

# Memory, Data, & Addressing I

CSE 351 Summer 2021

**Instructor:**

Mara Kirdani-Ryan

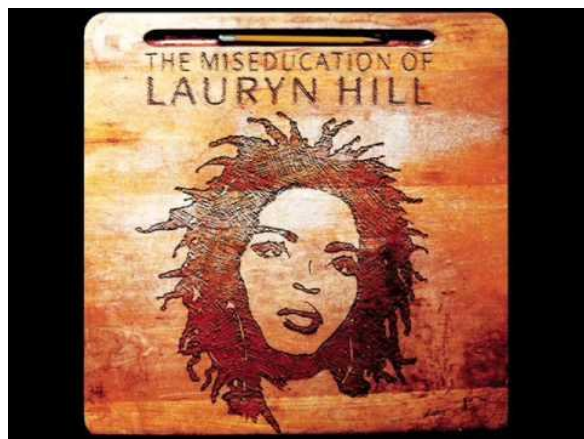
**Teaching Assistants:**

Kashish Aggarwal

Nick Durand

Colton Jobs

Tim Mandzyuk



# Questions in Lecture

- Ohyay is new for us too!
  - Help us make this course phenomenal
- Questions in “Questions” pane, staff will respond in chat, then remove the question
  - “Questions” got a bit cluttered on Monday
  - Keep an eye on chat! See if someone’s asked your question!

# Gentle reminders!

- Pre-Course Survey and hw0 due tonight @ 8pm
- hw1 due Friday (6/25) @ 8pm
- Lab 0 due Monday (6/28) @ 8pm
  - This lab is *exploratory* and looks like a hw; the other labs will look a lot different (involve writing code etc.)
  - Don't worry if everything in Lab 0 doesn't make perfect sense right now! We'll continue to cover everything in more detail
  - Lab 0: get used to modifying and running C code to see the output
    - A powerful tool for understanding the technical concepts in this course!
- Readings should be completed by 10am on day of lecture
- Lecture activities should be completed by 10am of NEXT lecture

# How'd the reading go?

# Unit Summaries

- No exams this quarter! Yay!
- We're doing something else instead...

# Unit Summaries\*

- No exams this quarter! Yay!
- We're doing something else instead...
  
- 3 Tasks per Summary (we'll have 3 total):
  1. Create a floorplan (4-5 hours)
  2. Reflect on your learning (30min)
  3. Answer a few "exam-style" questions (30min)
  
- \*this is reiterating [http://cs.uw.edu/351/unit\\_summaries](http://cs.uw.edu/351/unit_summaries), look there for complete details

# Task 1: Create a Floorplan

- We're treating units as "floors" in the HoC
- What would the floorplan for this "floor" look like?
  - Schematics, drawings, sketches, whatever format works for you!
- Good CS Folks™ work to understand socio-technical context!
  - Demonstrate technical & socio-technical understanding!
- You'll also submit:
  - A design doc that justifies your decisions
  - How long this all took you
  - *See the website for details*

# Task 1: Design

- This is a design task!
  - There isn't a clear, correct answer!
- **Design Requires Iteration and Feedback!**
  - Well-scoring Task 1s probably tried a few things out
- Key Idea: Prototyping
  - *Roughly sketch a few ideas*, then pick one to work on
- Get feedback on prototypes and early iterations!
  - We'll have some in-class time, ask for feedback in OH
  - Small-group work ok, but *your final design should represent your synthesis, words, and design*
  - We can't grade in advance, but we can critique



# Task 1: Evaluation Criteria

- **Completeness:** Did you cover everything?
  - Are omissions justified?
- **Cohesion:** Do things fit together sensibly?
  - We'll be gentle, we understand it isn't a perfect fit
- **Clarity:** Can we understand you?
  - Have you worked through meaningful iterations? Does this seem polished for 4-5 hours of work?
- **Creativity:** Does your floorplan feel uniquely yours?
- We're not grading on conventional artistic skill!
  - Be unconventional! We're cool with that!

# Task 1: Qualification

- This is different than anything from my undergrad
  - Probably different from most CSE courses as well
- **I want to try this at least once!**
- No examples, but we're here to help!

# Task 1: Qualification

- This is different than anything from my undergrad
  - Probably different from most CSE courses as well
- **I want to try this at least once!**
- No examples, but we're here to help!
- If you're not sure where to start, that's ok!
  - Design tasks usually feel this way
- **Most of the work you do after college will be a design task!**
  - No examples, no clear answers, requiring creativity and justification
  - Hopefully, this is good preparation 😊

# Unit Summary: Call to Action

- Start thinking about how these pieces fit together!
    - Start now! It'll save you time! 😁
  - Reach out if you have concerns/questions!
    - This is an experiment\*, I'd like it to be a good one
  - Demonstrate your socio-technical understanding!
- 
- \*an experiment that's based in best-practices in education

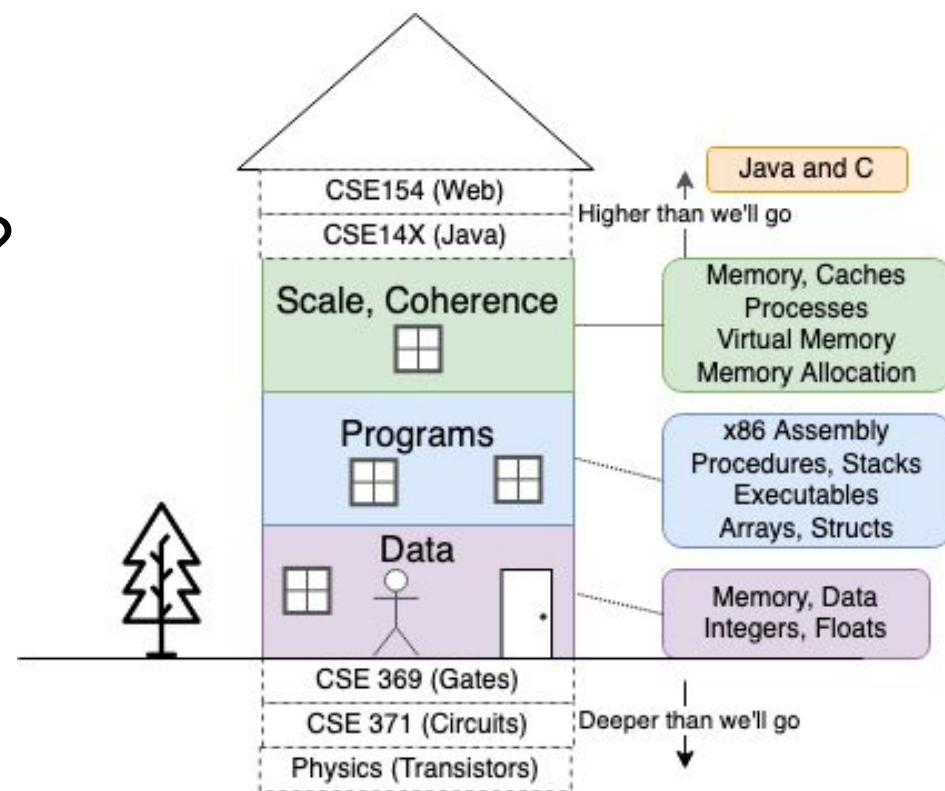
"It is creative apperception more than anything else that makes the individual feel that life is worth living. Contrasted with this is a relationship to external reality which is one of compliance, the world and its details being recognized but only as something to be fitted in with or demanding adaption.

"It is creative apperception more than anything else that makes the individual feel that life is worth living. Contrasted with this is a relationship to external reality which is one of compliance, the world and its details being recognized but only as something to be fitted in with or demanding adaption. Compliance carries with it a sense of futility for the individual and is associated with the idea that nothing matters and life is not worth living. In a tantalizing way, many individuals have experienced just enough of creative living to recognize that for most of their time they are living uncreatively, as if caught up in the creativity of someone else, or of a machine" (p87).

*D.W. Winnicott*

# First Floor: Data

- How do we represent data (strings, numbers) computationally?
- What limits exist? Why?
- What values were encoded into data representations?
- What was prioritized? Why?
- Today: Memory!

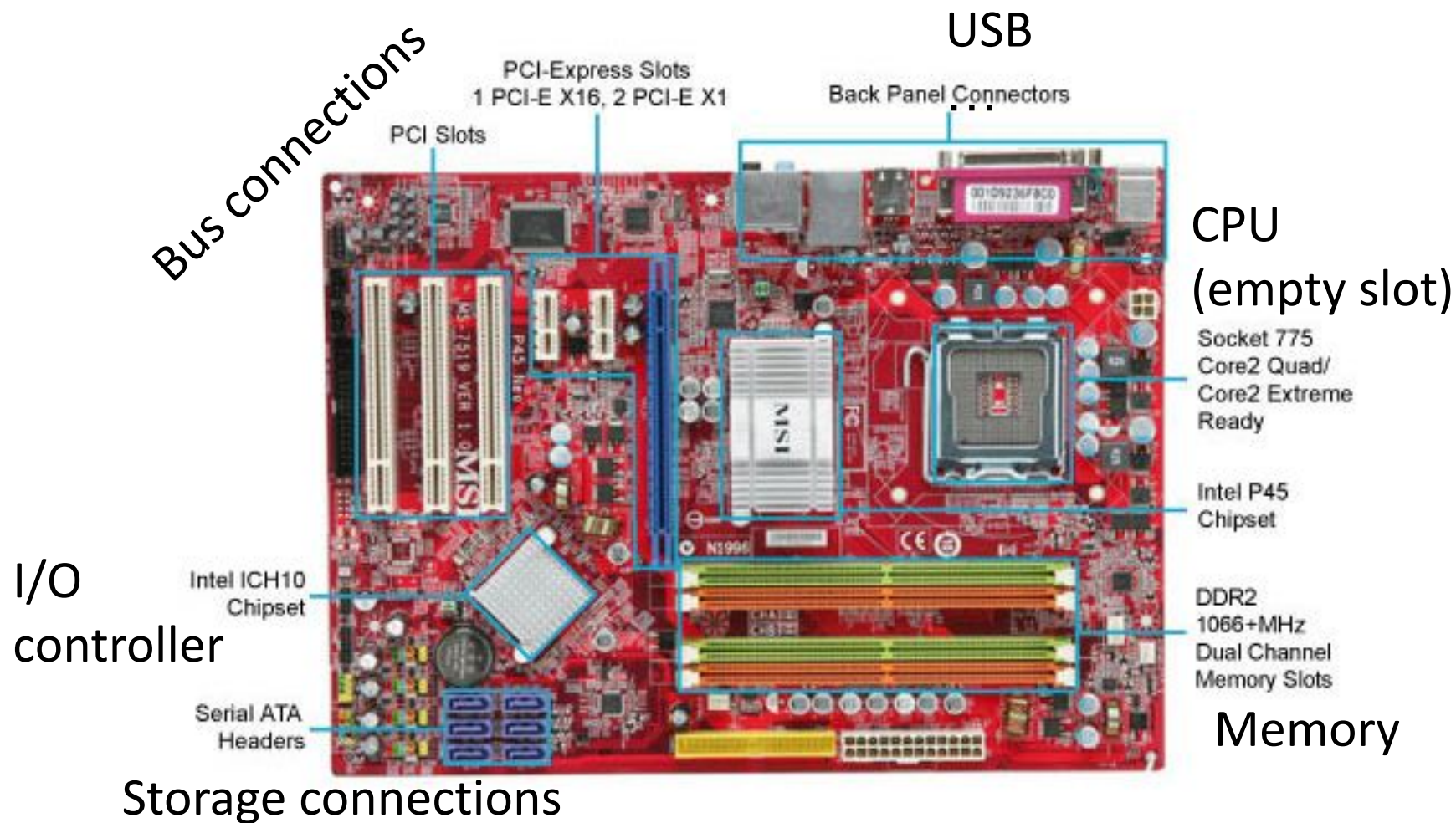


# Memory, Data, and Addressing

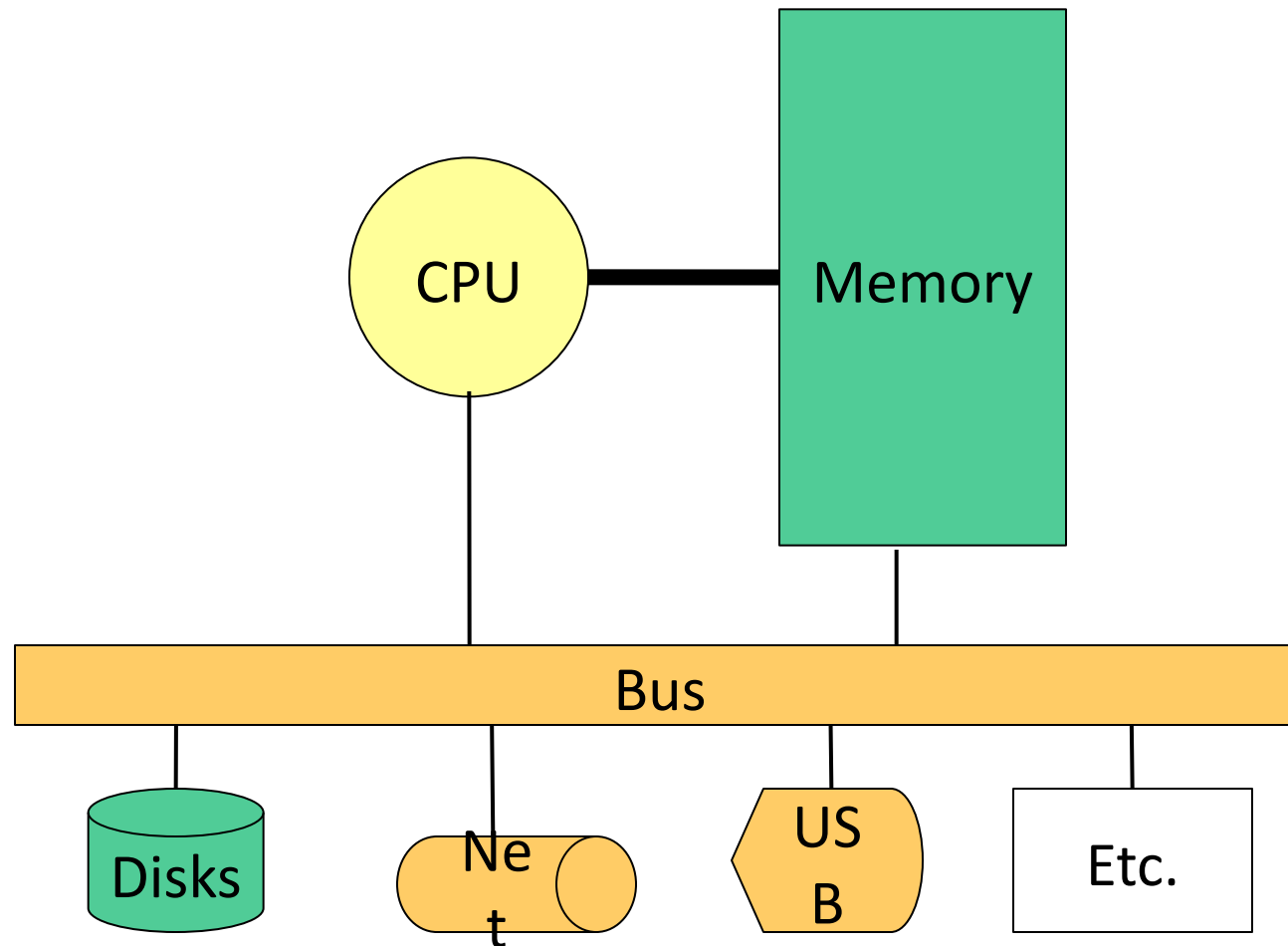
- Hardware - High Level Overview
- Representing information as bits and bytes
  - Memory is a byte-addressable array
  - “Word” size = address size = register size
- Organizing and addressing data in memory
  - Endianness – ordering bytes in memory
- Manipulating data in memory using C
- Boolean algebra and bit-level manipulations
- *How do we store information for other floors and systems to access?*



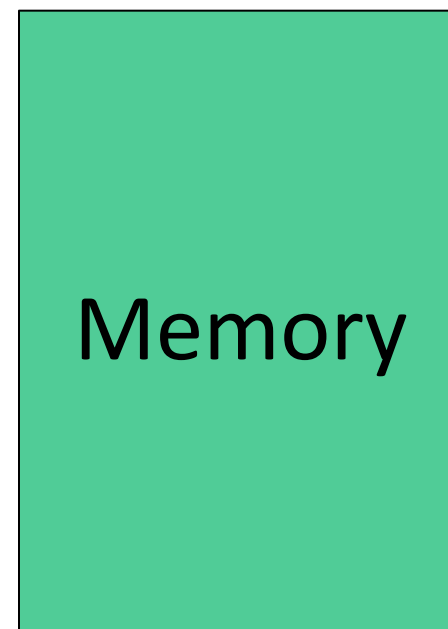
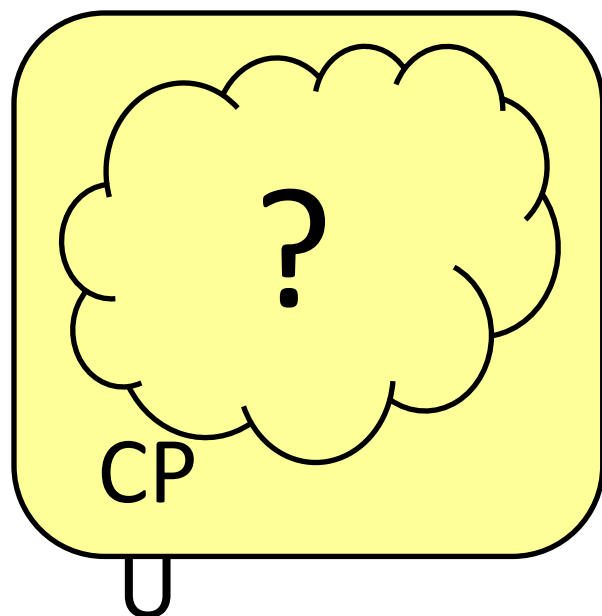
# (Modern) Hardware: Physical View



# (Modern) Hardware: Logical View



# Hardware: 351 View (version 0)

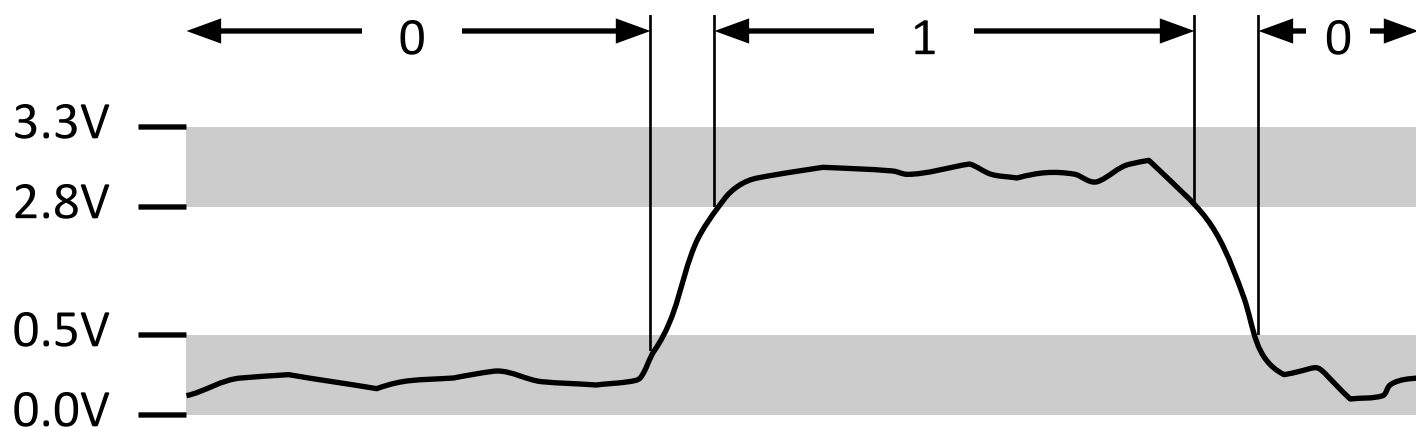


- The CPU **executes** instructions
- Memory **stores** data
- Binary encoding!
  - Instructions *are* just data

How are data  
and instructions  
represented?

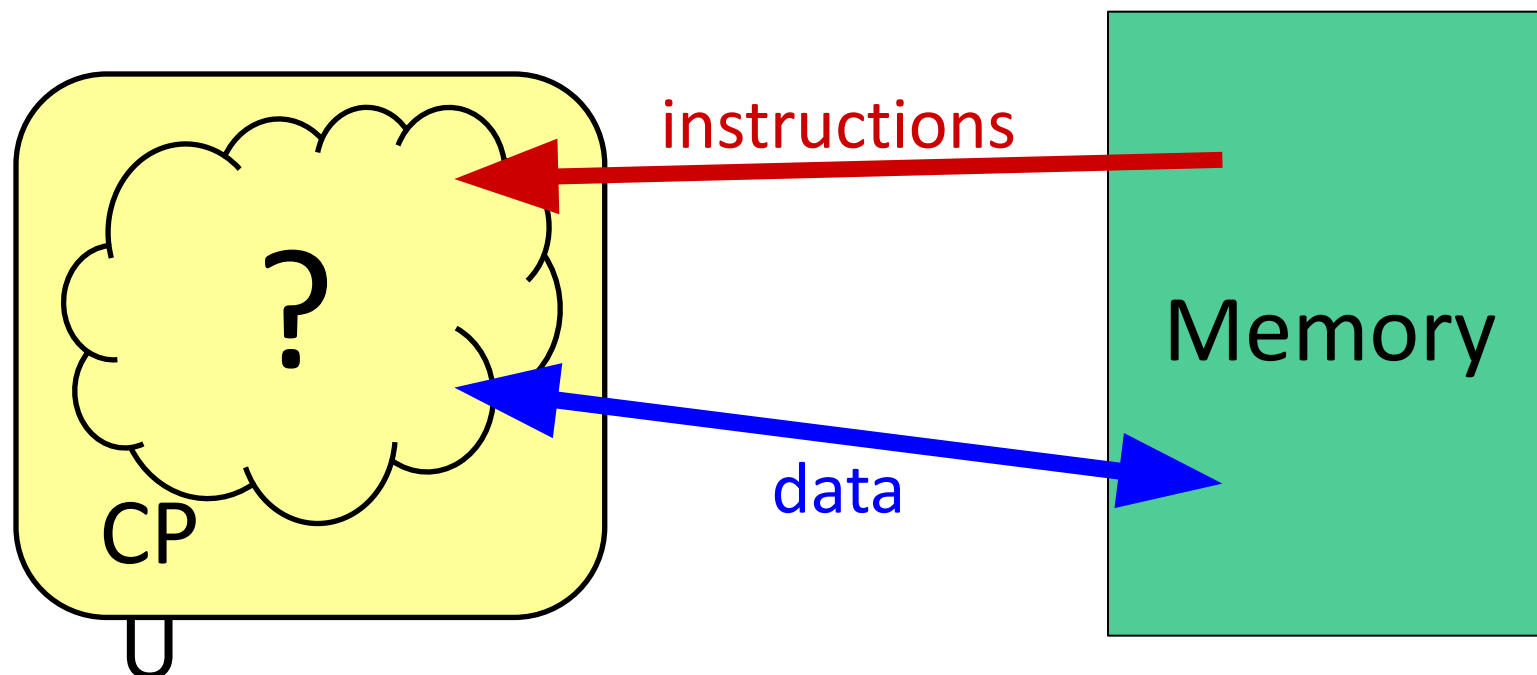
# Aside: Why Base 2?

- Electronic implementation
  - Easy to store with bi-stable elements
  - Reliably transmitted on noisy and inaccurate wires



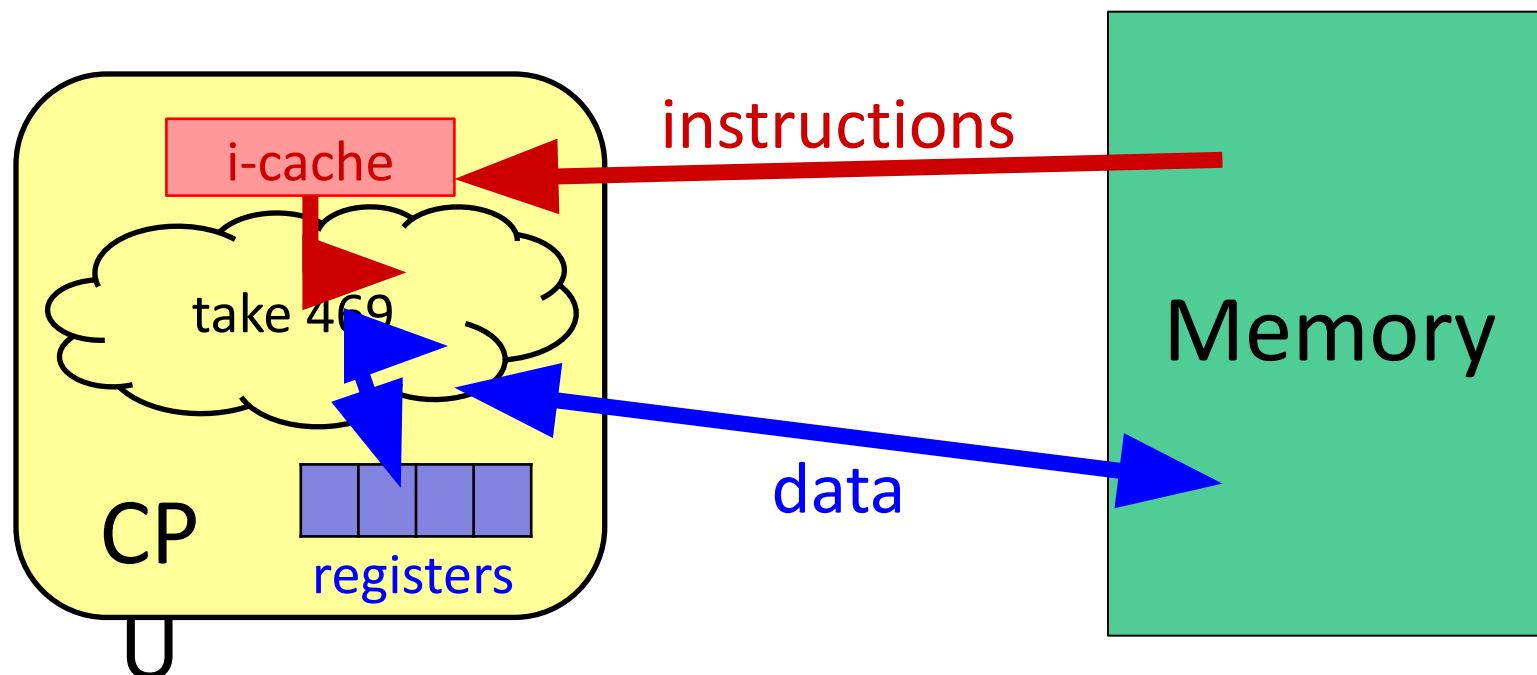
- Other bases possible, but not yet viable:
  - DNA data storage (base 4: A, C, G, T) is hot@UW
  - Quantum computing

# Hardware: 351 View (version 0)



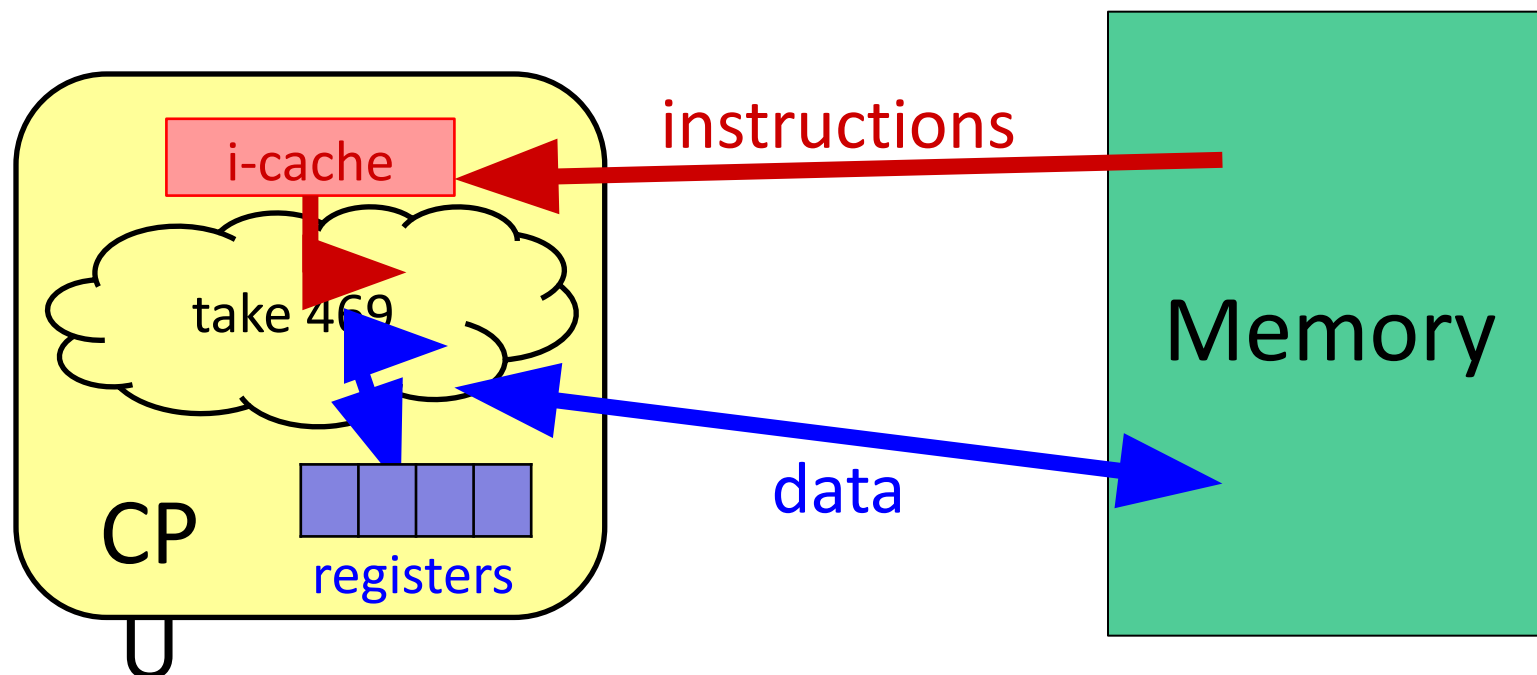
- To execute an instruction, the CPU must:
  - 1) Fetch the instruction
  - 2) (if applicable) Fetch data needed by the instruction
  - 3) Perform the specified computation
  - 4) (if applicable) Write the result back to memory

# Hardware: 351 View (version 1)



- More CPU details:
  - Instructions are held temporarily in the **instruction cache**
  - Other data are held temporarily in **registers**
- **Instruction fetching** is hardware-controlled
- **Data movement** is programmer-controlled (assembly)

# Hardware: 351 View (version 1)



- ❖ Let's start with memory!

How does a program find its data in memory?

# Review Questions

- By looking at the bits stored in memory, I can tell what a particular 4 bytes is being used to represent.

 True  False  Help!



# Review Questions

- By looking at the bits stored in memory, I can tell what a particular 4 bytes is being used to represent.

 True  False  Help!

- We can fetch a piece of data from memory if we have its address.

 True  False  Help!

# Review Questions

- By looking at the bits stored in memory, I can tell what a particular 4 bytes is being used to represent.

 True  False  Help!

- We can fetch a piece of data from memory if we have its address.

 True  False  Help!

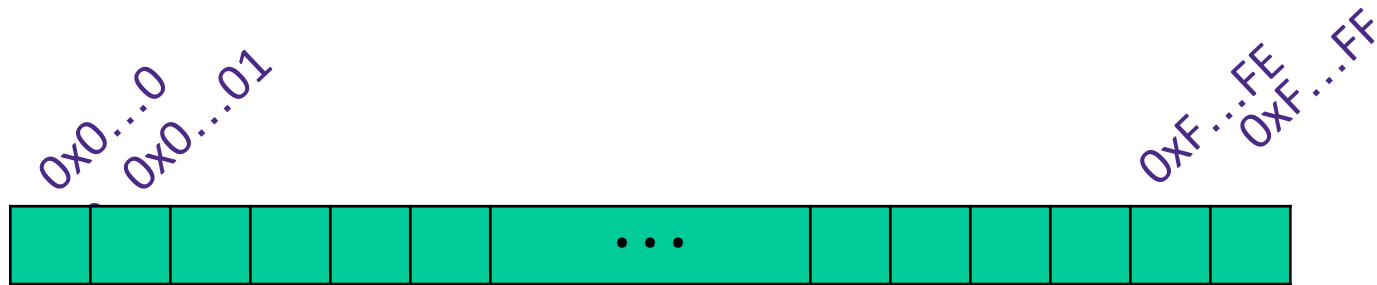
- Which of the following *bytes* have a most-significant bit (MSB) of 1?

 0x63  0x90  0xCA  0xF   
Help!

# Binary Encoding Additional Details

- Because storage is finite in reality, everything is stored as “fixed” length
  - Data is moved and manipulated in fixed-length chunks
  - Multiple fixed lengths (e.g. 1 byte, 4 bytes, 8 bytes)
  - Leading zeros now *must* be included up to “fill out” the fixed length
- Example: the “eight-bit” representation of the number 4 is 0b00000100
  - Most Significant Bit (MSB)
  - Least Significant Bit (LSB)

# Addresses refer to Bytes of Memory!



- Conceptually, memory is a single, large array of bytes, each with a unique *address* (index)
  - Each address is just a number represented in *fixed-length* binary
- Programs refer to bytes in memory by their *addresses*
  - Domain of possible addresses = *address space*
  - We can store addresses as data to “remember” where other data is in memory
- But not all values fit in a single byte... (e.g. 351)
  - Many operations use multi-byte values

# Machine “Words”

- Instructions encoded into machine code (binary)
  - Historically (still true in some assembly languages), all instructions were exactly the size of a **word**
- We chose to match word size to address size
  - word size = address size = register size
  - word size =  $w$  bits  $\rightarrow 2^w$  addresses
- Current x86 systems use **64-bit (8-byte) words**
  - Potential address space:  $2^{64}$  addresses  
 $2^{64}$  bytes  $\approx$   **$1.8 \times 10^{19}$  bytes**  
= 18 billion billion bytes = 18 EB (exabytes)
  - Actual physical address space: **48 bits**

# Data Representations

- Sizes of data types (in bytes)

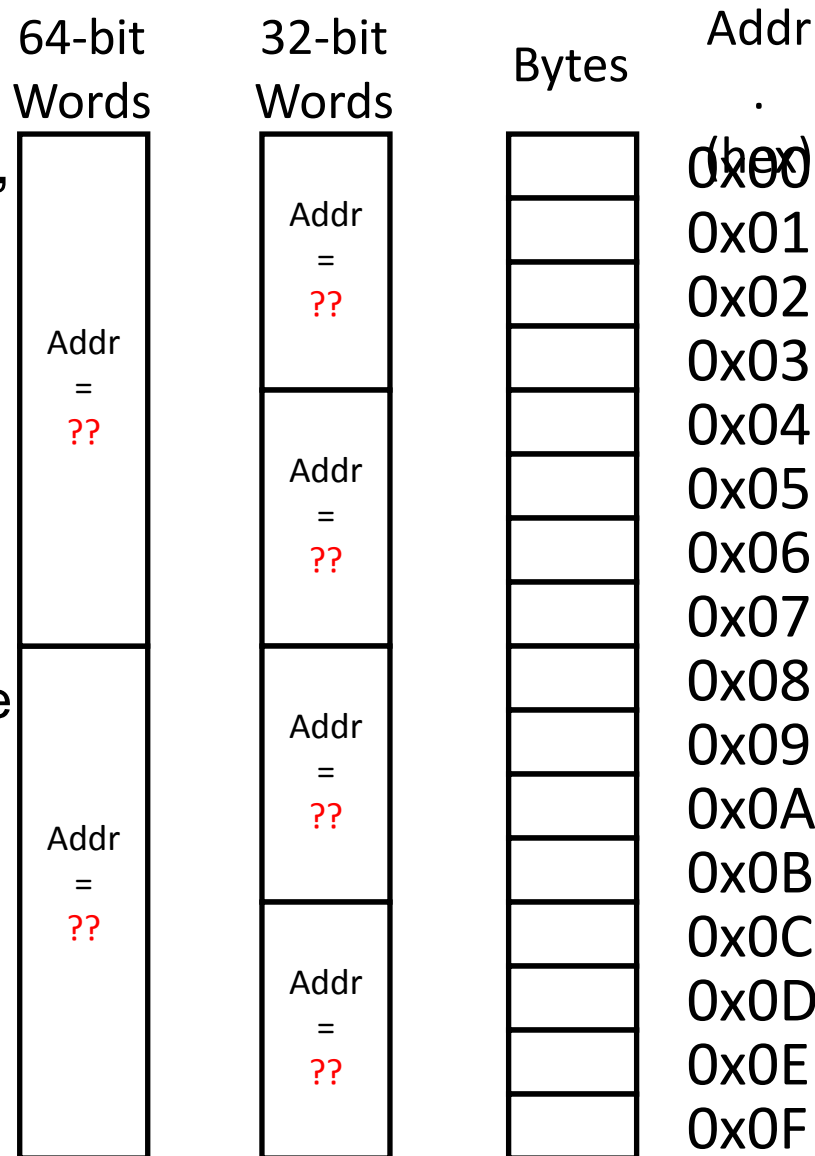
Java Data Type	C Data Type	32-bit (old)	x86-64
boolean	bool	1	1
byte	char	1	1
char	<b>none, why?</b>	2	2
short	short int	2	2
int	int	4	4
float	float	4	4
	long int	4	8
double	double	8	8
long	long long	8	8
	long double	8	16
<b>(reference)</b>	<b>pointer *</b>	<b>4</b>	<b>8</b>

address size = word size

To use "bool" in C, you must `#include <stdbool.h>`

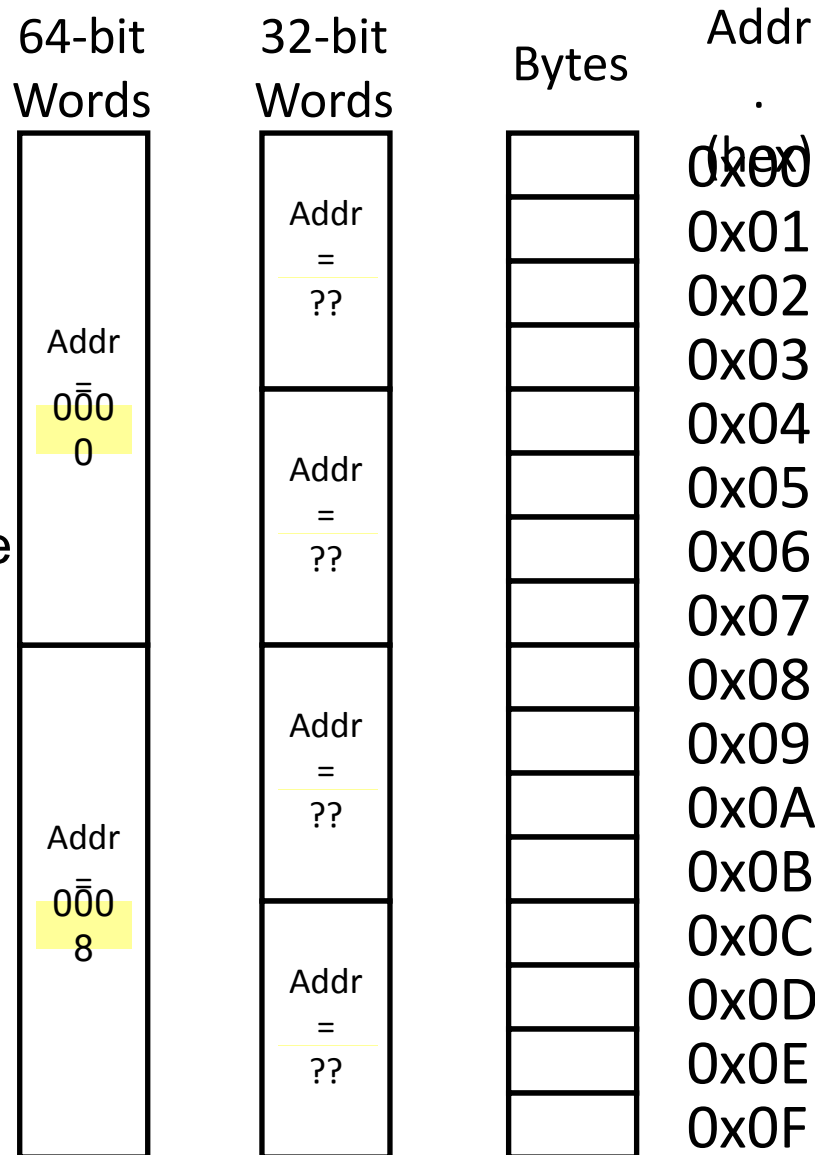
# Word-Oriented View of Memory

- Addresses still specify locations of bytes in memory, but we can choose to *view* memory as a series of word-sized chunks of data instead
  - Addresses of successive words differ by word size
  - Which byte's address should we use for each word?



# Addresses of multi-byte data

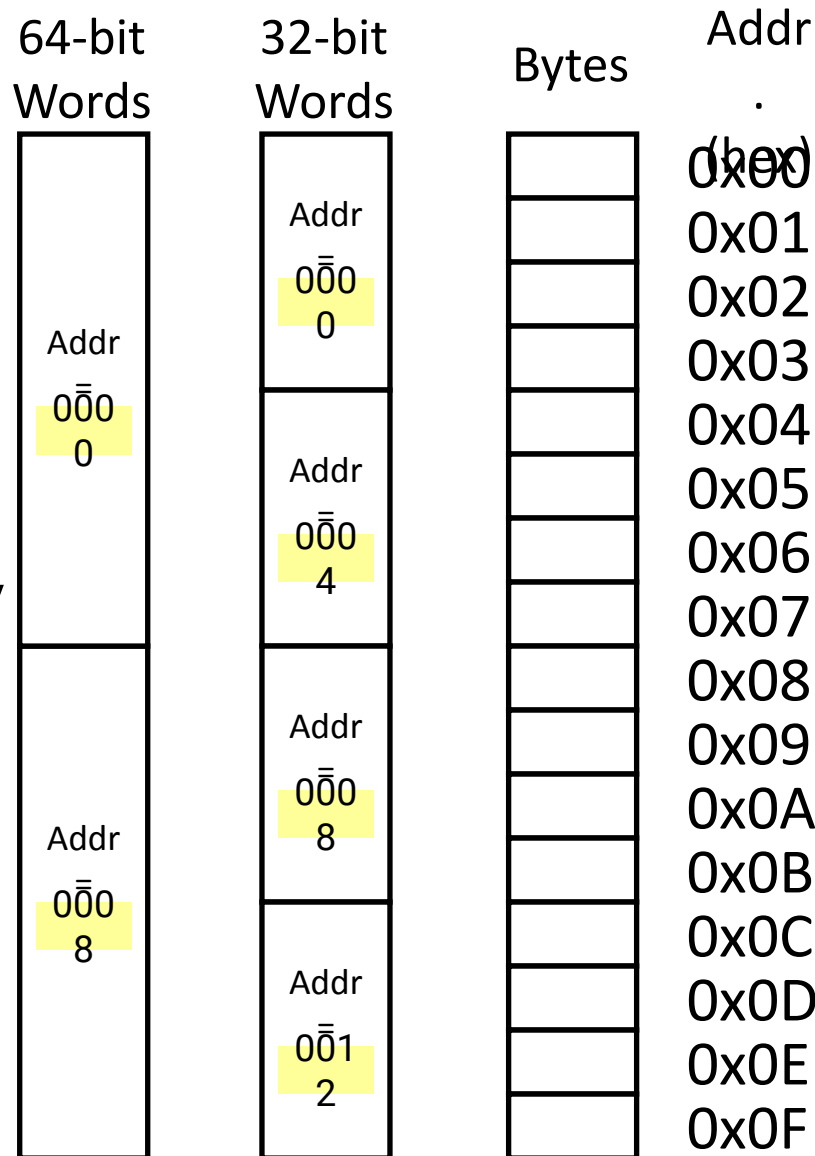
- Addresses -> bytes in memory, but we can choose to *view* memory as a series of word-sized chunks of data
  - Addresses of successive words differ by word size
  - Which byte's address should we use for each word?
- The address of *any* chunk of memory is given by the address of the first byte
  - To specify a chunk of memory, need *both* its **address** and **size**





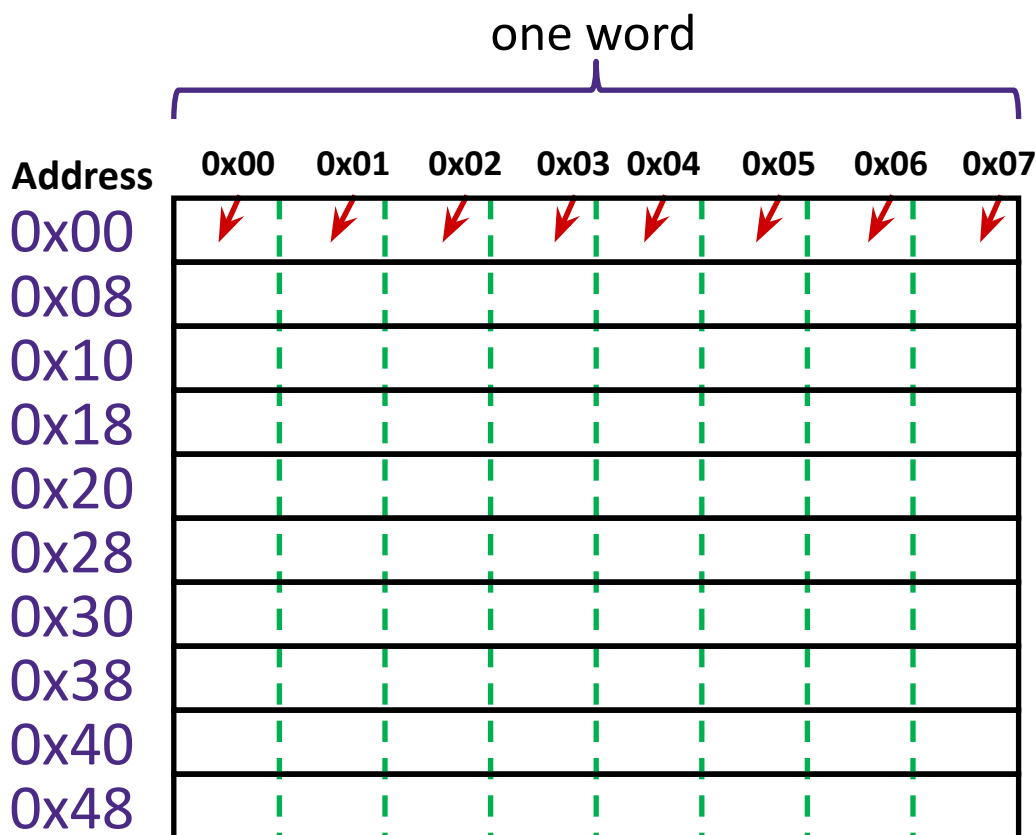
# Alignment

- The address of a chunk of memory is considered **aligned** if its address is a multiple of its size
  - View memory as a series of consecutive chunks of this particular size and see if your chunk doesn't cross a boundary



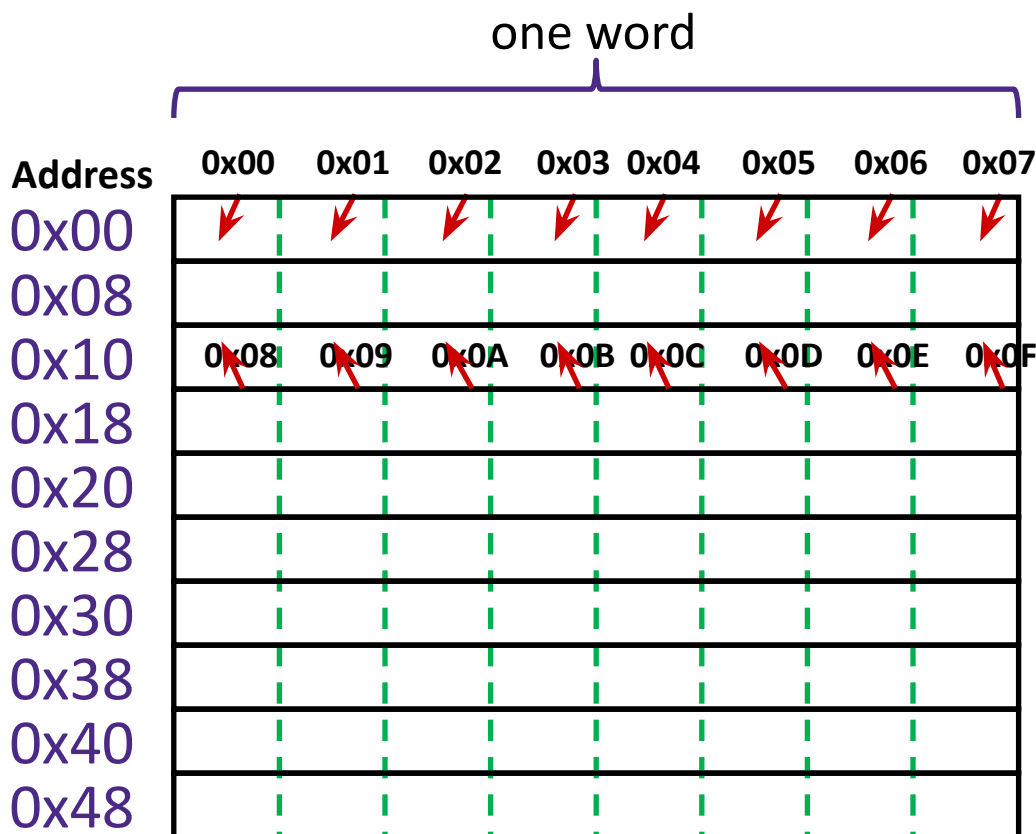
# A Picture of Memory (64-bit view)

- A “64-bit (8-byte) word-aligned” view of memory:
  - In this type of picture, each row is composed of 8 bytes
  - Each cell is a byte
  - An aligned, 64-bit chunk of data will fit on one row



# A Picture of Memory (64-bit view)

- A “64-bit (8-byte) word-aligned” view of memory:
  - In this type of picture, each row is composed of 8 bytes
  - Each cell is a byte
  - An aligned, 64-bit chunk of data will fit on one row



# Addresses and Pointers

64-bit example  
(pointers are 64-bits wide)

big-endian

- An *address* refers to a location in memory
- A *pointer* is a data object that holds an address
  - Address can point to *any* data

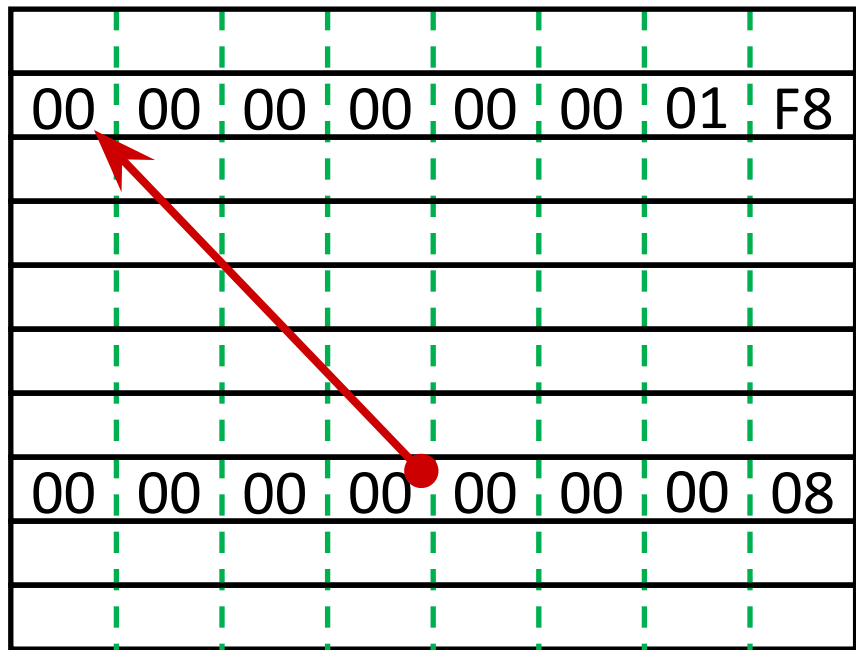
- Value 504 stored at address **0x08**

- $504_{10} = 0x1F8$   
= 0x 00 ... 00 01 F8

- Pointer stored at **0x38** points to address **0x08**

Address

0x00								
0x08	00	00	00	00	00	00	01	F8
0x10								
0x18								
0x20								
0x28								
0x30								
0x38	00	00	00	00	00	00	00	08
0x40								
0x48								



# Addresses and Pointers

64-bit example  
(pointers are 64-bits wide)

big-endian

- An *address* refers to a location in memory
- A *pointer* is a data object that holds an address
  - Address can point to *any* data
- Pointer stored at **0x48** points to address **0x38** 😞
  - Pointer to a pointer!
- Is the data stored at **0x08** a pointer?
  - Could be, depending on how you use it

Address

0x00

0x08

0x10

0x18

0x20

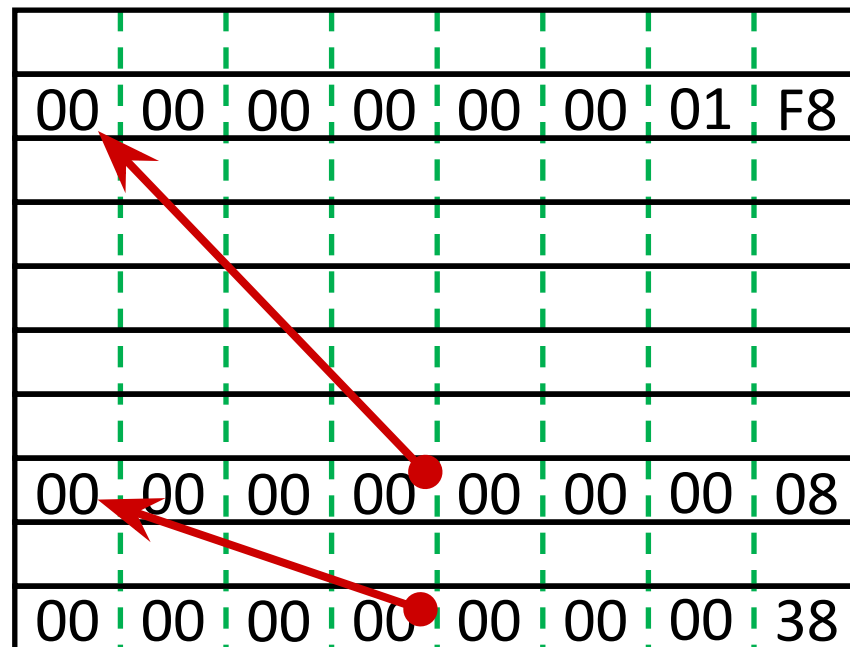
0x28

0x30

0x38

0x40

0x48



# Data Representations

- Sizes of data types (in bytes)

Java Data Type	C Data Type	32-bit (old)	x86-64
boolean	bool	1	1
byte	char	1	1
char	none, why?	2	2
short	short int	2	2
int	int	4	4
float	float	4	4
	long int	4	8
double	double	8	8
long	long long	8	8
	long double	8	16
<b>(reference)</b>	<b>pointer *</b>	<b>4</b>	<b>8</b>

address size = word size

To use "bool" in C, you must `#include <stdbool.h>`

# Memory Alignment Revisited

- A primitive object of  $K$  bytes must have an address that is a multiple of  $K$  to be considered *aligned*

	Type
1	char
2	short
4	int, float
8	long, double, pointers

- For good memory system performance, Intel (x86) recommends data be aligned
  - x86-64 hardware will work correctly otherwise
    - *Design choice*: x86-64 instructions are *variable* length

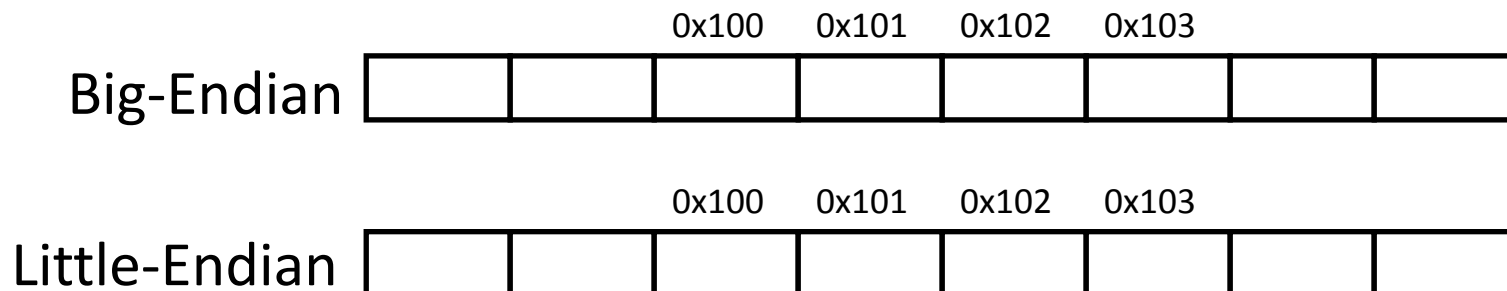
# Byte Ordering

- How should bytes within a word be ordered?
  - Goal: consecutive bytes in consecutive addresses
  - **Ex:** store the 4-byte (32-bit) `int`: `0x a1 b2 c3 d4`
- Byte ordering convention: *Endianness*
  - The two options are **big-endian** and **little-endian**
    - In which address does the least significant *byte* go?
    - Based on *Gulliver's Travels*: tribes cut eggs on different sides (big, little)



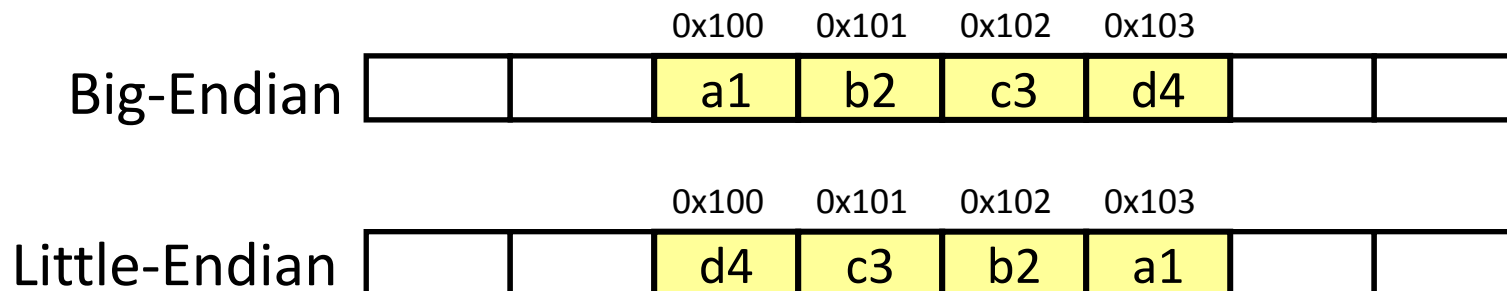
# Byte Ordering

- Big-endian (SPARC, z/Architecture)
  - Least significant byte has highest address
- Little-endian (x86, x86-64)
  - Least significant byte has lowest address
- Bi-endian (ARM, PowerPC)
  - Endianness can be specified as big or little
- **Ex:** 4-byte data 0xa1b2c3d4 at address 0x100



# Byte Ordering

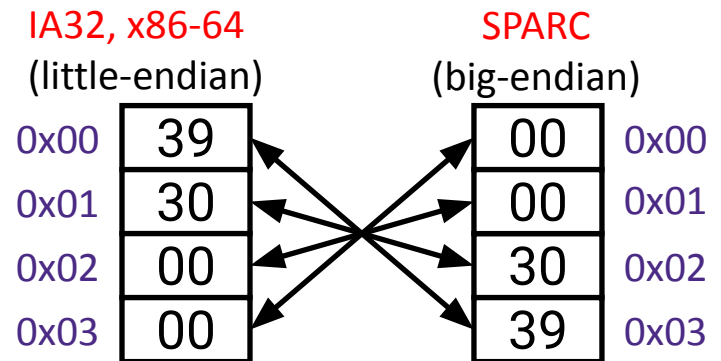
- Big-endian (SPARC, z/Architecture)
  - Least significant byte has highest address
- Little-endian (x86, x86-64)
  - Least significant byte has lowest address
- Bi-endian (ARM, PowerPC)
  - Endianness can be specified as big or little
- **Ex:** 4-byte data 0xa1b2c3d4 at address 0x100



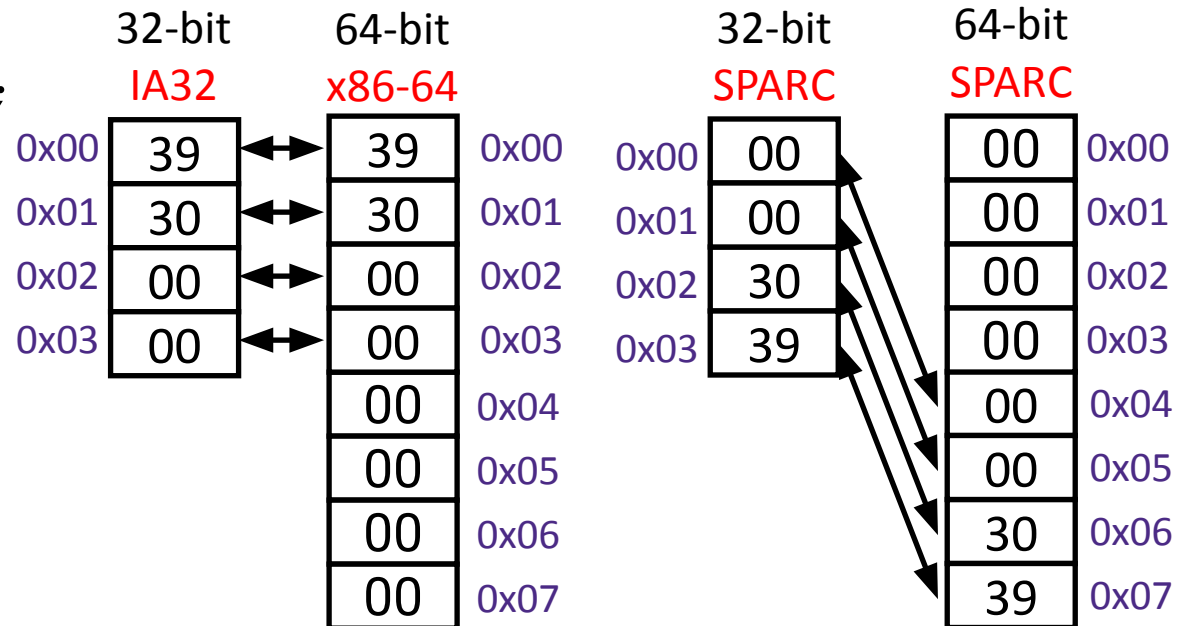
# Byte Ordering

Decimal:	12345
Binary:	0011 0000 0011 1001
Hex:	3 0 3 9

```
int x = 12345;
// or x = 0x3039;
```



```
long int y = 12345;
// or y = 0x3039;
```



(A long int is the size of a word)

# Polling Question

- We store the value  $0x\ 01\ 02\ 03\ 04$  as a **word** at address  $0x100$  in a big-endian, 64-bit machine
- What is the **byte of data** stored at address  $0x104$ ?

 **0x04**

 **0x40**

 **0x01**

 **0x10**

 **We're lost...**

# Endianness

- *Endianness only applies to memory storage*
- Often programmer can ignore endianness because it is handled for you
  - Bytes wired into correct place when reading or storing from memory (hardware)
  - Compiler and assembler generate correct behavior (software)
- Endianness still shows up:
  - Logical issues: accessing different amount of data than how you stored it (e.g. store `int`, access LSB as `char`)
  - Need to know exact values to debug memory errors
  - Manual translation to and from machine code (in 351)

# Challenge Question

- Assume the state of memory is as shown below for a little-endian machine.

0x100					0x107				
9F	23	B7	C8	55	D0	00	04	08	

- If we (1) *read* the value of an `int` at address `0x102`, (2) add 8 to it, and then (3) store the new value as an `int` at address `0x104`, which of the following addresses retain their original value?

**A. 0x102**    **B. 0x104**    **C. 0x105**    **D. 0x107**

# But, how'd we get here?

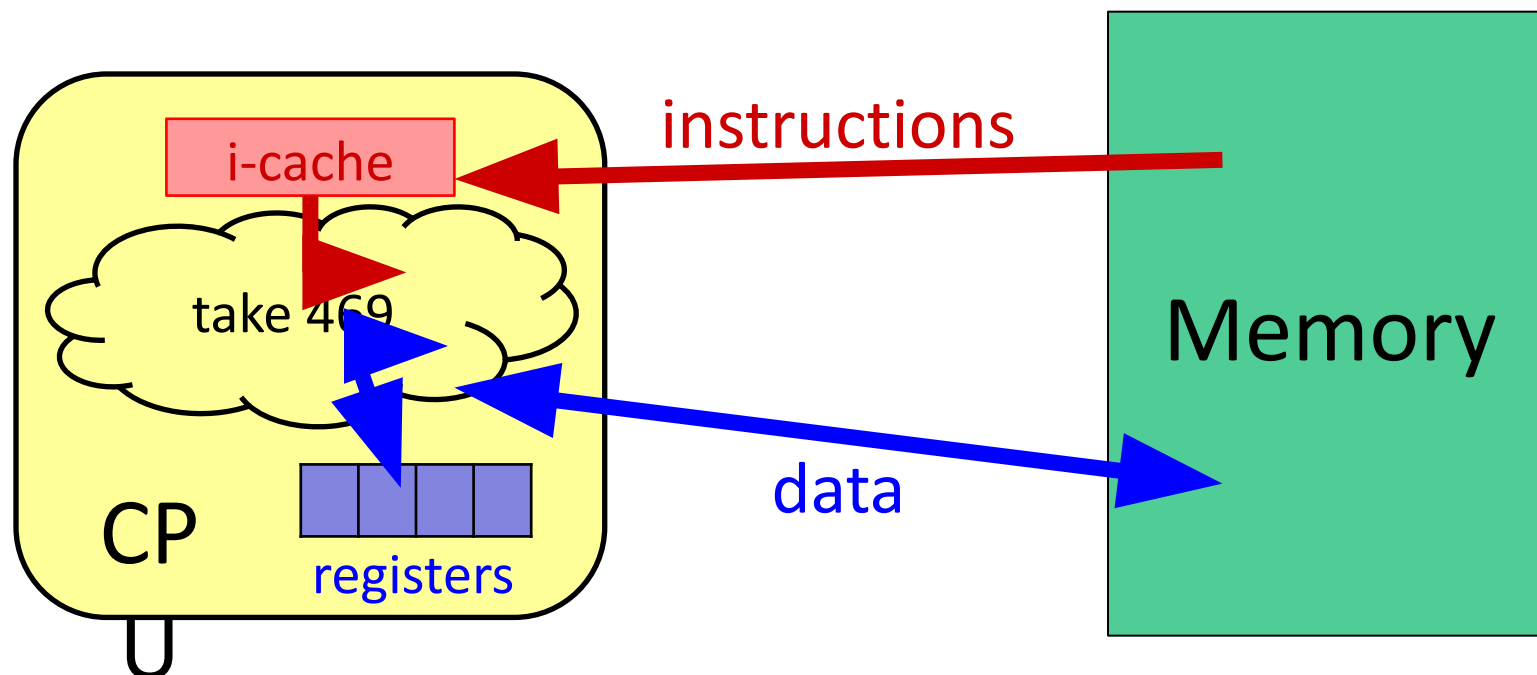
- We'll be answering this question again and again
- Modern hardware, modern house, historic relics
- No decisions were an accident!
  - Priorities might've been misaligned, dated, inconsistent
  - Priorities might've been racist, sexist, ableist
- Let's go back to modern\* foundations...

\*We'll go further back in a week

What were the prevailing notions of computing during its modern incarnation?



# Hardware: 351 View (version 1)



- More CPU details:
  - Instructions are held temporarily in the **instruction cache**
  - Other data are held temporarily in **registers**
- **Instruction fetching** is hardware-controlled
- **Data movement** is programmer-controlled (assembly)

# (Modern) Hardware: Historic View

- **Computer:** one who computes



The women of Bletchley Park, Credit: BBC

# (Modern) Hardware: Historic View

- **Computer:** one who computes



The women of Bletchley Park, Credit: BBC

- Mostly white cis-women
- “Boring, repetitive work”, doing math quickly

# Computing in the US

- **Computer:** one who computes
- Observatory calculations @ Harvard (1870s)



Human Computers at NACA, Credit: NASA



Human Computers at JPL, Credit: JPL

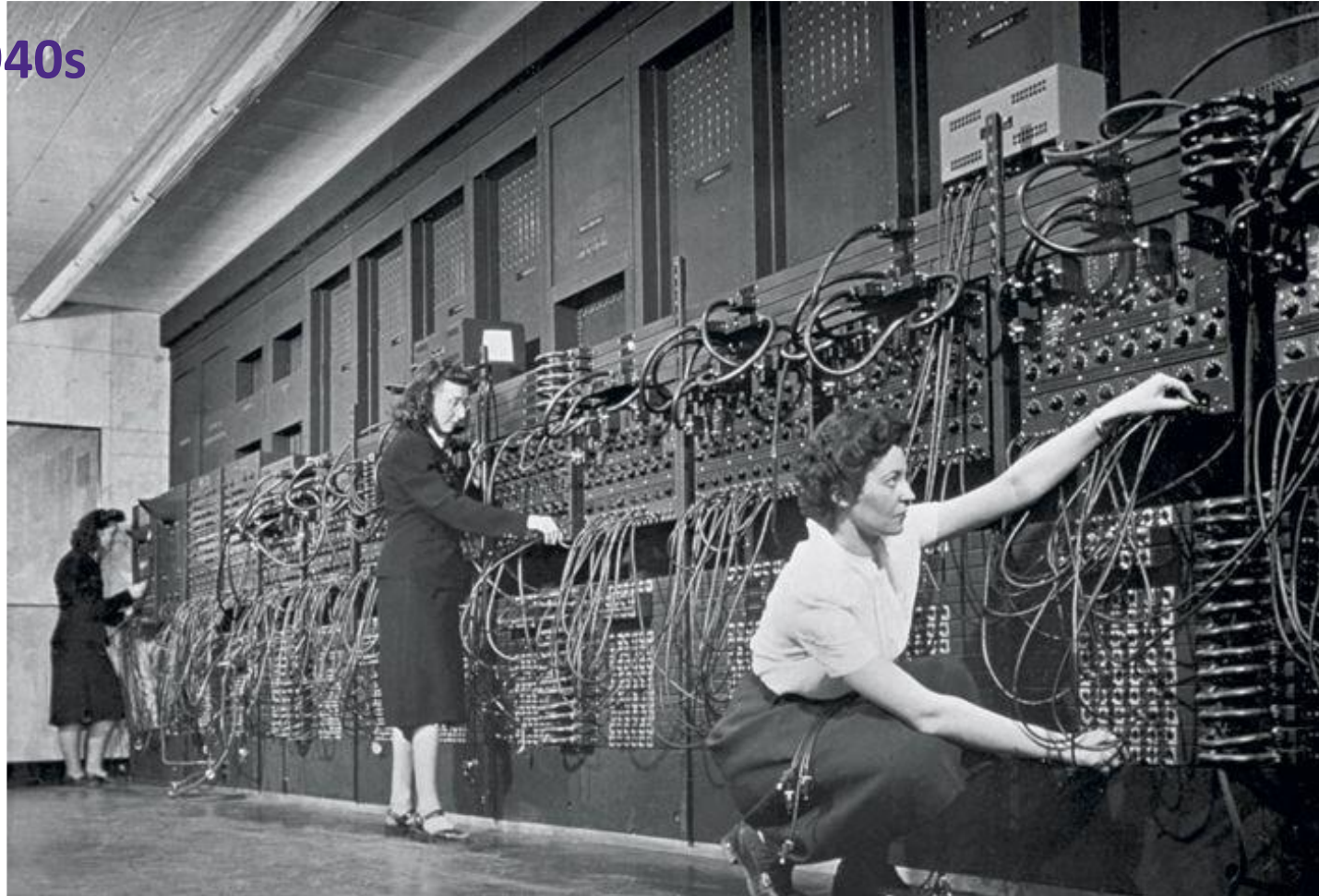


# The ENIAC: Augmenting/Automating



# Programming, historically

1940s



Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946.

Photo: Corbis

<http://fortune.com/2014/09/18/walter-isacson-the-women-of-eniac/>



# The Computer Girls

BY LOIS MANDEL

A trainee gets \$8,000 a year  
... a girl "senior systems analyst"  
gets \$20,000—and up!  
Maybe it's time to investigate....

Ann Richardson, IBM systems engineer, designs a bridge via computer. Above (left) she checks her facts with fellow systems engineer, Marvin V. Fuchs. Right, she feeds facts into the computer. Below, Ann demonstrates on a viewing screen how her facts designed the bridge, and makes changes with a "light pen."

Twenty years ago, a girl could be a secretary, a school teacher . . . maybe a librarian, a social worker or a nurse. If she was really ambitious, she could go into the professions and compete with men . . . usually working harder and longer to earn less pay for the same job.

Now have come the big, dazzling computers—and a whole new kind of work for women: programming. Telling the miracle machines what to do and how to do it. Anything from predicting the weather to sending out billing notices from the local department store.

And if it doesn't sound like woman's work—well, it just is.

("I had this idea I'd be standing at a big machine and pressing buttons all day long," says a girl who programs for a Los Angeles bank. I couldn't have been further off the track. I figure out how the

computer can solve a problem, and then instruct the machine to do it."

"It's just like planning a dinner," explains Dr. Grace Hopper, now a staff scientist in systems programming for Univac. (She helped develop the first electronic digital computer, the Eniac, in 1946.) "You have to plan ahead and schedule everything so it's ready when you need it. Programming requires patience and the ability to handle detail. Women are 'naturals' at computer programming."

What she's talking about is *aptitude*—the one most important quality a girl needs to become a programmer. She also needs a keen, logical mind. And if that zeroes out the old Billie Burke-Gracie Allen image of femininity, it's about time, because this is the age of the Computer Girls. There are twenty thousand of them in the United (cont. on page 54)



**“People like Grace Hopper were very consciously mobilizing gender stereotypes to get women in.”**

**Janet Abbate, *Recoding Gender***



# Modern Hardware: Historic Relics

1. “Boring, repetitive work” should be automated for efficiency/profit
  - Two narratives, both true: Automation/Augmentation
  - Consistently eliminating jobs of marginalized folks

# Historic Robots

- *Robot*: (Czech) compulsory service
  - Slav *robot*a: servitude, hardship
- Robots: tool to replace “unskilled” work, servants

*The robots are coming!  
When they do, you'll  
command a host of  
push-button servants.*

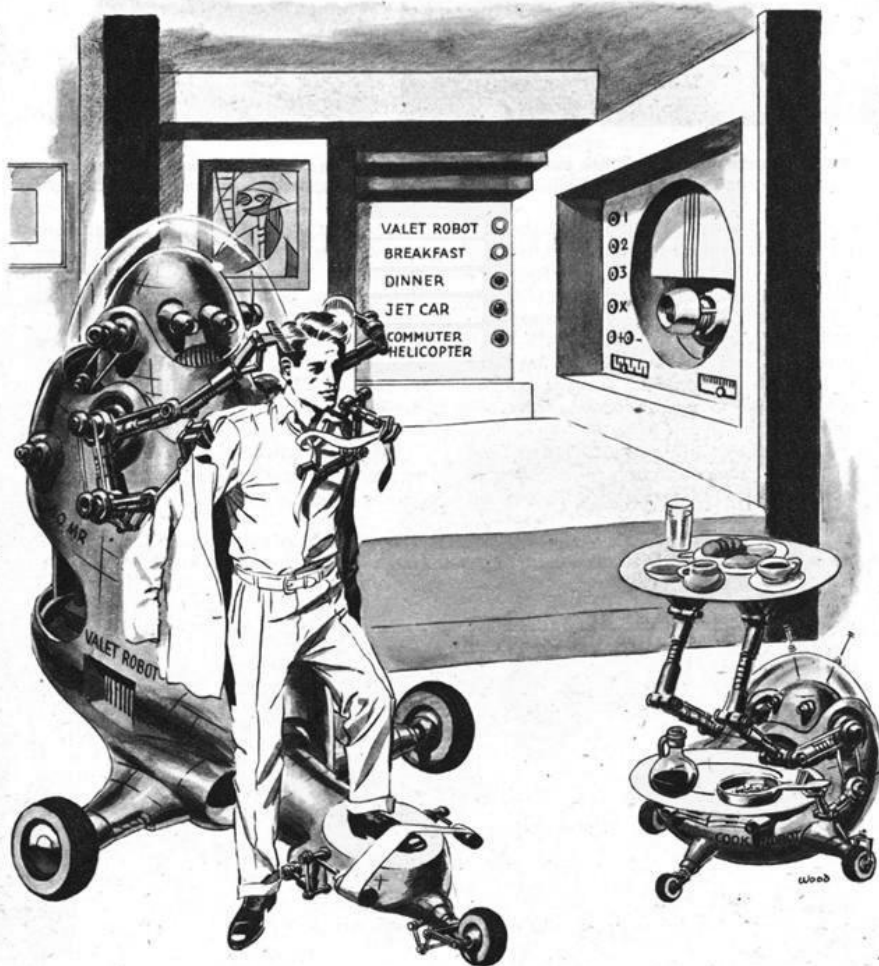
By O. O. Binder

Robots will dress you, comb your hair and serve meals in a jiffy.

## You'll Own

**I**N 1863, Abe Lincoln freed the slaves. But by 1965, slavery will be back! We'll all have personal slaves again, only this time we won't fight a Civil War over them. Slavery will be here to stay.

Don't be alarmed. We mean robot "slaves." Let's take a peek into the future



## "Slaves" by 1965

to see what the Robot Age will bring. It is a morning of 1965. . .

You are gently awakened by soft chimes from your robot clock, which also turns up the heat, switches on radio news and signals your robot valet, whom you've affectionately named "Jingles." He turns on your shower, dries you with a blast of warm air, and runs an electric shaver over your stubble. Jingles helps you dress, tying your necktie perfectly and parting your hair within a millimeter of where you like it.

Down in the kitchen, Steela, the robot cook, opens a door in her own alloy body and withdraws eggs, toast and coffee from her built-in stove. Then she dumps the dishes back in and you hear her internal dishwasher bubbling as you leave for the garage.

In your robot car you simply set a dial for your destination and relax. Your automatic auto does the rest—following a radar beam downtown, passing other cars, slowing down in speed zones, gently applying radar brakes when necessary, even gassing up when your tank is empty. You give a friendly wave to robot traffic cops who break up all traffic jams with electronic speed and perception. Suddenly you hear gun shots. A thief is emptying his gun at a robot cop, who just keeps coming, bullets bouncing from his steel chest. The panicky thug races away in his car but the robot cop shifts himself into eighth gear and overtakes the bandit's car on foot.

If you work at an office, your robot secretary takes dictation on voice tapes and types internally at the same time, handing you your letter as soon as you say "yours truly." If you go golfing, the secretary answers the phone, records any messages, and also delivers any pre-recorded message of yours.

At home, your robot reciter reads books to you from your microfilm library. His eye can see microscopic prints. Or you play chess with a robot companion, matching your wits against an electronic brain.

In 1956 research scientists already devised robot game players who always won against human opponents. Of

course the 1965 robots can be adjusted as you wish by buttons for high, average or low skill.

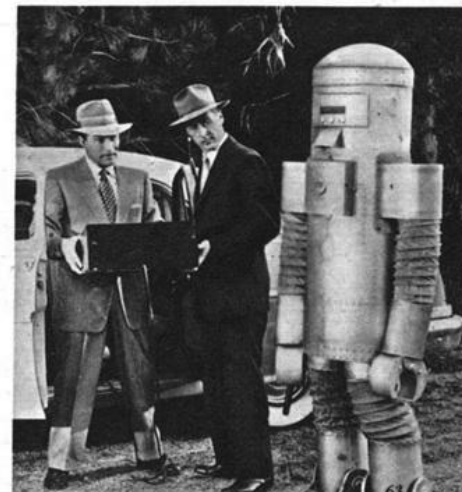
When a heavy snow falls you don't have to shovel the walk. Neither does your robot caretaker. He merely sprays cheap atomic heat around the grounds, melting the snow as fast as it falls. Yours is a robot home, too, turning all day on a foundation turntable to enjoy the utmost benefits of the sun.

At bedtime, you snap on the robot guard who detects any burglars electronically. It's a cheaper version of the robot alarm system in 1956, guarding precious documents like the original Constitution, in the National Archives Building.

During the night, no mice or rats can escape the super-sensitive ears and infra-red eyes of your roving robot cat. Back in 1956 scientists experimented with the first robot animals, such as the robot mole that could follow light beams, the robot moth dancing around flames and robot mice finding their way out of mazes.

Fanciful, this picture of the near future? A foretaste of such robot wonders

Metal star of *Zombies Of The Stratosphere* heeds his masters in science-fiction movie.



# It was racist, and continues to be



Twitter Engineering   
@TwitterEng

We're starting with a set of words we want to move away from using in favor of more inclusive language, such as:

Avoid non-inclusive language	⇒	Prefer inclusive versions
Whitelist	⇒	Allowlist
Blacklist	⇒	Denylist
Master/slave	⇒	Leader/follower, primary/replica, primary/standby
Grandfathered	⇒	Legacy status
Gendered pronouns (e.g. guys)	⇒	Folks, people, you all, y'all
Gendered pronouns (e.g. he/him/his)	⇒	They, them, their
Man hours	⇒	Person hours, engineer hours
Sanity check	⇒	Quick check, confidence check, coherence check
Dummy value	⇒	Placeholder value, sample value

12:52 PM · Jul 2, 2020 · [Twitter Web App](#)

\*This isn't to praise Twitter, this was the first list I found

# Modern Hardware: Historic Relics

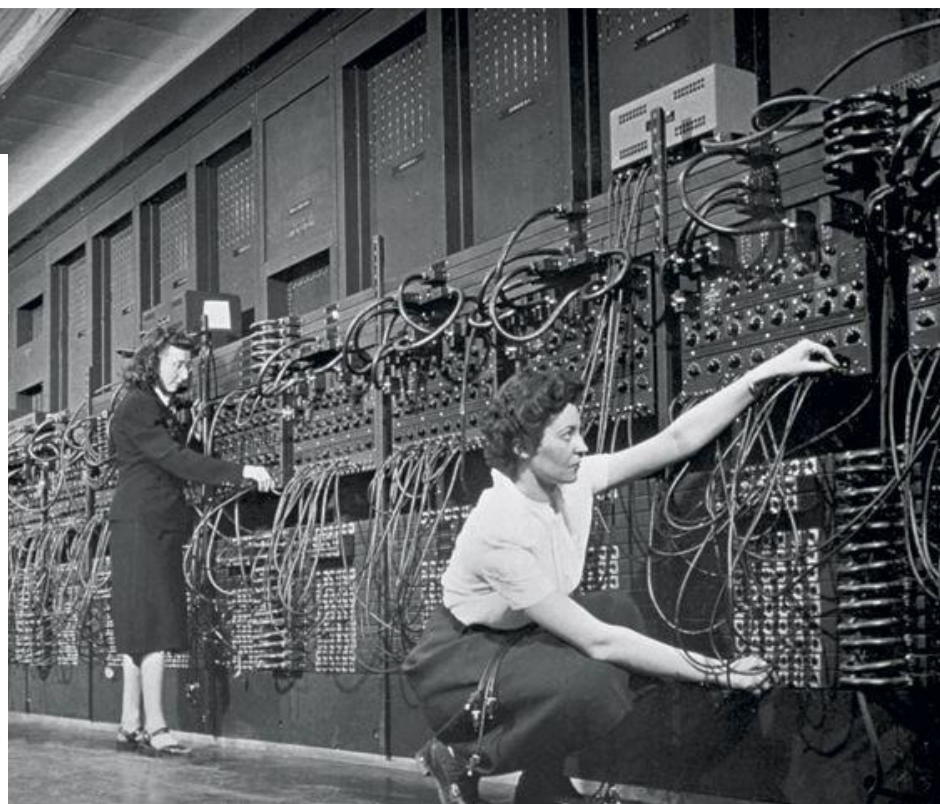
1. “Boring, repetitive work” should be automated
  - Two narratives, both true: Automation/Augmentation
  - Consistently eliminating jobs of marginalized folks
  - This relic remains!
2. Boring, repetitive work = “robot work”
  - Performed by those deemed *less than human*
  - Robot work should be done by robots (non-human)
    - “Robot work” anything unvalued by the powerful
  - If the task can’t be automated, use people (less-human)
    - Frequently, this ends up being marginalized people, who later have their jobs automated



# Programming, historically

1940s

1970s



Jean Jennings (left), Marlyn Wescoff (center), and Ruth Lichterman program ENIAC at the University of Pennsylvania, circa 1946.

Photo: Corbis

<http://fortune.com/2014/09/18/walter-isacson-the-women-of-eniac/>

<https://s-media-cache-ak0.pinimg.com/564x/91/37/23/91372375e2e6517f8af128aab655e3b4.jpg>

# Modern Robots: Personal Computers



## How to talk your parents into parting with \$1300.

There's a new Apple Personal Computer called the II, that's so simple and so affordable that getting your parents to buy one should be easier than learning to sign.

It's that a, you know what to say. For example, just tell your parents that the II has the best true 128K 1152 keyboard, and built-in RS-172 ports and a built-in half-inch disk drive. Or that it has a respectable 96 40-character display and built-in measurements so it can use an Apple II mouse.

The funny thing is, it's not an 8-point computer, but all these specs

may make your parents uncomfortable. Just tell them that the Apple II can run more than 10,000 programs written for the Apple II's. It's the most popular computer in education at all levels, and it



works just the same as the Apple computer you learn in school.

You might also mention that it's a bargain. It comes with everything you need to start computing in one box - including an RF modulator that lets you hook it up to your TV for instant viewing.



get it done. There's even a free 4-minute video on computer basics that

you use when you're too busy to show them how.

All for under \$1,300\*\*

Of course, they probably won't want to hear that it runs more games than any other computer in the world except the Apple II.

But they might like to know that it also runs advanced business software. Including specialized programs for every profession from accounting to learning to programming. Not to mention personal productivity software to manage their

personal finances and letters.

Speaking of which, they can do just about anything with the Apple II's price from their letters if they use it for business.

Even if they always keep it at home.

That's another thing right now with the wide array of Apple II accessories and peripherals. Like Apple's 128K 300

modem, or the II's low-cost full color graphics, best printer, Scriber.

But even then, you can grow just as fast as you do.

Now, if all of these carefully measured arguments fail on your parents, don't despair. There's still



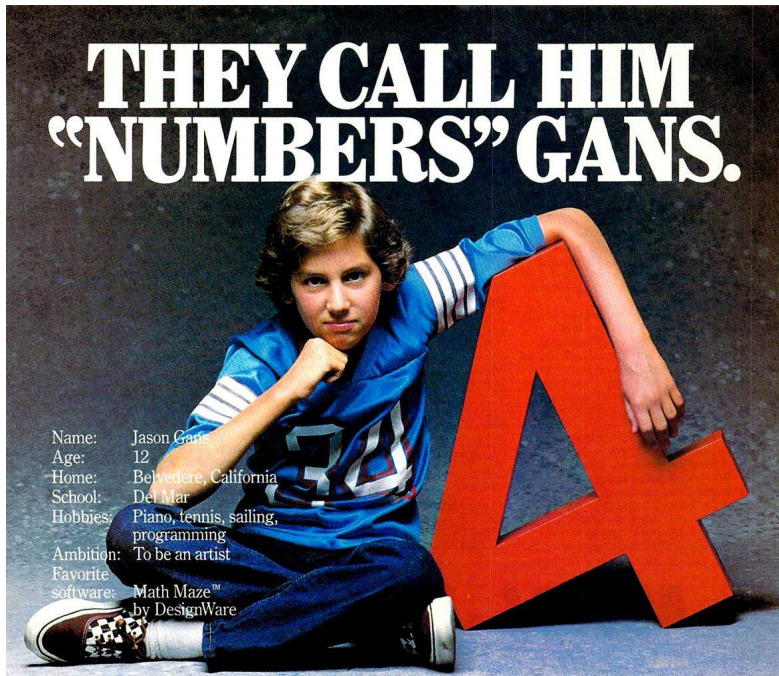
one thing more you can do. Get a paper trail.





# The advent of personal computing

## THEY CALL HIM "NUMBERS" GANS.



Name: Jason Gans  
 Age: 12  
 Home: Belvedere, California  
 School: Del Mar  
 Hobbies: Piano, tennis, sailing, programming  
 Ambition: To be an artist  
 Favorite software: Math Maze™ by DesignWare

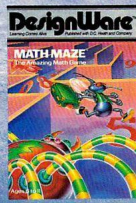
*"Math Maze is neat because you do more than just add and subtract numbers all the time. You've got to find them first. And then get there before you get caught. It's got real good graphics. I can even change the background color. And make the math as challenging as I want. There's lots of mazes, too. But the best thing is, I can make up my own. So when my friends come over, I've always got something new."*

**DESIGNWARE ON CREATIVITY.** Children learn the most through creative problem solving. That's why *Math Maze*, like so many DesignWare games, is an open-ended exercise that challenges and nourishes young minds. In a way that's a lot of fun. DesignWare programs encourage kids to draw on something they just happen to have an unlimited supply of—imagination!

Apple II is a registered trademark of Apple Computer, Inc. IBM is a registered trademark of International Business Machines. Atari is a registered trademark of Atari, Inc.

**DesignWare™**  
LEARNING COMES ALIVE

**SPELLING, MATH, OR LANGUAGE** Games like *Math Maze*, *Spellcrafter*™ and *Creator Creator*™ inspire youngsters to tap into that fertile idea-field. To actively become part of the program, in effect creating "new" games as they go along. And all the while building up solid skills in the basics. And all the while having a lot of plain old fun. DesignWare. We make learning come alive. On Apple II® IBM® and Atari® computers. Ask for DesignWare products at your local software retailer. Or call us at 800-572-7767 (in California, 415-546-1866) and ask for our free catalog. As Jason Gans says, "Hey—they don't call me 'Numbers' for nothing, you know!"



## What kind of man owns his own computer?



Rather revolutionary, the whole idea of owning your own computer? Not if you're a diplomat, printer, scientist, inventor... or a kite designer, too. Today there's Apple Computer. It's designed to be a *personal* computer. To uncomplicate your life. And make you more effective.

**It's a wise man who owns an Apple.**

If your time means money, Apple can help you make more of it. In an age of specialists, the most successful specialists stay away from uncreative drudgery. That's where Apple comes in.

Apple is a real computer, right to the core. So just like big computers, it manages data, crunches numbers, keeps records, processes your information and prints reports. You concentrate on what you do best. And let Apple do the rest. Apple makes that easy with three programming languages—including Pascal—that let you be your own software expert.

**Apple, the computer worth not waiting for.**

Time waiting for access to your company's big main-frame is time wasted. What you need in your department—

on your desk—is a computer that answers only to you... Apple Computer. It's less expensive than timesharing. More dependable than distributed processing. Far more flexible than centralized EDP. And, at less than \$2500 (as shown), downright affordable.

**Visit your local computer store.**

You can join the personal computer revolution by visiting the Apple dealer in your neighborhood. We'll give you his name when you call our toll free number (800) 538-9696. In California, (800) 662-9238.

Apple Computer, 10250  
 Bandley Drive,  
 Cupertino,  
 CA 95014.









# Modern Hardware: Historic Relics

1. “Boring, repetitive work” should be automated
  - Two narratives, both true: Automation/Augmentation
  - Consistently eliminating jobs of marginalized folks
2. Boring, repetitive work is “robot work”
  - Robot work should be performed by robots
    - *“Robot work” anything unvalued by the powerful, once including computing, programming*
  - If the task can’t be automated, use people
    - Frequently, this ends up being marginalized people
3. Augmentation is highly valued, and exclusive
  - “Boring, repetitive work” is more available
- We’ll get to earlier computational machines later!

# Breakout rooms!

- Join any breakout corresponding to your section
  - Section AA – Any room from A1 to A6
- I'll bring y'all back in 5 minutes

*Do you see these norms today? Where?*

We'll share out with Ohyay reactions!