu/~cs61c/`



Are Computers Smart?

◦ To a programmer:

- **Very complex operations/functions:**
 - `(map (lambda (x) (* x x)) '(1 2 3 4))`
- **Automatic memory management:**
 - `List l = new List;`
- **“Basic” structures:**
 - Integers, reals, characters, plus, minus, print commands



Are Computers Smart?

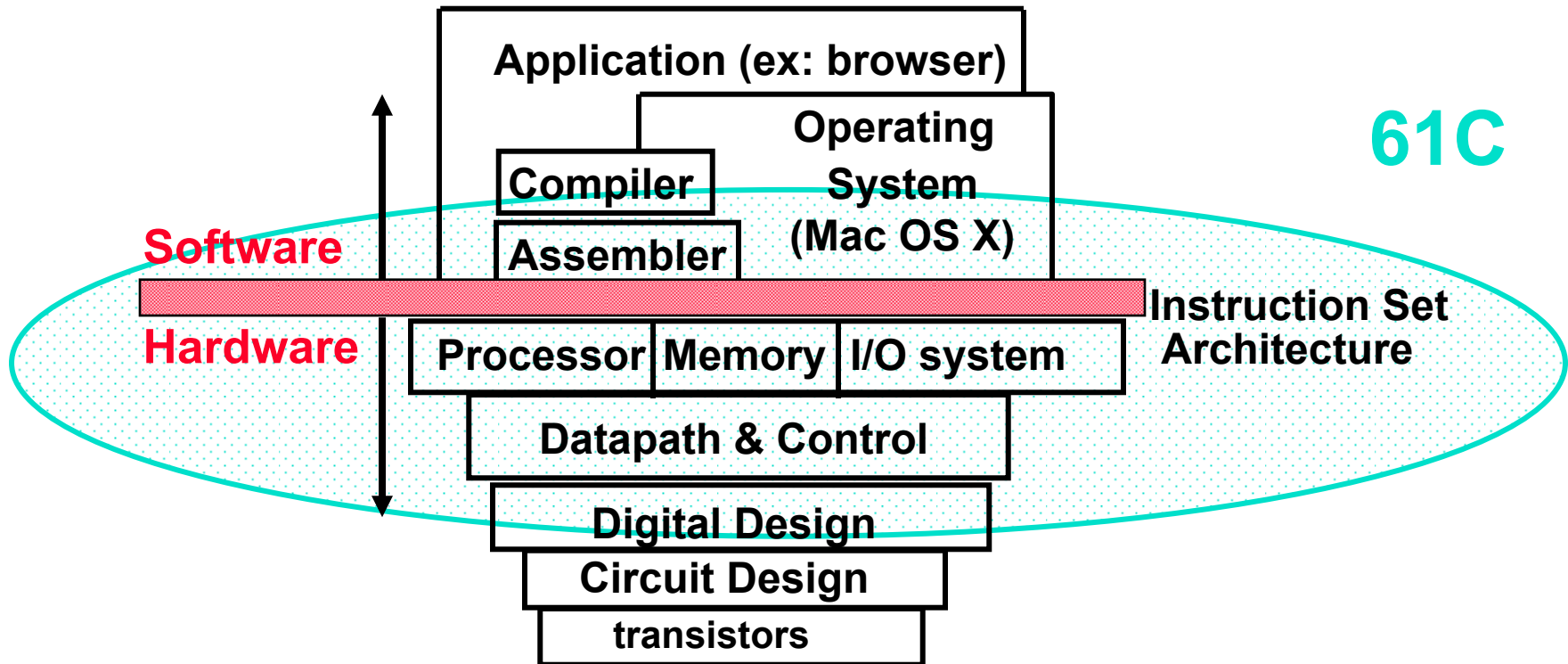
° In real life:

- Only a handful of operations:
 - {and, or, not} or {nand, nor}
- No memory management.
- Only 2 values:
 - {0, 1} or {hi, lo} or {on, off}
 - 3 if you count <undef>



Are Computers Smart?

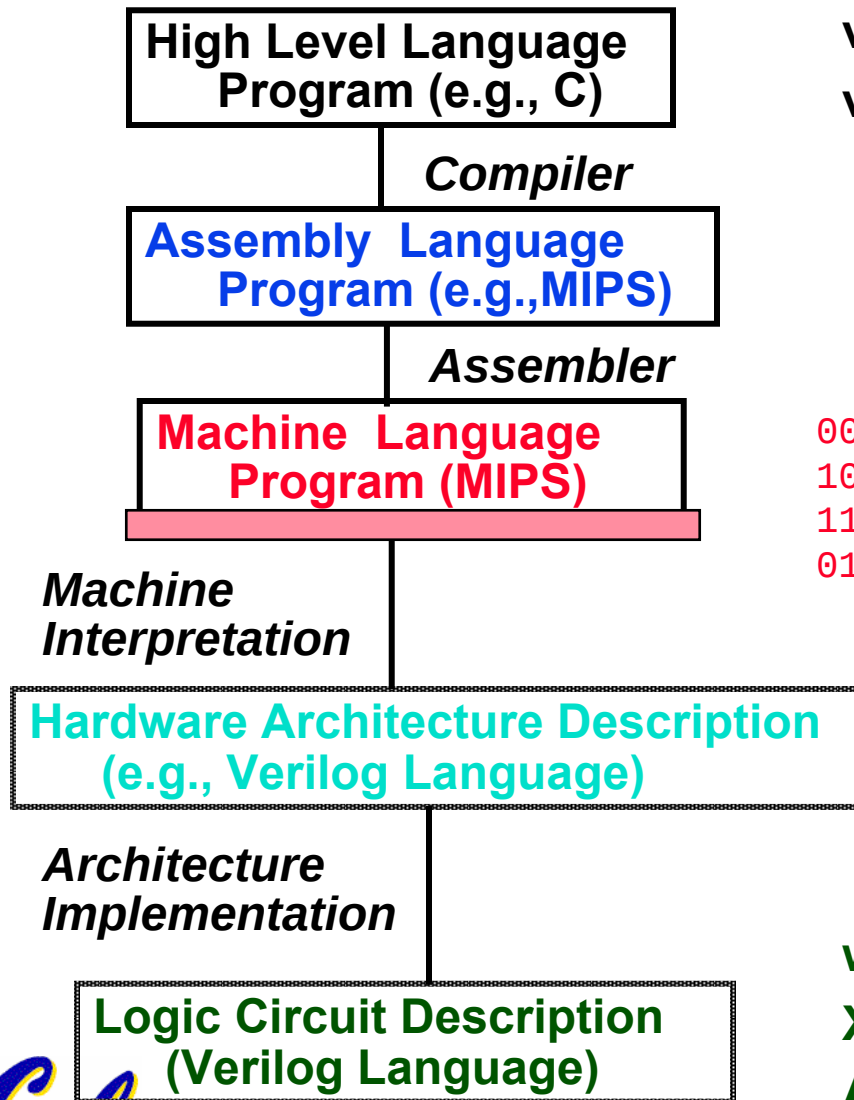
“61C is about complexity and performance at the near-machine level”



i.e. ABSTRACTION!
(at the machine and near-machine level)



61C Levels of Representation



```
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

```
lw $t0, 0($2)  
lw $t1, 4($2)  
sw $t1, 0($2)  
sw $t0, 4($2)
```

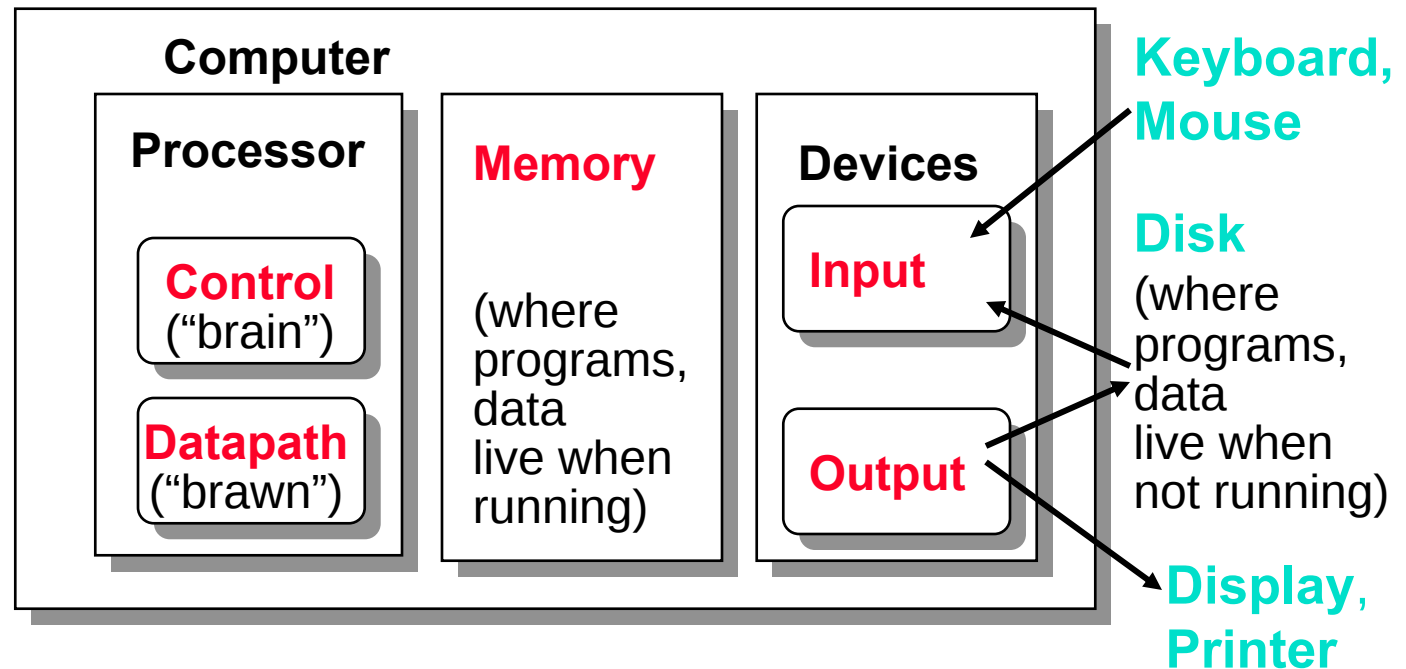
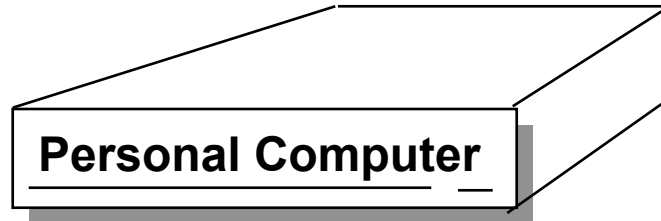
```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```

```
wire [31:0] dataBus;  
regFile registers (databus);  
ALU ALUBlock (inA, inB, databus);
```

```
wire w0;  
XOR (w0, a, b);  
AND (s, w0, a);
```



Anatomy: 5 components of any Computer



Overview of Physical Implementations

The hardware out of which we make systems.

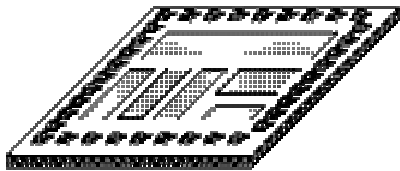
- **Integrated Circuits (ICs)**
 - **Combinational logic circuits, memory elements, analog interfaces.**
- **Printed Circuits (PC) boards**
 - **substrate for ICs and interconnection, distribution of CLK, Vdd, and GND signals, heat dissipation.**
- **Power Supplies**
 - **Converts line AC voltage to regulated DC low voltage levels.**
- **Chassis (rack, card case, ...)**
 - **holds boards, power supply, provides physical interface to user or other systems.**



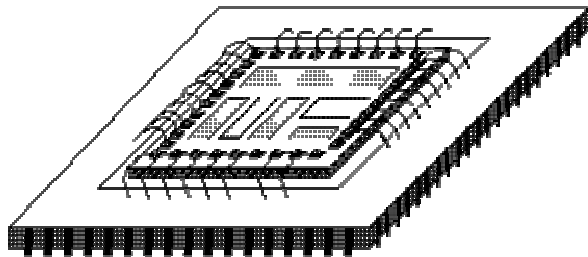
Connectors and Cables.

Integrated Circuits (2003 state-of-the-art)

Bare Die



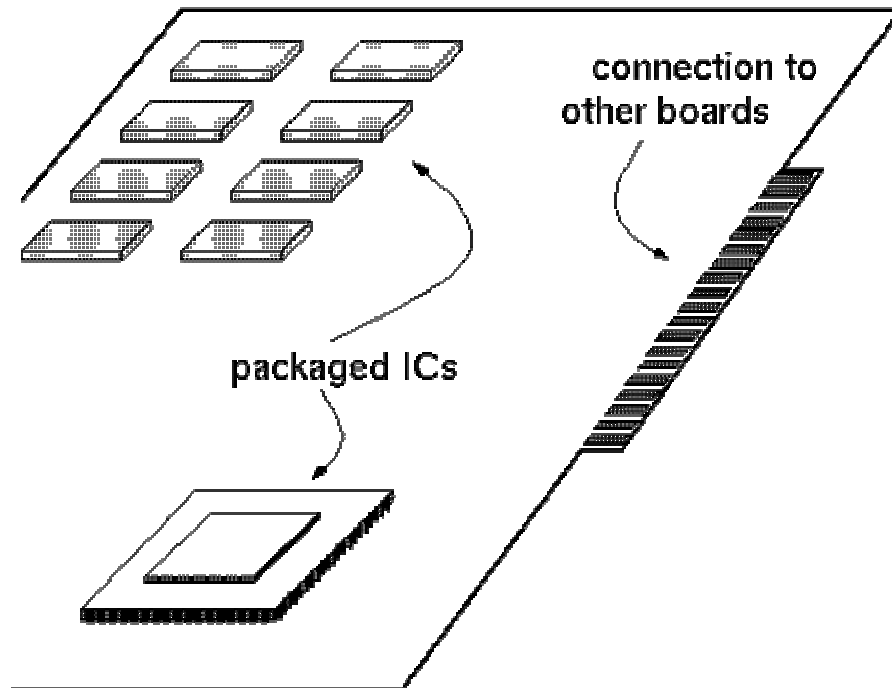
Chip in Package



- Primarily Crystalline Silicon
- 1mm - 25mm on a side
- 2003 - feature size $\sim 0.13\mu\text{m} = 0.13 \times 10^{-6} \text{ m}$
- 100 - 400M transistors
- (25 - 100M “logic gates”)
- 3 - 10 conductive layers
- “CMOS” (complementary metal oxide semiconductor) - most common.

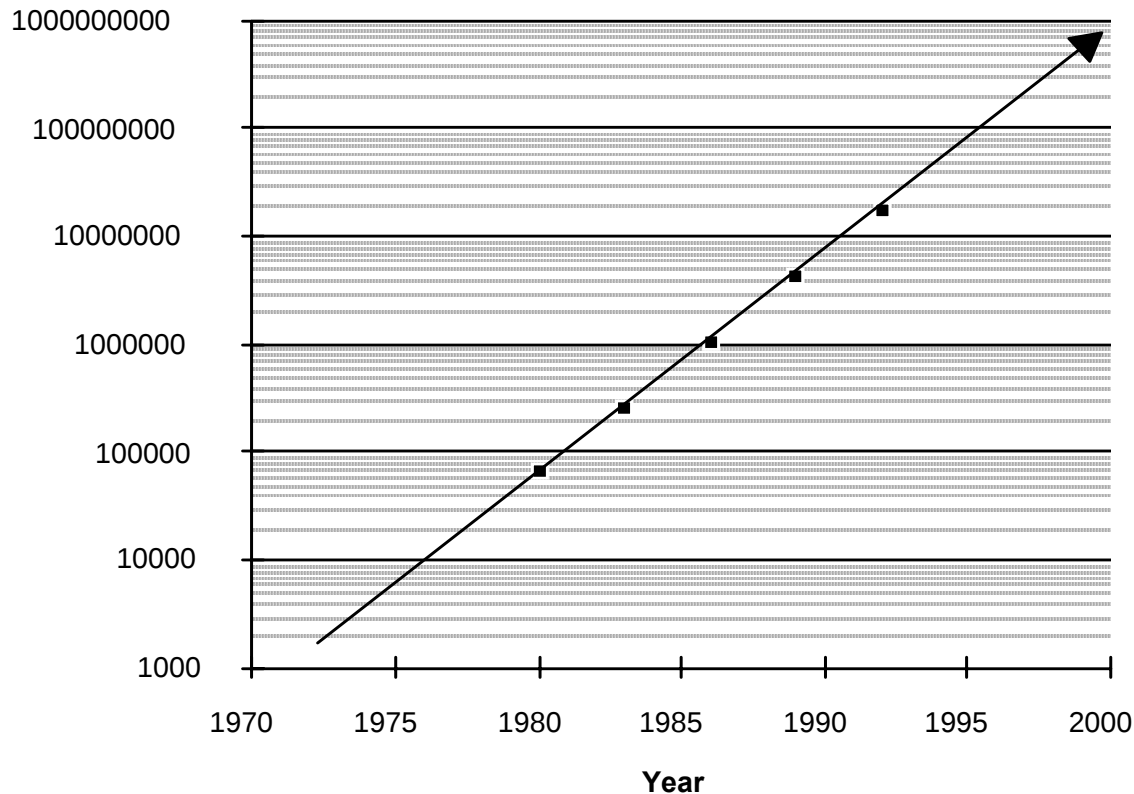
- Package provides:
 - spreading of chip-level signal paths to board-level
 - heat dissipation.
- Ceramic or plastic with gold wires.

Printed Circuit Boards



- fiberglass or ceramic
- 1-20 conductive layers
- 1-20in on a side
- IC packages are soldered down.

Technology Trends: Memory Capacity (Single-Chip DRAM)

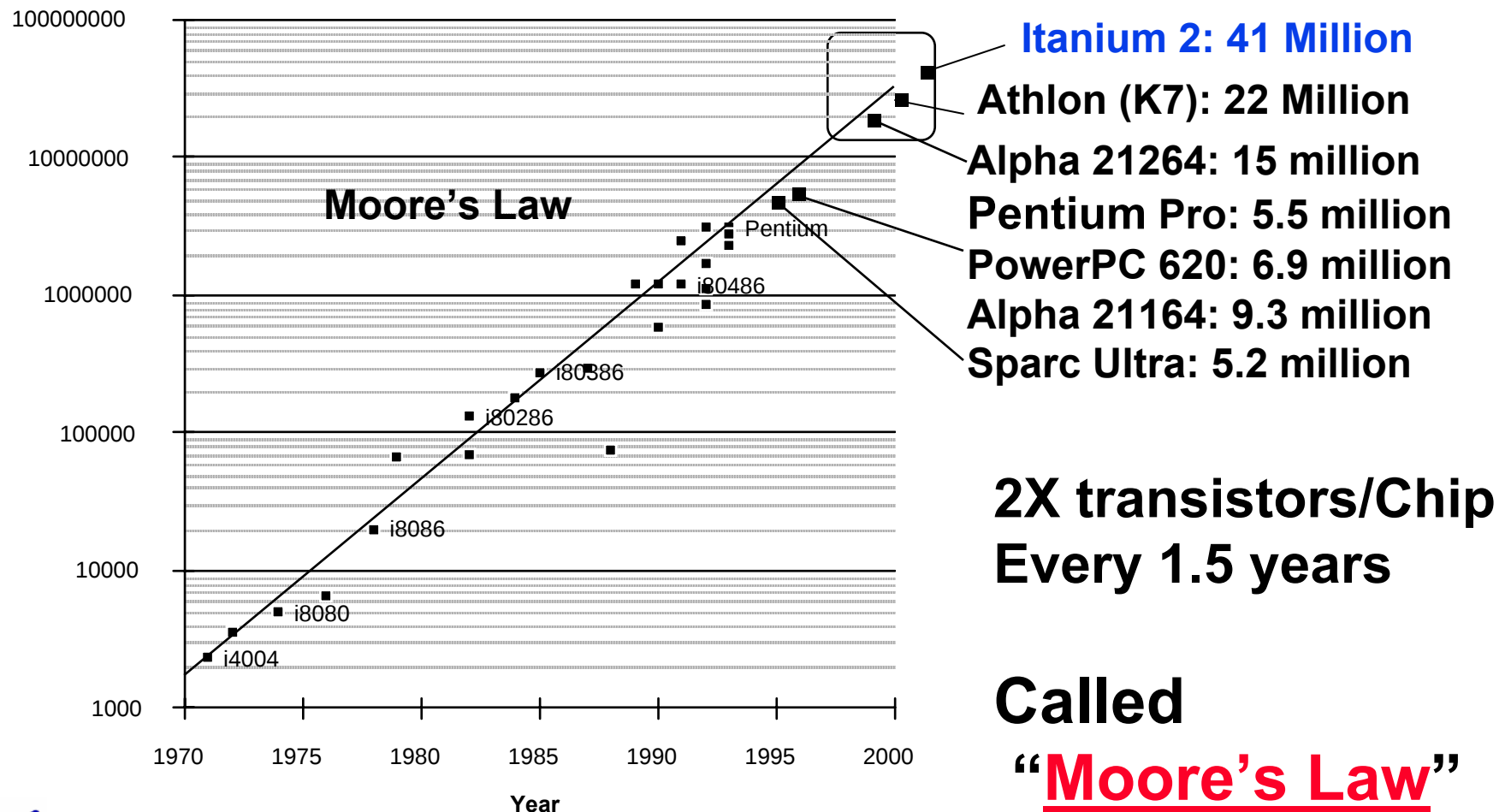


year	size (Mbit)
1980	0.0625
1983	0.25
1986	1
1989	4
1992	16
1996	64
1998	128
2000	256
2002	512

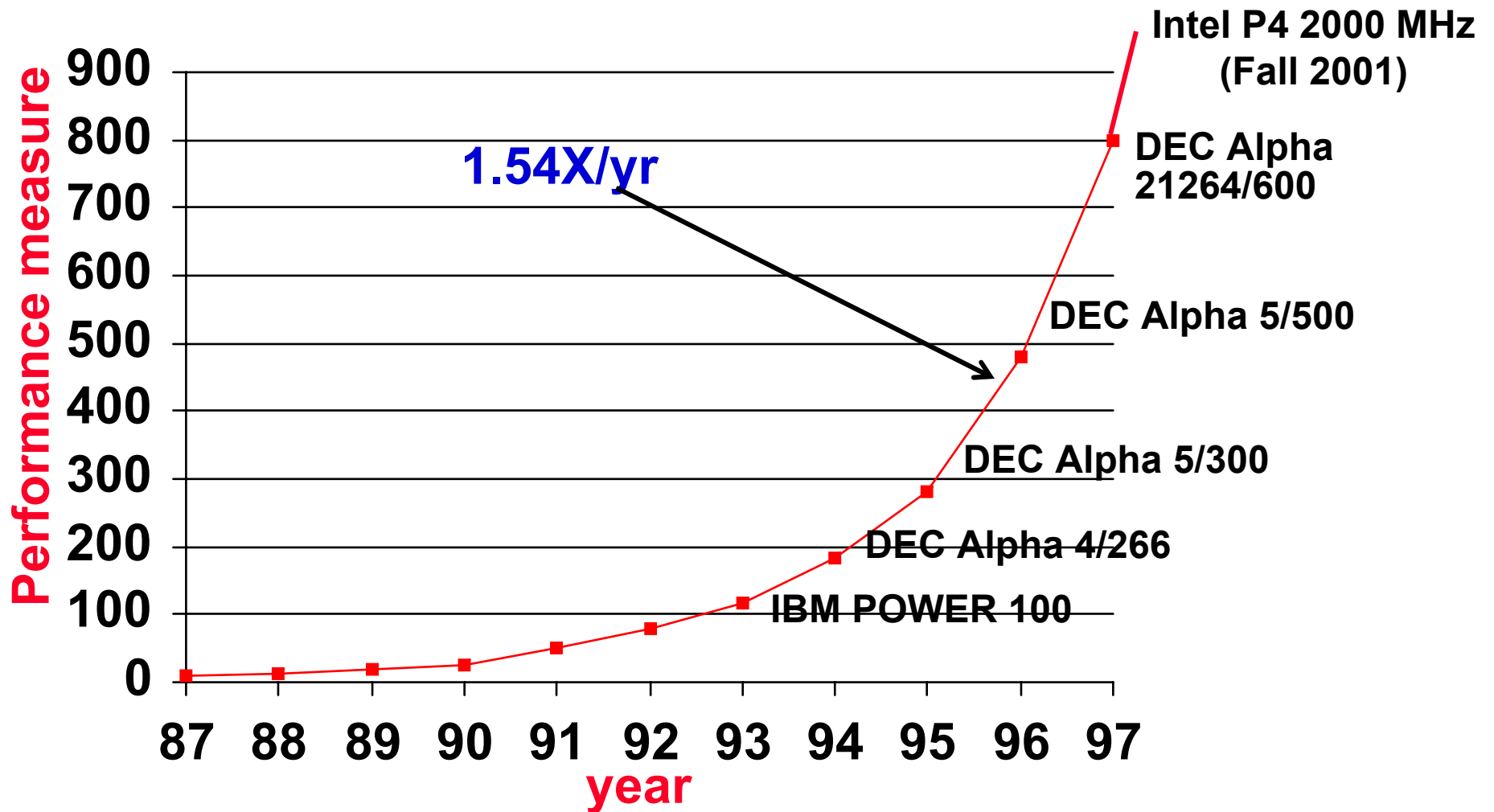
- Now 1.4X/yr, or 2X every 2 years.
- 8000X since 1980!



Technology Trends: Microprocessor Complexity



Technology Trends: Processor Performance



We'll talk about processor performance later on...



Computer Technology - Dramatic Change!

◦ Memory

- DRAM capacity: 2x / 2 years (since '96);
64x size improvement in last decade.

◦ Processor

- Speed 2x / 1.5 years (since '85);
100X performance in last decade.

◦ Disk

- Capacity: 2x / 1 year (since '97)
250X size in last decade.



Computer Technology - Dramatic Change!

◦ State-of-the-art PC when you graduate: (at least...)

- Processor clock speed: 5000 **Mega**Hertz
(5.0 **Giga**Hertz)
- Memory capacity: 4000 **Mega**Bytes
(4.0 **Giga**Bytes)
- Disk capacity: 2000 **Giga**Bytes
(2.0 **Tera**Bytes)
- New units! **Mega** => **Giga**, **Giga** => **Tera**

(**Tera** => **Peta**, **Peta** => **Exa**, **Exa** => **Zetta**
Zetta => **Yotta** = 10^{24})



CS61C: So what's in it for me?

- **Learn some of the big ideas in CS & engineering:**
 - **5 Classic components of a Computer**
 - **Data can be anything (integers, floating point, characters): a program determines what it is**
 - **Stored program concept: instructions just data**
 - **Principle of Locality, exploited via a memory hierarchy (cache)**
 - **Greater performance by exploiting parallelism**
 - **Principle of abstraction, used to build systems as layers**
 - **Compilation v. interpretation thru system layers**
 - **Principles/Pitfalls of Performance Measurement**



Others Skills learned in 61C

◦ Learning C

- If you know one, you should be able to learn another programming language largely on your own
- Given that you know C++ or Java, should be easy to pick up their ancestor, C

◦ Assembly Language Programming

- This is a skill you will pick up, as a side effect of understanding the Big Ideas

◦ Hardware design

- We think of hardware at the abstract level, with only a little bit of physical logic to give things perspective
- CS 150, 152 teach this



Course Lecture Outline

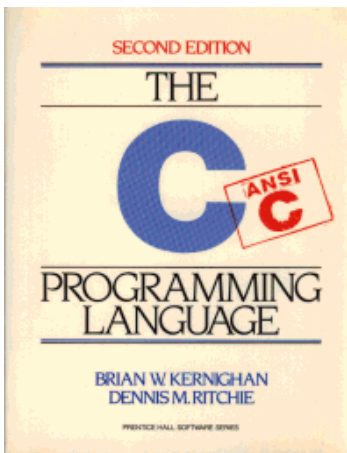
- **Number representations**
- **C-Language (basics + pointers)**
- **Storage management**
- **Assembly Programming**
- **Floating Point**
- **make-ing an Executable**
- **Caches**
- **Virtual Memory**
- **Logic Design**
- **Introduction to Verilog (HDL)**
- **CPU organization**
- **Pipelining**
- **Performance**
- **I/O Interrupts**
- **Disks, Networks**
- **Advanced Topics**



Texts



◦ Required: *Computer Organization and Design: The Hardware/Software Interface, Second Edition*, Patterson and Hennessy (COD)



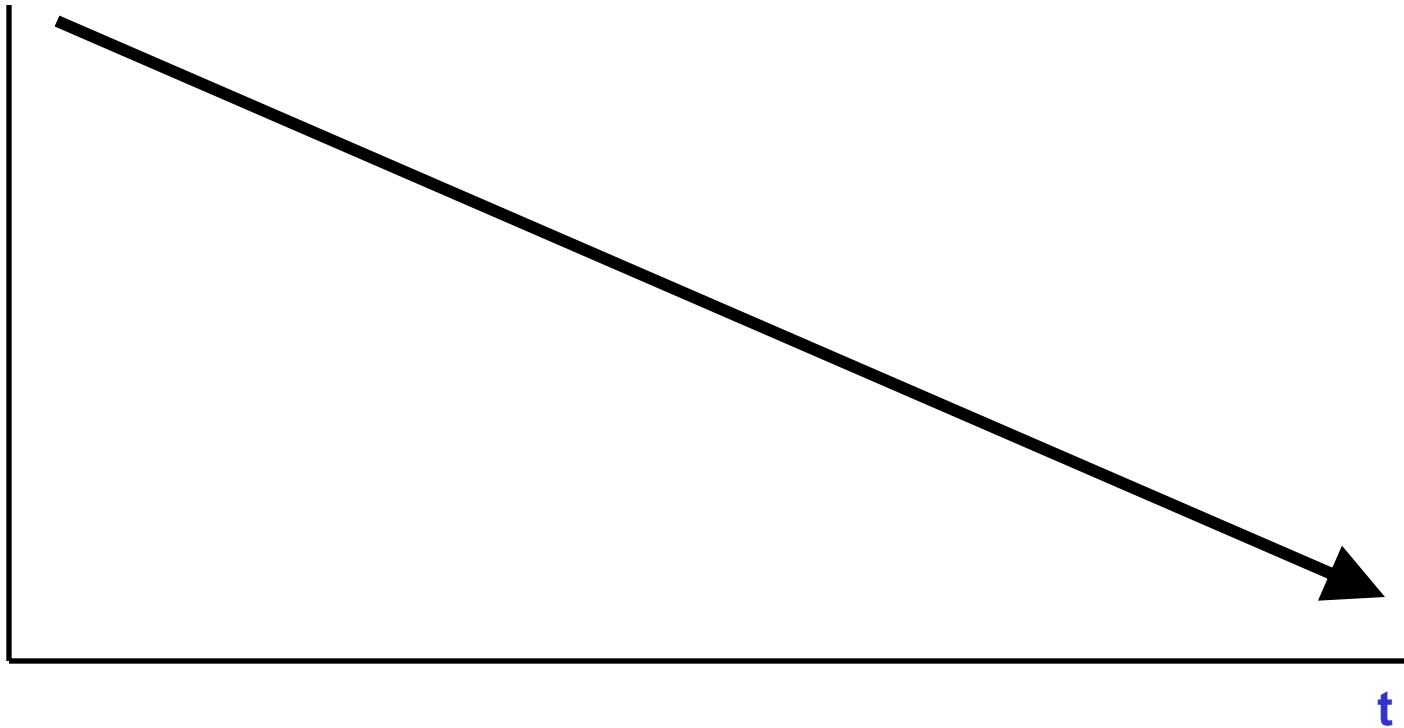
◦ Required: *The C Programming Language*, Kernighan and Ritchie (K&R), 2nd edition

◦ Reading assignments on web page

*Read P&H Chapter 1, Sections 4.1 & 4.2,
K&R Chapters 1-4 for Tuesday.*

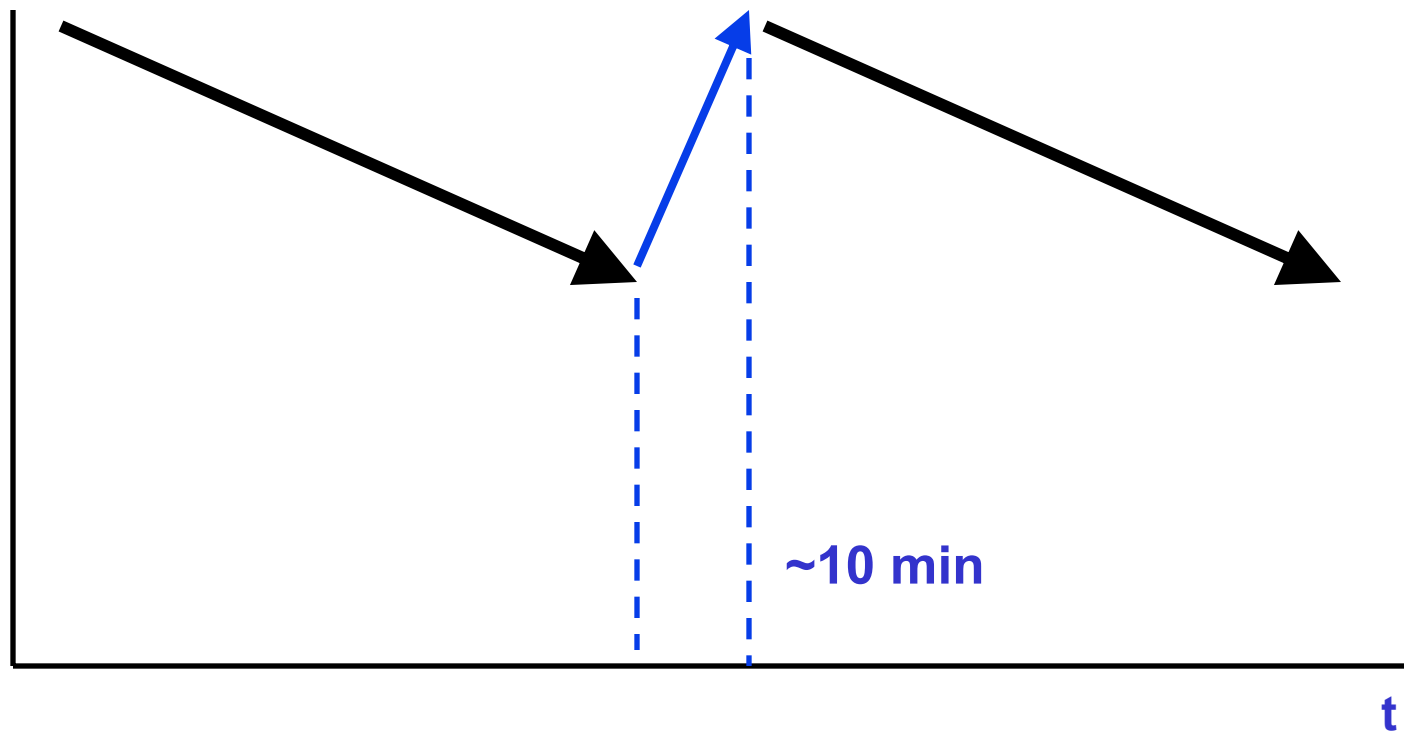


What is this?



Attention over time!

What is this?!

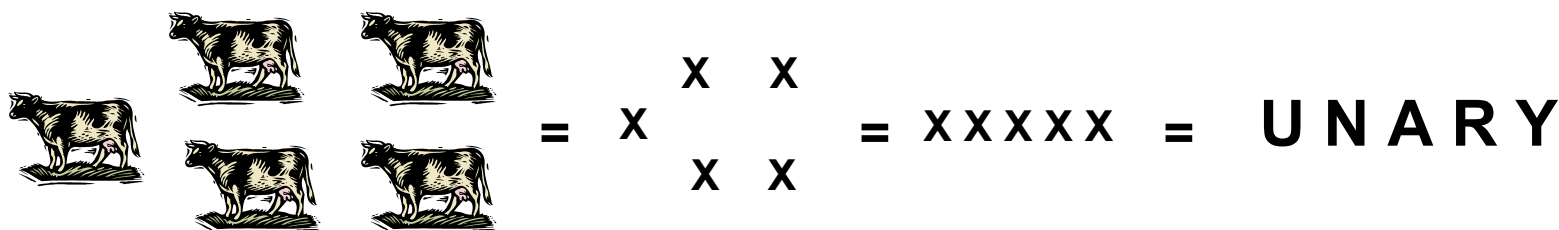


Attention over time!

Numbers!

What is the “natural” base for numbers?

2? 10? 16? 8? 137?



- * Set theory constructs numbers in a unary way.
- * Unary is inefficient in space.
- * All other bases are a compact way of re-presenting numbers....



For Example: Base 10

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Example:

3271 =

$$(3 \times 10^3) + (2 \times 10^2) + (7 \times 10^1) + (1 \times 10^0)$$

**(Unary would need 3271 digits to
express 3271_{b10} !)**



Numbers and Bases In General

- Number Base $B \Rightarrow B$ unique values per digit:
 - Base 10 (Decimal): $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 - Base 2 (Binary): $\{0, 1\}$ (or $\{7, 9\}$; $\{a, b\}$, ...)
- (Unsigned) Number representation:
 - $d_{31}d_{30} \dots d_1d_0$ is a 32 digit non-negative number
 - value = $d_{31} \times B^{31} + d_{30} \times B^{30} + \dots + d_1 \times B^1 + d_0 \times B^0$
- N-digit Base $B \rightarrow B^N$ unique values!
- All bases > 1 are equal in expressive power.
- But some bases are more equal than others ...



Numbers: positional notation

- Base 10

- Binary: {0,1} (In binary digits called bits’')

- $0b11010 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
 $= 16 + 8 + 2$
 $= 26_{\text{ten}}$

- Can we find a base that converts to binary easily?



Numbers: positional notation

- Base 10
- Binary: {0,1} (In binary digits called bits")
 - $0b11010 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
 $= 16 + 8 + 2$
 $= 26$
- Hexadecimal:
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
 - Normal digits + 6 more from the alphabet
 - In C, written as 0x... (e.g., 0xFAB5)
 - $(\text{'F'} * 16^3) + (\text{'A'} * 16^2) + (\text{'B'} * 16^1) + (5 * 16^0) = 64181b_{10}$



Hexadecimal Numbers: Base 16

° **Hexadecimal:** {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

• **Conversion: Binary $\Leftrightarrow$ Hex**

- 1 hex digit represents 16 decimal values
- 4 binary digits represent 16 decimal values

$\Rightarrow$ 1 hex digit replaces 4 binary digits

0x0	0000	0x4	0100	0x8	1000	0xc	1100
0x1	0001	0x5	0101	0x9	1001	0xd	1101
0x2	0010	0x6	0110	0xa	1010	0xe	1110
0x3	0011	0x7	0111	0xb	1011	0xf	1111

° **Example:**

• 1010 1100 0011 (binary) = 0x_____ ?



Decimal vs. Hexadecimal vs. Binary

Examples:

**1010 1100 0011b2
= 0xAC3**

**0b10111
= 0001 0111b2
= 0x17**

**0x3F9
= 11 1111 1001b2**

00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111



Which base do we use?

- **Decimal:** great for humans, especially when doing arithmetic
- **Hex:** if human looking at long strings of binary numbers, its much easier to convert to hex and look 4 bits/symbol
 - Terrible for arithmetic on paper
- **Binary:** what computers use; you will learn how computers do +, -, *, /
 - To a computer, numbers always binary
 - Regardless of how number is written:
 $32_{\text{ten}} == 32_{10} == 0x20 == 100000_2 == 0b100000$
 - Use subscripts “ten”, “hex”, “two” in book, slides when might be confusing



What to do with representations of numbers?

- **Just what we do with numbers!**

- Add them
- Subtract them
- Multiply them
- Divide them
- Compare them

$$\begin{array}{r} 11 \\ 1010 \\ + 0111 \\ \hline 10001 \end{array}$$

- **Example: $10 + 7 = 17$**

- ...so simple to add in binary that we can build circuits to do it!
- subtraction just as you would in decimal
- Comparison: How do you tell if $X > Y$?



How to Represent Negative Numbers?

- So far, **unsigned numbers**
- Obvious solution: define leftmost bit to be sign!
 - $0 \Rightarrow +$, $1 \Rightarrow -$
 - Rest of bits can be numerical value of number
- This is ~ how YOU do signed numbers in decimal!
- Representation called **sign and magnitude**
- MIPS uses 32-bit integers. $+1_{\text{ten}}$ would be:
0000 0000 0000 0000 0000 0000 0000 0001
- And -1_{ten} in sign and magnitude would be:
1000 0000 0000 0000 0000 0000 0000 0001



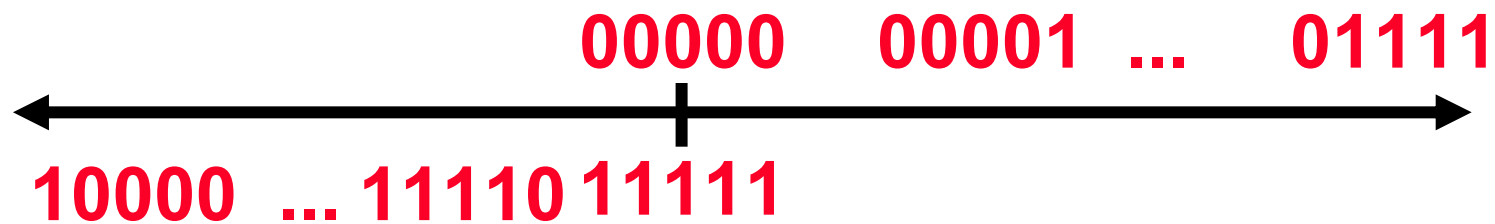
Shortcomings of sign and magnitude?

- Arithmetic circuit complicated
 - Special steps depending whether signs are the same or not
- Also, two zeros
 - $0x00000000 = +0_{\text{ten}}$
 - $0x80000000 = -0_{\text{ten}}$
 - What would two 0s mean for programming?
- Therefore sign and magnitude abandoned*



Another try: complement the bits

- Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
- Called One's Complement
- Note: positive numbers have leading 0s, negative numbers have leading 1s.



- What is -00000 ? Answer: 11111
- How many positive numbers in N bits?
- How many negative ones?



Shortcomings of One's complement?

- Arithmetic still a somewhat complicated.
- Still two zeros
 - $0x00000000 = +0_{\text{ten}}$
 - $0xFFFFFFFF = -0_{\text{ten}}$
- Although used for awhile on some computer products, one's complement was eventually abandoned because another solution was better....



Another Attempt ...

- Gedanken: Decimal Car Odometer

00003 → 00002 → 00001 → 00000 → 99999 → 99998

- Binary Odometer:

00011 → 00010 → 00001 → 00000 → 11111 → 11110

- With no obvious better alternative, pick representation that makes the math simple!

- 99999_{ten} == -1_{ten}

- 11111_{two} == -1_{ten} 11110_{two} == -2_{ten}

- This representation is Two's Complement

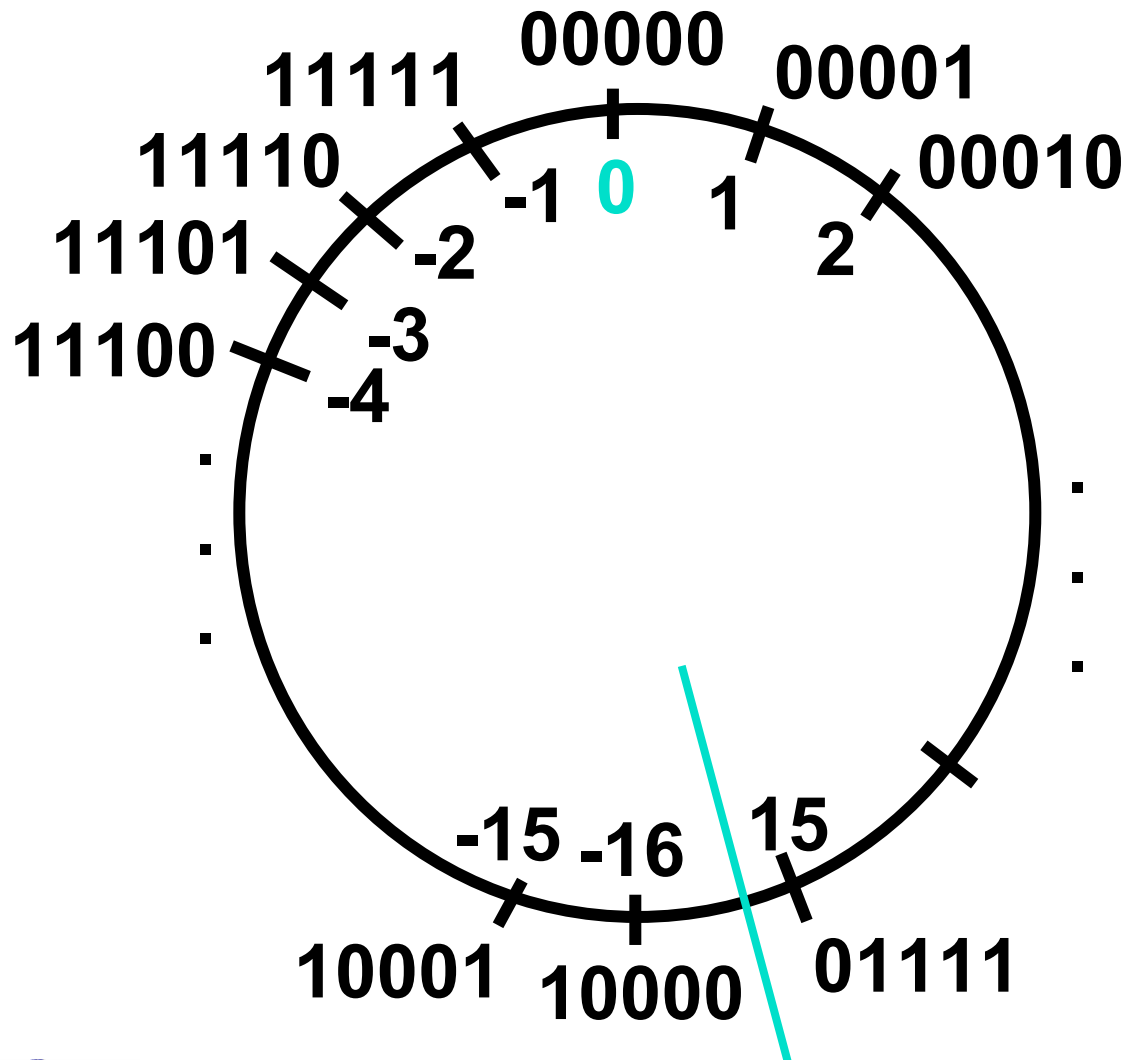


2's Complement Properties

- As with sign and magnitude, leading 0s $\Rightarrow$ positive, leading 1s $\Rightarrow$ negative
 - 000000...xxx is ≥ 0 , 111111...xxx is < 0
 - except 1...1111 is -1, not -0 (as in sign & mag.)
- Only 1 Zero!



2's Complement Number "line": $N = 5$



- 2^{N-1} non-negatives
- 2^{N-1} negatives
- one zero
- how many positives?

Two's Complement for N=32

0000 ... 0000 0000 0000 0000	_{two} =	0 _{ten}
0000 ... 0000 0000 0000 0001	_{two} =	1 _{ten}
0000 ... 0000 0000 0000 0010	_{two} =	2 _{ten}
...		
0111 ... 1111 1111 1111 1101	_{two} =	2,147,483,645 _{ten}
0111 ... 1111 1111 1111 1110	_{two} =	2,147,483,646 _{ten}
0111 ... 1111 1111 1111 1111	_{two} =	2,147,483,647 _{ten}
1000 ... 0000 0000 0000 0000	_{two} =	-2,147,483,648 _{ten}
1000 ... 0000 0000 0000 0001	_{two} =	-2,147,483,647 _{ten}
1000 ... 0000 0000 0000 0010	_{two} =	-2,147,483,646 _{ten}
...		
1111 ... 1111 1111 1111 1101	_{two} =	-3 _{ten}
1111 ... 1111 1111 1111 1110	_{two} =	-2 _{ten}
1111 ... 1111 1111 1111 1111	_{two} =	-1 _{ten}

- ° One zero; 1st bit called **sign bit**
- ° 1 “extra” negative: no positive 2,147,483,648_{ten}

