Bits, bytes, words, and pages: How computers really operate

Greg Astfalk

May 16, 2019

Abstract

- We explain how information is represented within computers. This includes the computer's instructions. With this we can describe how computers, including your smart phones, actually operate. We offer some impressive numbers regarding the attributes of our processors and computers. Our expectation is that the audience will have a new and better understanding, and appreciation, of computing devices.

Prologue

- Our goal is to convey a deeper understanding and appreciation of how computers actually function
- We don’t need to be computer architects, electrical engineers, or scientists to understand the concepts
- What we discuss applies equally to smart phones, tablets, laptops, PCs, automobiles, smart refrigerators, servers, etc.
  - Independent of Android, IOS, macOS, Windows, or Linux

Caveats

- Some aspects of this talk are very complex in practice
  - We can’t cover such complexity in detail in this talk
  - A disclaimer is in the slide’s upper-right corner in most such places
  - If you want to know more details, ask 1on1 after the lecture
- This is not a Wikipedia-based talk
  - If you want to know my credentials to cover this topic, ask 1on1 after the lecture
- A few “bonus” photos at the end of the slide deck (enjoy)

Big numbers

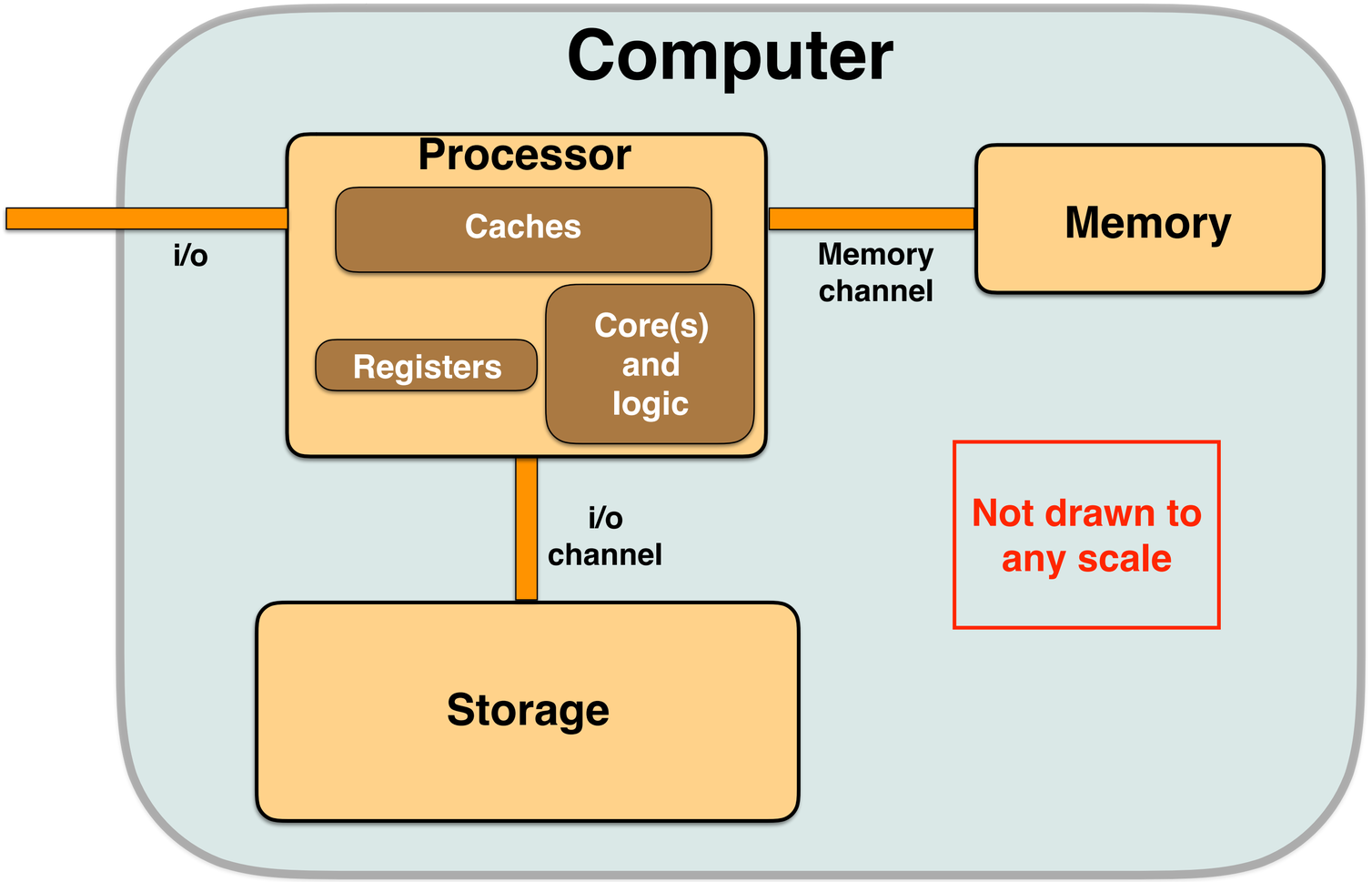
- We'll encounter some of these and we don't want them taken for granted

Kilo	2^{10}	1,024	Thousand	10^{3}	1,000
Mega	2^{20}	1,048,576	Million	10^{6}	1,000,000
Giga	2^{30}	1,073,741,824	Billion	10^{9}	1,000,000,000
Tera	2^{40}	1,099,511,627,776	Trillion	10^{12}	1,000,000,000,000
Peta	2^{50}	1,125,899,906,842,624	Quadrillion	10^{15}	1,000,000,000,000,000
Exa	2^{60}	1,152,921,504,606,846,976	Quintillion	10^{18}	1,000,000,000,000,000,000

NB: "Gazillion" is not a formally defined number; sorry…

Computer architecture

- Some familiarity with components of a computer's hardware will help in this lecture
- This very simplified cartoon's elements will suffice
  - Explanations follow



Architecture cartoon explained (1 of 2)

- Processor core(s) and logic
  - The core execute the instructions, “logic” is the other working areas
  - Almost exclusively binary logic
- Processor registers
  - A small number (~32) of fast, ephemeral holding places for data
- Caches
  - Ephemeral storage between registers and memory to improve performance (two types, explained later)

Architecture cartoon explained (2 of 2)

- Memory
  - Ephemeral repository for a executing program's information
  - Contents are "addressed" by a linear numbering (more later)
- Storage
  - Persistent and permanent repository for data and programs
- "Wires"
  - Not actually, but conductors connecting one physical location to another physical location

Bit

- A basic and, fundamentally important, aspect of all computing devices is the “bit”
- bit = **B**inary dig**IT**
- Binary means two
  - Hence a bit can only have one of two possible values

Bit values

- What values can a bit have?
- We use the terminology of 0 or 1
- We also (sometimes) say a bit is “on” or “off”
- 0 or 1 is the best choice of terminology
  - This has reasons rooted in arithmetic, as we will see

Important aside: Digital logic

- Electronic circuitry which operates on discrete values
  - It is safe to assume, for computers, the discrete values are binary
  - Example: today's microprocessors
- Contrast to "analog logic"
  - Operates on continuous, time-varying signals
  - Example: audio ADCs and DACs and "radios"

Bits in real-life

- How is a bit actually represented within a computer?
- Electronically
  - Details on next slide
- We often speak of “moving” bits
  - We don’t physically move a “thing”
  - We do move the “state” or signals representing bits

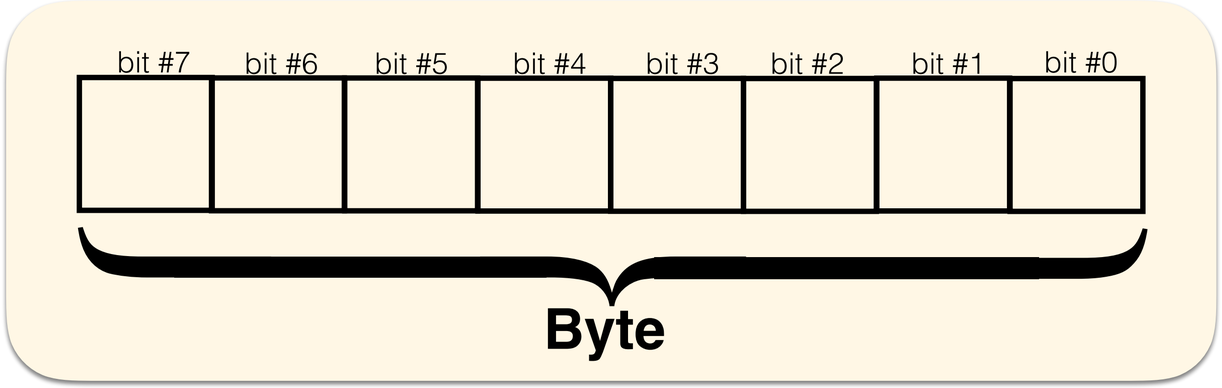
Bits in electronic form

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- A bit:
  - in the logic is generally a transistor or gate that is “on” or “off”
  - in a register is (generally) the “state” in a flip-flop latch
  - in a cache is dual-inverter state in a 6T (i.e., 6 transistor) SRAM cell
  - in memory is the presence or absence of capacitive charge
  - in storage is
    - trapped charge in a special type of transistor (phones and tablets)
    - a small magnetized area on a disk (servers and some laptops)
  - on a “wire” is either a positive, negative or zero voltage

Byte

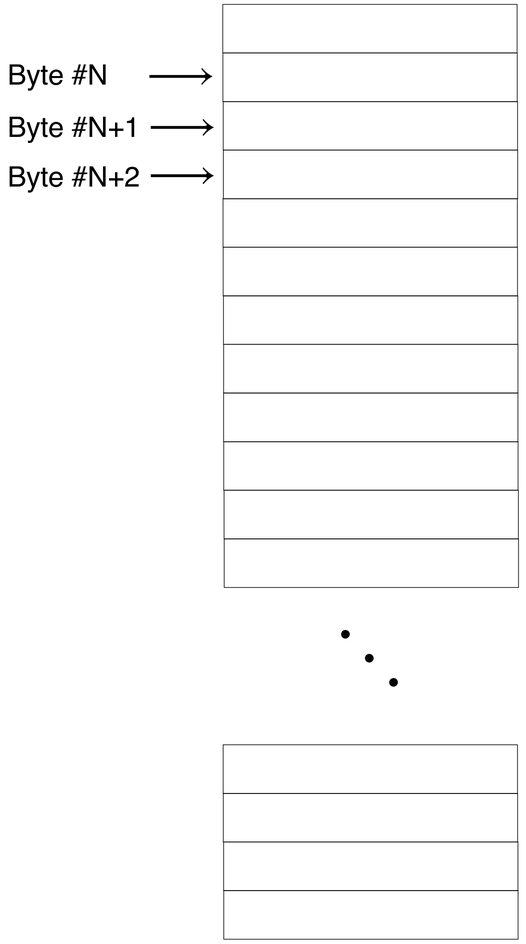
- A byte is nothing more than 8 bits treated together
  - A processor can treat a subset of a byte but never breaks it apart
- A byte is the smallest entity that a processor can “address”
- This is no more complicated than it seems



Bytes and memory

Disclaimer: There is more complexity to this topic than what is presented on this slide.

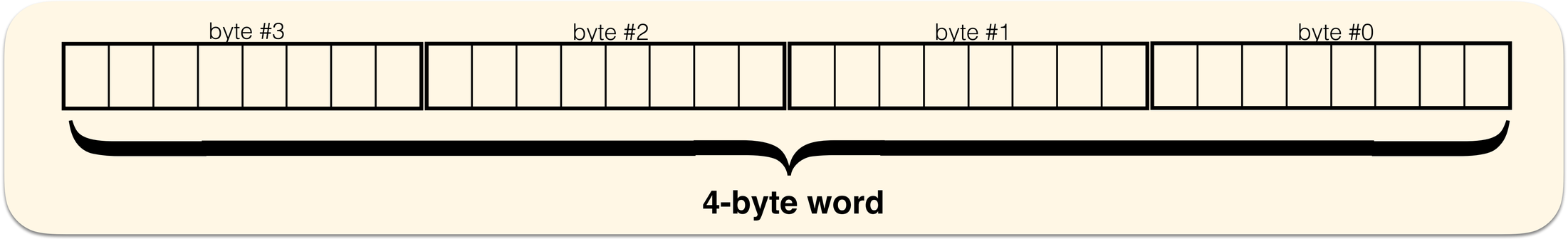
- The memory of a computer is a very large set of bytes
- The bytes of memory are numbered sequentially
- Addressing memory is, of necessity, complex with several types of addresses
  - Virtual addresses
  - Physical addresses



Word

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- A “word” is either 4 or 8 bytes in size
  - The notation 32-bit word or 64-bit word is also commonly used
- The bytes of a word are always stored contiguously in the registers, caches, memory, and storage



Page

- Our final structural construct is the “page”
- A page is
  - Merely a set of contiguous bytes
  - Generally, but not restricted to, 4,096 bytes in size
  - In some cases pages are huge (i.e., Gbytes in size)
    - Done for efficiency purposes
- Reminder: bits ⇒ bytes ⇒ words ⇒ pages

Moving on…

- We have defined the structural aspects of computer data “containers”, i.e., bits, bytes, words, and pages
- We now shift to the topic of number representation
  - This is a slight but important step sideways to cover number “bases”

Base-10 numbers (1 of 2)

- This may sound new and complex
  - It isn't!
- We really know base-10 numbers and arithmetic very well
- We use it all the time without even being aware
- Consider the number 6,728,441
  - This says that we have 6 millions, 7 hundred thousands, 2 ten thousands, etc.

Base-10 numbers (2 of 2)

- The first, rightmost, digit represents the number of 1s
- The second digit represents the number of 10s
- By induction the n^{th} digit represents $10^{(n-1)}$
  - First digit: $10^0 = 1$
  - Second digit: $10^1 = 10$
  - Third digit: $10^2 = 100$
  - etc.

Base-2 numbers (1 of 3)

- Base-2 numbers merely use 2, rather than 10, as the base
- We humans don’t need to directly use base-2 numbers
  - We cover it for understanding how computing devices operate
- Important point
  - Since the base is 2 (i.e., binary) it is a natural match to bits

Base-2 numbers (2 of 3)

- In a base-2 number:
  - The rightmost digit is the number of 1s (2^0)
  - The second digit is the number of 2s (2^1)
  - The third digit is the number of 4s (2^2)
  - The fourth digit is the number of 8s (2^3)
  - etc.
- Other than the base this is no different from our comfortable understanding of base-10 numbers

Base-2 numbers (3 of 3)

- For exposition let's reconsider our earlier, and arbitrary, base-10 number, 6728441
- In base-2 this same number is
  - 11001101010101011111001
- Right, this looks worse, not better
  - Worse for us
  - (Much) better for computers

Base-2 arithmetic, finale

- Whether we use base-10 or base-2 the number represents the same number of marbles, people, cats, etc.
- What we learned in grade school about borrowing, carrying, multiplying, etc. still applies in base-2 arithmetic
  - The logic in a processor is significantly simpler, more compact, and much faster if we use base-2 arithmetic
    - There is synergy among bits, base-2 arithmetic, and digital logic

Moving on…

- We are now conversant with the data “containers” and base-2 representation of numbers
- We now shift to the topic of what types of data are held in these containers and how they are represented
- Reminder: we want all our data to be binary since it is synergistic with digital logic

Data types

- Almost all the types of data which a computer deals with are
  - Characters
  - Integer numbers
  - Floating-point numbers
  - Application-specific sets of bits
    - For example, the bits used to represent our photos and videos
    - We will not discuss this further in this lecture
  - Bits internal to the processor for its operation
    - We will not discuss this further in this lecture

Character data (1 of 3)

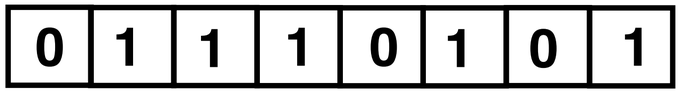
- Computers are always showing us characters
  - What we read in email, txt messages, web pages, etc. are characters
  - Hence it is a common type of data
- We (generally) store a single character in a single byte
- A character is merely a number stored in a byte

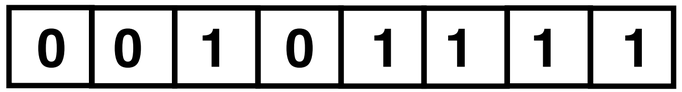
Character data (2 of 3)

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- ASCII is a industry standard (circa early 60's) for characters
  - ASCII = American Standard Character Information Interchange
  - A mapping (aka "encoding") of characters to numbers
- There is an extension beyond ASCII called "unicode"
  - Uses more than one byte per character, but still a character to number mapping/encoding
  - Allows for many more characters, say all the Kanji characters
  - Widely used today, but no further discussion in this lecture

Character data (3 of 3)

- Consider two examples
- The lower-case letter “u” character in ASCII
  - “u” = 117 = 1110101 = 

- The slash “/” character in ASCII
  - “/” = 47 = 101111 = 

- Note: base-2 representation of the number (aka “character”)

The ASCII codes (only for completeness)

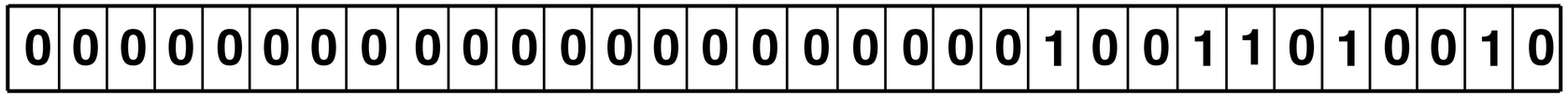
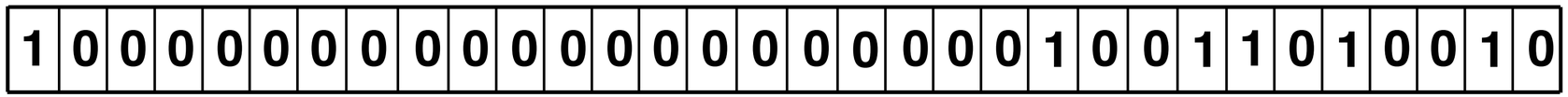
0	nul	1	soh	2	stx	3	etx	4	eot	5	enq	6	ack	7	bel
8	bs	9	ht	10	nl	11	vt	12	np	13	cr	14	so	15	si
16	dle	17	dc1	18	dc2	19	dc3	20	dc4	21	nak	22	syn	23	etb
24	can	25	em	26	sub	27	esc	28	fs	29	gs	30	rs	31	us
32	space	33	!	34	“	35	#	36	$	37	%	38	&	39	‘
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	l	125	}	126	~	127	del

The **blue** characters are non-printable.

Number data (1 of 3)

- We represent the integers (natural numbers if you prefer) in a word of contiguous bytes
- A 4-byte word has 32 bits so the largest possible integer is
  - $2^{32} - 1 = 4{,}294{,}967{,}295$
- An 8-byte word has 64 bits so we can represent
  - $2^{64} - 1 = 18{,}446{,}744{,}073{,}709{,}551{,}615$

Number data (2 of 3)

- But what about negative numbers?
- We allocate 1 bit, the Most Significant Bit (MSB), of the word to be the “sign” bit
  - Reduces the largest representable number by a factor of 2
- Example
  - +1234 = 
  - –1234 = 

Number data (3 of 3)

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- There is a possible complexity with integer arithmetic
- The value of a number can “overflow”
  - The desired number is greater than the largest representable number
  - Consider, for 32-bit integers, the multiplication (2,147,483,647) × (8) which produces a number $>2^{32}$
- The processor’s logic handles this and signals an error
  - Well written software will alert the user to such an error

More general numbers

- The integers can not deal with numbers such as 283453.7, 1.0001 or 3.14159265358979323846
  - These are referred to as floating-point numbers
  - Mathematically, they include the rationals and irrationals
- Need a more complex representation than for integers
- Owing to the complexity we will only cover this briefly
  - Well okay, we will use five slides to cover it

Floating-point numbers (1 of 5)

- We first note that any floating-point number can be represented in the canonical format
  - $\pm x.y \times 10^{(\pm z)}$
- We refer to this as “scientific notation”
- Examples
  - $12.0 = 1.2 \times 10^{1}$
  - $-456665345252.23 = -4.566653452522 3 \times 10^{11}$
  - $0.0998299 = 9.98299 \times 10^{-2}$

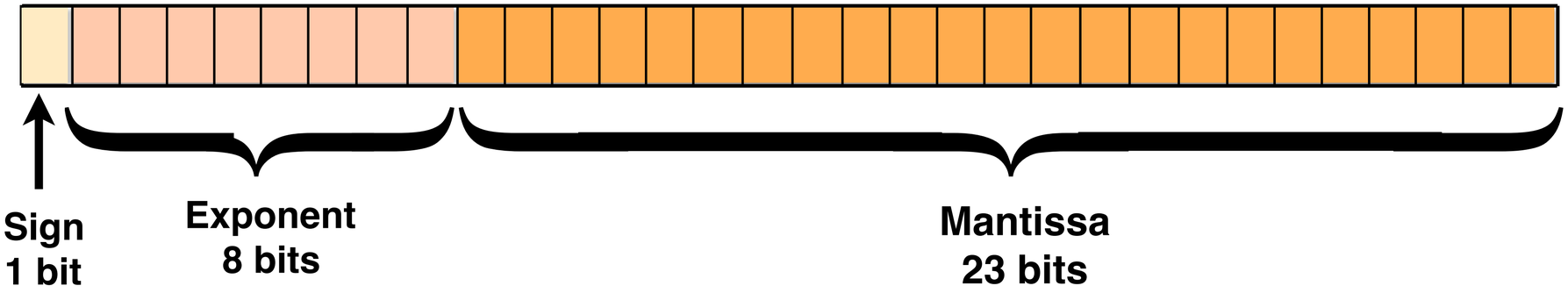
Floating-point numbers (2 of 5)

- In the canonical form for floating-point numbers we need to represent five things in the finite number of bits in a word, either 32-bit or 64-bit
  - Base (2 by definition so no need to be represented)
  - Exponent
  - Exponent sign (we use a “shift” rather than an explicit sign)
  - Mantissa (also called the “fraction” or “significand”)
  - Sign

Floating-point numbers (3 of 5)

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- A standard is used throughout the computer industry
  - Formally the “IEEE 754” standard
- The bit-level layout for a 32-bit floating-point word



- The exponent and mantissa are represented in base-2

Floating-point numbers (4 of 5)

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- Since we use base-2 for the representations, bits and binary logic are perfect tools for floating-point arithmetic
- The processor's logic handles all the usual arithmetic operations (+ – × ÷) and exceptions like overflow, etc.
- Floating-point logic is more extensive than for integers
  - However it is very modest relative to the overall logic of the processor

Floating-point numbers (5 of 5)

- What exactly can we represent with the IEEE 754 floating-point numbers

	precision in decimal digits	smallest number	largest number
32-bit word	~8	1.17549×10^{-38}	$3.40282 \times 10^{+38}$
64-bit word	~16	2.22507×10^{-308}	$1.79769 \times 10^{+308}$

When 1 does not look like 1…

- Recall that a 32-bit integer 1 is
  - 00000000000000000000000000000001
- 1.0 as a 32-bit IEEE-754 floating-point number is
  - 00111111100000000000000000000000
- The ASCII character “1” is
  - 110001
- Strange? How does the processor know which to use?

Important understanding about data

- The computer's *internal* data representations are designed for binary logic efficiency, compactness, and speed
- The software we use presents the data *externally* in human-readable forms
- One of the pleasant, user-friendly things about our digital devices

Moving on…

- We now know about data “containers”, base-2, and the common data types
- Now shift our focus to how we tell the processor what to do
  - This topic is more complex than the preceding material

Instructions

- We tell computers, specifically the processor, what to do for us by giving it *instructions*
  - Also called “code” or “machine instructions”
- Programs, apps if you prefer, are written in high-level programming languages† and then compiled in to machine-level instructions

† For example: Fortran, C, Java, Cobol, Python, SQL, Ruby, etc.

Instructions, cont’d

- A processor instruction generally causes one simple thing, and one simple thing only, to occur
- Owing to this there is a large expansion from programming language statements to machine-level instructions
- An application may have (many) millions of machine instructions

Instruction data

- This slide’s title is not a contradiction
- Instructions are merely bits stored in words
- The words containing our data are not fundamentally different than the words containing the instructions
  - Both are a set of bytes, containing bits, addressable in the memory
- At this point you may say, “Huh?”
  - As you should, but patience please

Instruction execution

- An instruction is presented to the processor's core
- The core's logic determines the action(s) it needs to perform based on the bits (i.e., format) of the instruction
- Multiple instructions may complete in a single processor "cycle"
  - Think of the processor cycle as a heartbeat or metronome
  - More specifics to follow

Instruction types

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- Processor instruction formats are **very** complex
- Even for a specific processor different instruction types have different formats
- For simplicity of exposition we consider a generic (i.e., “made up”) instruction format which is not from Intel, ARM, IBM, MIPS, or SPARC

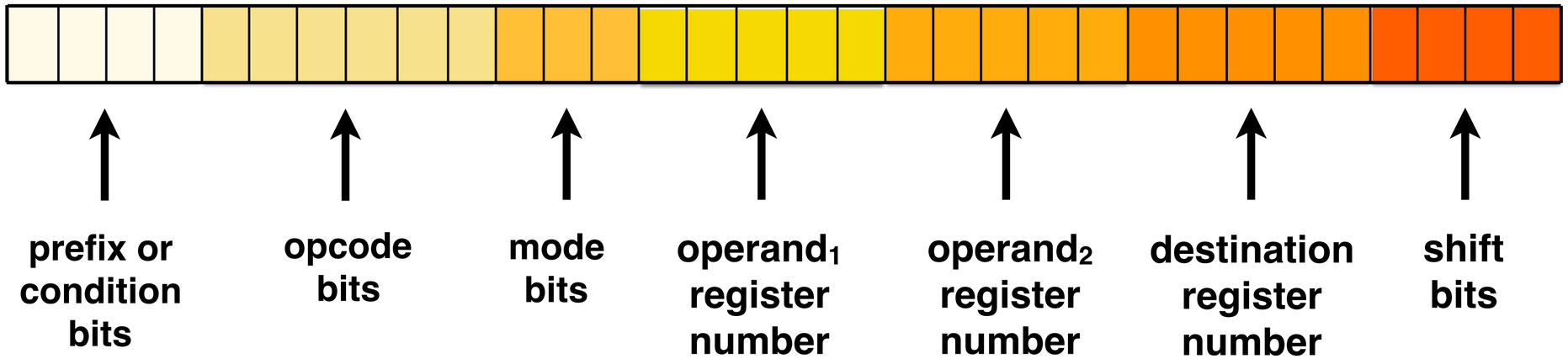
An instruction deconstructed

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- Instructions have several “fields” that specify
  - the operation itself
  - the operand(s)
  - variations on the instruction’s behavior
  - conditions for execution, or not
  - etc.

Instruction format (made up)

- The word containing an instruction is merely a set of 32 bits
- The processor's instruction logic decodes the instruction's bits and "does the right thing"



Instruction statistics

- ARM and Intel together dominate the processor market
- ARM and Intel each have several hundred instructions
  - Many of these instructions have multiple variations
- Examining the instructions executed by most applications
  - Only a small set of the possible instructions are executed most of the time

von Neuman architecture

- Essentially every computer ever made has what is called a “von Neumann” architecture
  - Named after the World War II era mathematician and polymath John von Neumann
- The key defining characteristics are:
  - A central processing unit for arithmetic and logic (aka the “core”)
  - Data and instructions co-exist in memory ⟵ **Our immediate focus**
  - Storage external to the processor
  - Input/output capabilities

A final puzzle piece

- How do we differentiate between *data* words and *instruction* words?
  - They coexist in memory owing to the von Neumann architecture
  - Picking an arbitrary word in memory, in isolation, it is generally not possible to know if it is data or an instruction
- The key is the “page” which we learned about earlier
  - Reminder: bits ⇒ bytes ⇒ words ⇒ pages

Types of pages

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- The computer's operating system understands, at least, two different types of pages
- For historical reasons the common terminology is
  - "text" pages containing only instructions
  - "data" pages containing only data
- The operating system and hardware knows whether to treat the page's words as instructions or as data
  - Complicated but it (obviously) does the right thing

Instructions vs. data

Disclaimer: There is more complexity to this topic than what is presented on this slide.

- The logic of the processor routes instructions from “text” pages to the I-cache (Instruction cache)
  - From the I-cache instructions go to the core’s functional units
    - Adder, multiply, compare, memory management, etc.
- The logic of the processor routes data from the “data” pages to the D-cache (Data cache) and also to registers
  - Instructions which operate on data require the data to be in registers
    - Almost always

Big picture side-bar

- An “app” is a collection of instructions and data
- An app is loaded from storage to memory by the operating system
  - Instructions and data are separated onto the appropriate page type
- The app’s instructions, and data, are “requested”, as needed, by
  - The core’s logic
  - The instructions

Moving on…

- We have covered all the planned technical material
- Let's do some "show and tell" to make tangible sense of processors and data
  - [*Pass around a few computer components*]
- The next few slides have some numbers that may seem hyperbolic to you
  - They are not exaggerated

Semiconductors

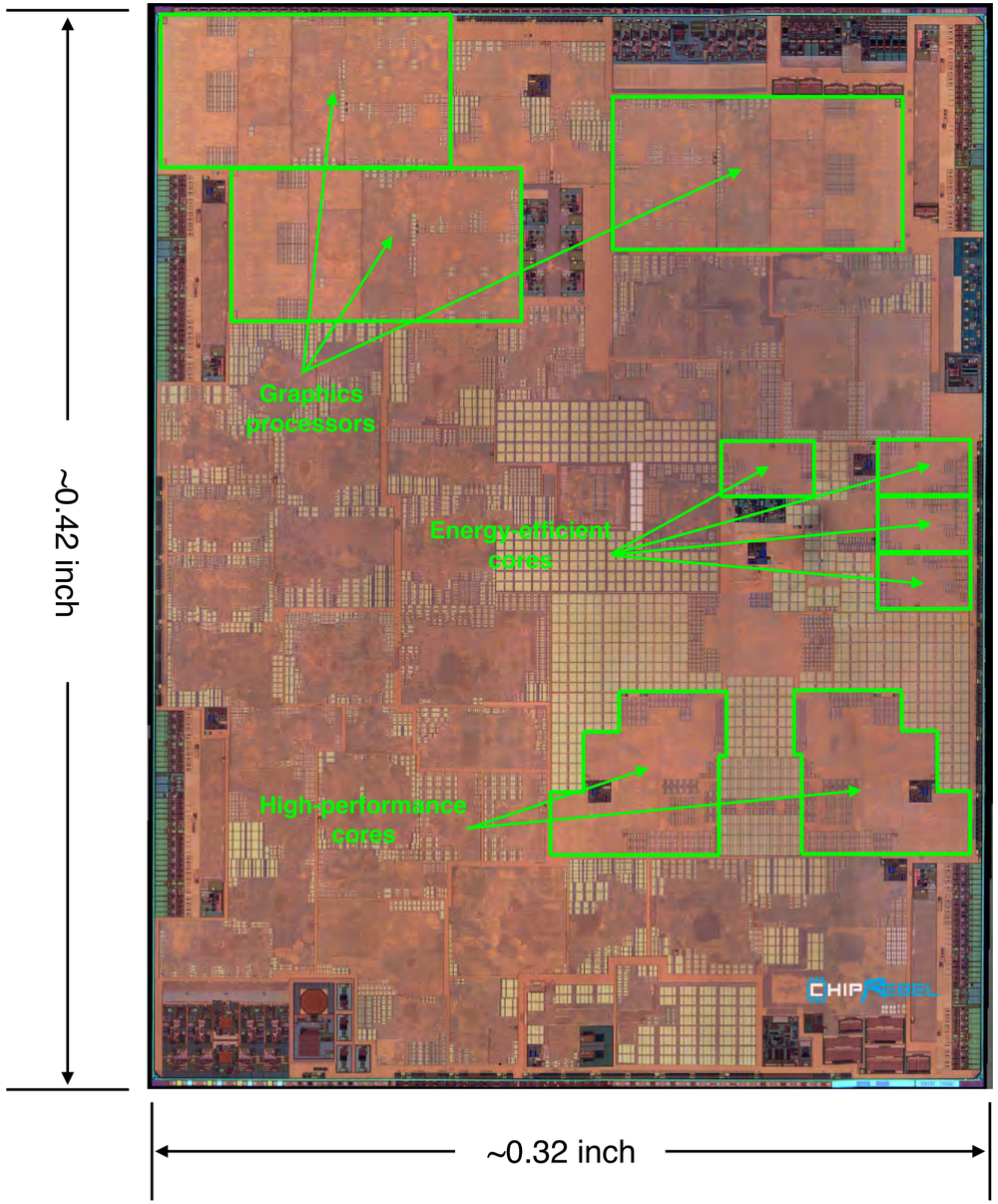
- The major functional components of computers are semiconductor die
- Dies are small pieces of silicon with binary logic on them
- Semiconductors are a important, fascinating, and impressive technology

How small is small?

- Finest semiconductor “pitch” today is ~10 nanometers
  - Pitch is the distance between two parallel conducting paths
  - Full disclosure: the industry is not consistent in defining pitch
- Human hair is 0.00066 to 0.0071 inches (17,000 to 180,000 nanometers) in diameter
- There can be 1,700 to 18,000 parallel semiconductor conducting paths across the width of a single human hair

How many is many?

- Apple's A11 ARM SoC
  - iPhone 8 and X
  - 10 nm pitch
  - 2.39 GHz
  - 87.6 mm^2 die
    - Approximately half the size of a postage stamp
  - 4.3 billion transistors



How fast is fast?

- Humans can barely distinguish 0.10 of a second
- The processor we showed on the previous slide can execute 2.4 billion instructions per second†
  - Thus it takes 0.000000000416 seconds (0.4 nanoseconds) to execute an instruction
- In practice, processors only execute productive instructions at a fraction of this rate

† Actually more than this but this is a story for another day

How many bits?

- Let’s consider a new high-end iPhone
  - It could have as much as 512 Gbytes of storage
- Hence, inside this iPhone there may be as many as
  - 4,398,046,511,104 bits
- In words; more than 4 trillion bits in a phone!
  - Even a lower-end version will have ~1 trillion bits

How much (or little) power?

- A processor in a datacenter server produces ~100 Watt
  - Silicon die less than 1 inch by 1 inch and ~0.01 inch thick
  - Sophisticated cooling is necessary (see later photos)
- A processor in a smartphone produces ~1 Watt
  - The variance of the power is large (very activity dependent)
  - Power usage is bursty
  - Generally low duty cycle
  - Processor switches to lower power core(s) when possible

Snapshot of processor/computer operation

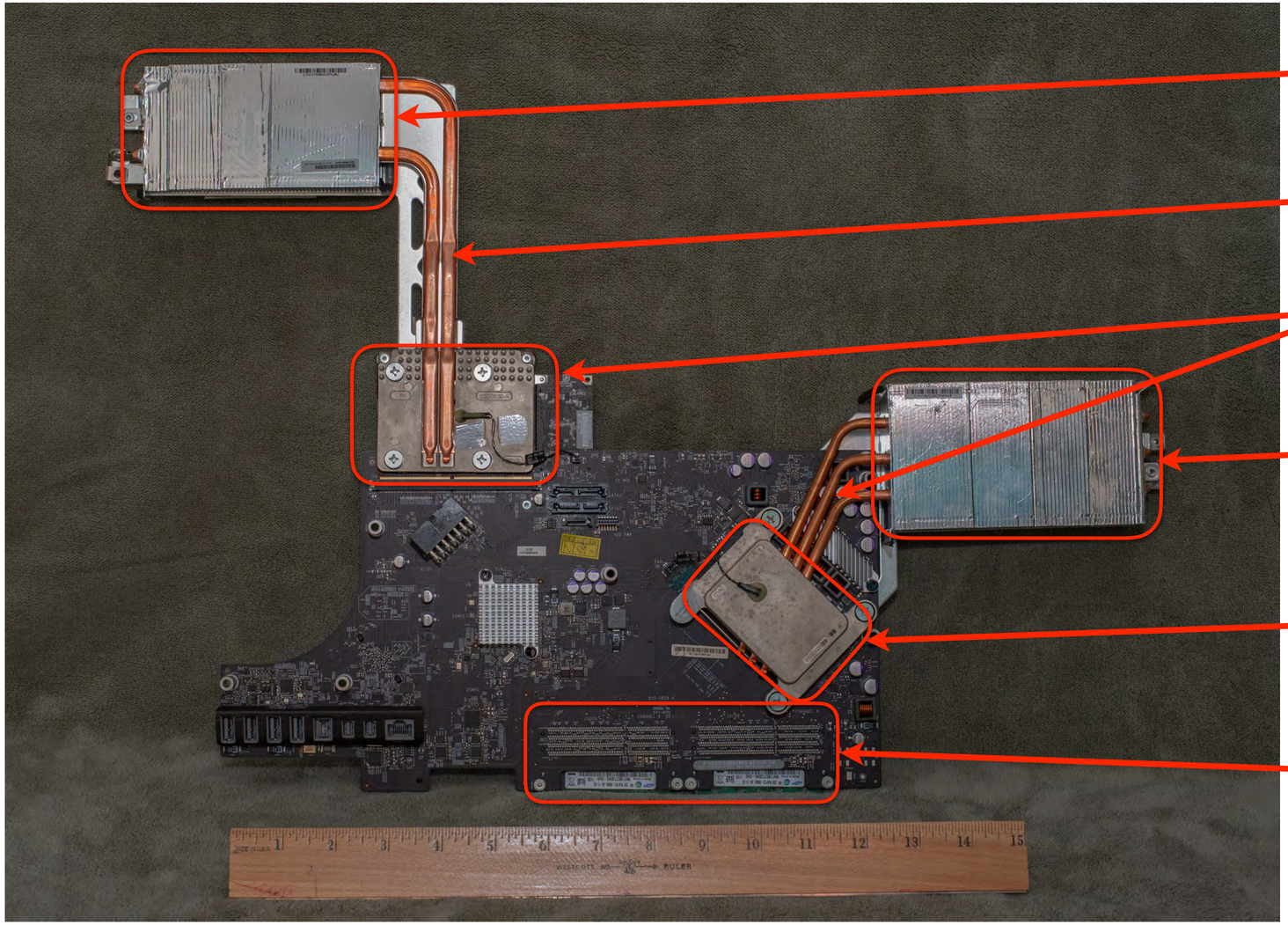
- At any given instant in time your computer, smart phone, tablet, etc. has billions of bits under its control
- Many of these bits are simultaneously moving within the processor and computer at approximately 110,000 miles per second
  - Common misunderstanding is that they move at the speed of light (186,282 miles per second); they don't
    - For specifics, ask 1on1 after the lecture
  - Recall that "bits", as a physical thing, don't actually move

A brief recap

- Bits are electronic 1s or 0s
- Bits ⇒ bytes ⇒ words ⇒ pages
- Bytes and words hold our data and computer instructions
- Not many types of data are in common use
- Processors execute instructions amazingly fast
- Computer storage can hold huge amounts of data
- Many complexities we did not discuss

Questions?

25” iMac motherboard, circa 2011



Graphics processor radiator

Heat pipes

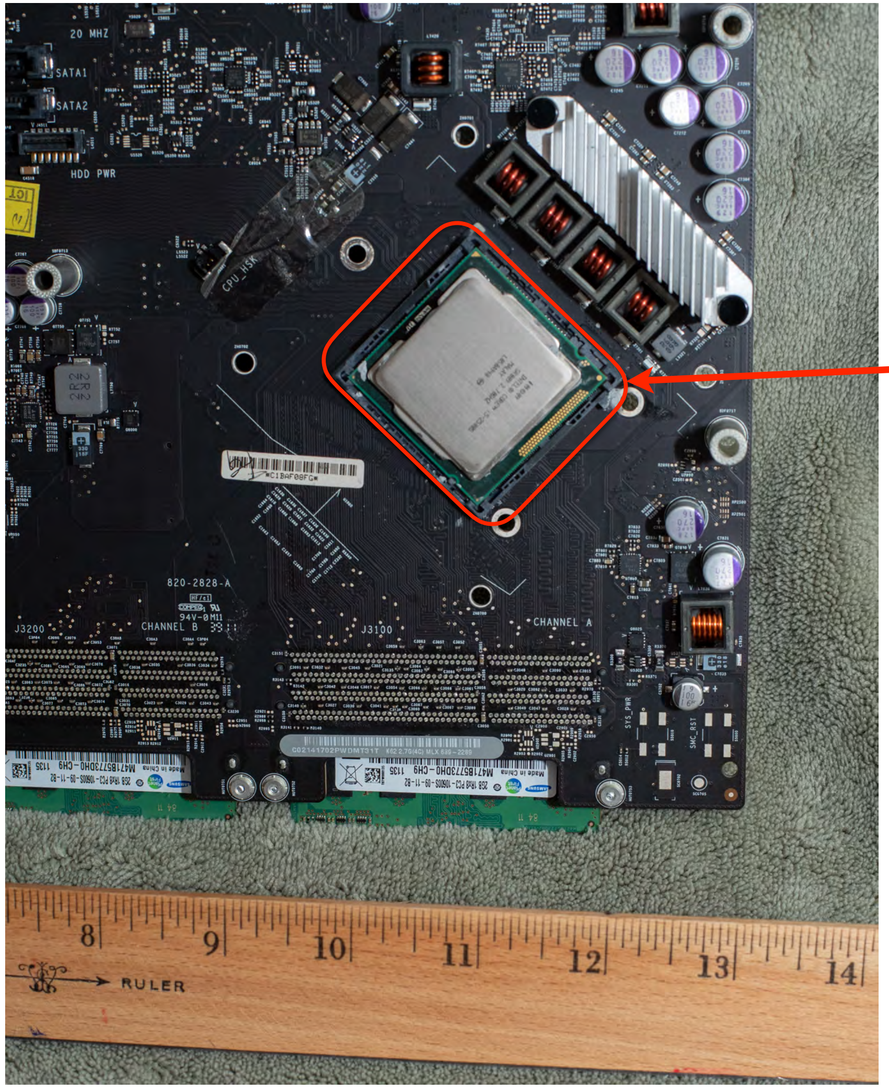
Graphics processor

Processor Radiator

Processor

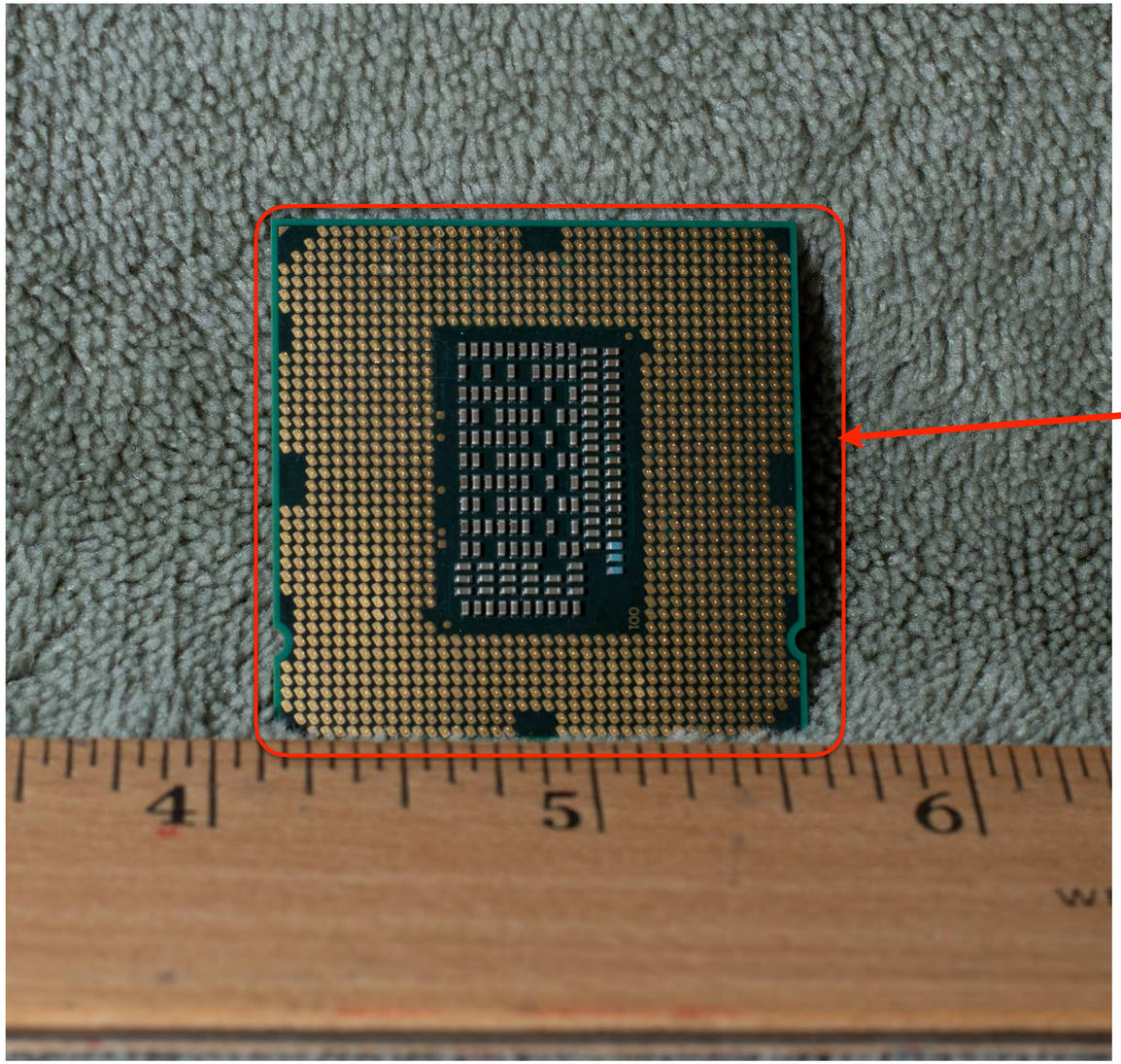
Memory (on back-side of the board)

25” iMac processor, circa 2011



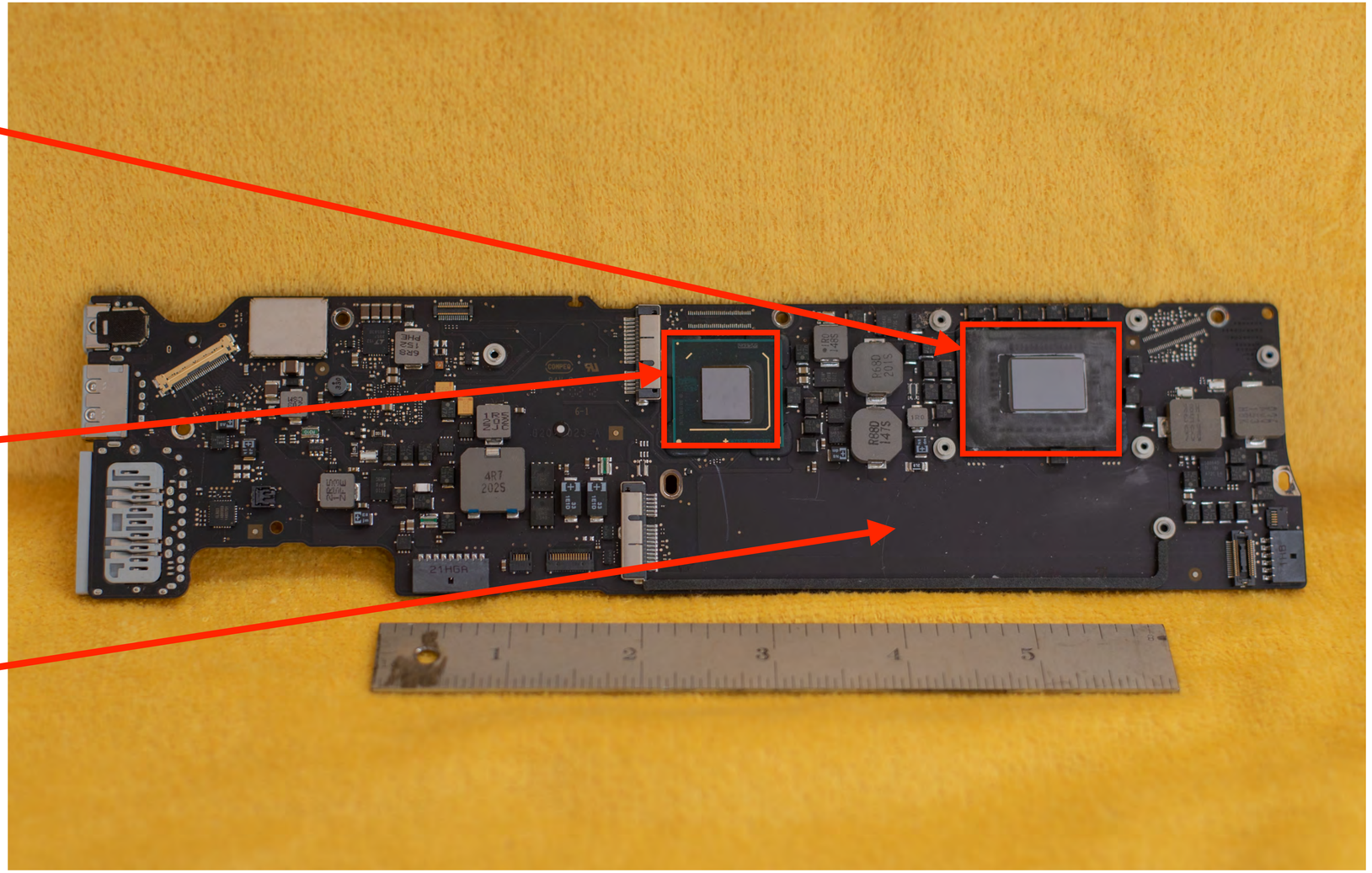
Processor with heat-sink cap

25” iMac processor, circa 2011



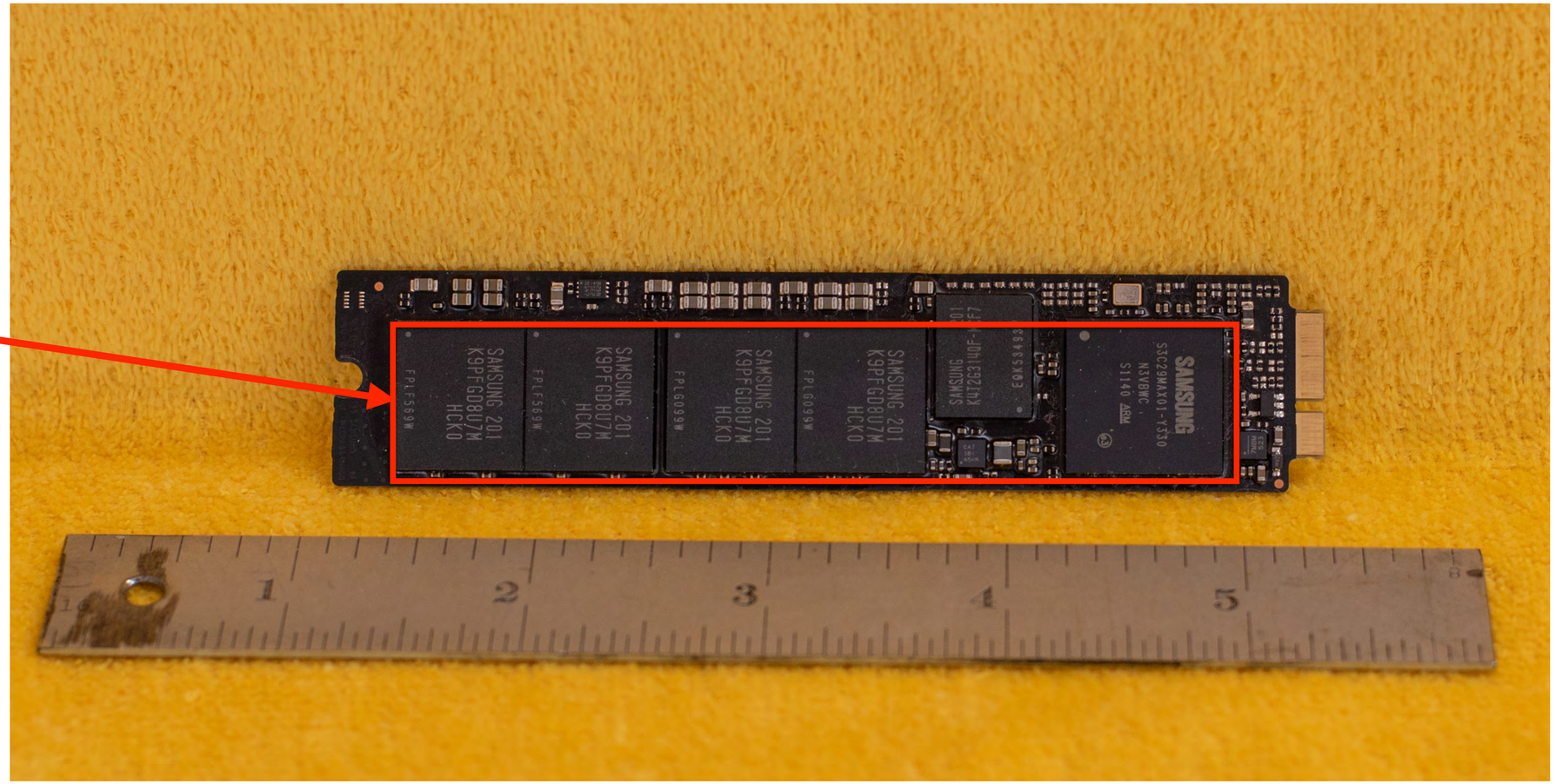
Processor chip-carrier
back side pin array
(~1500 pins)

MacBook Air, motherboard, front, circa 2011



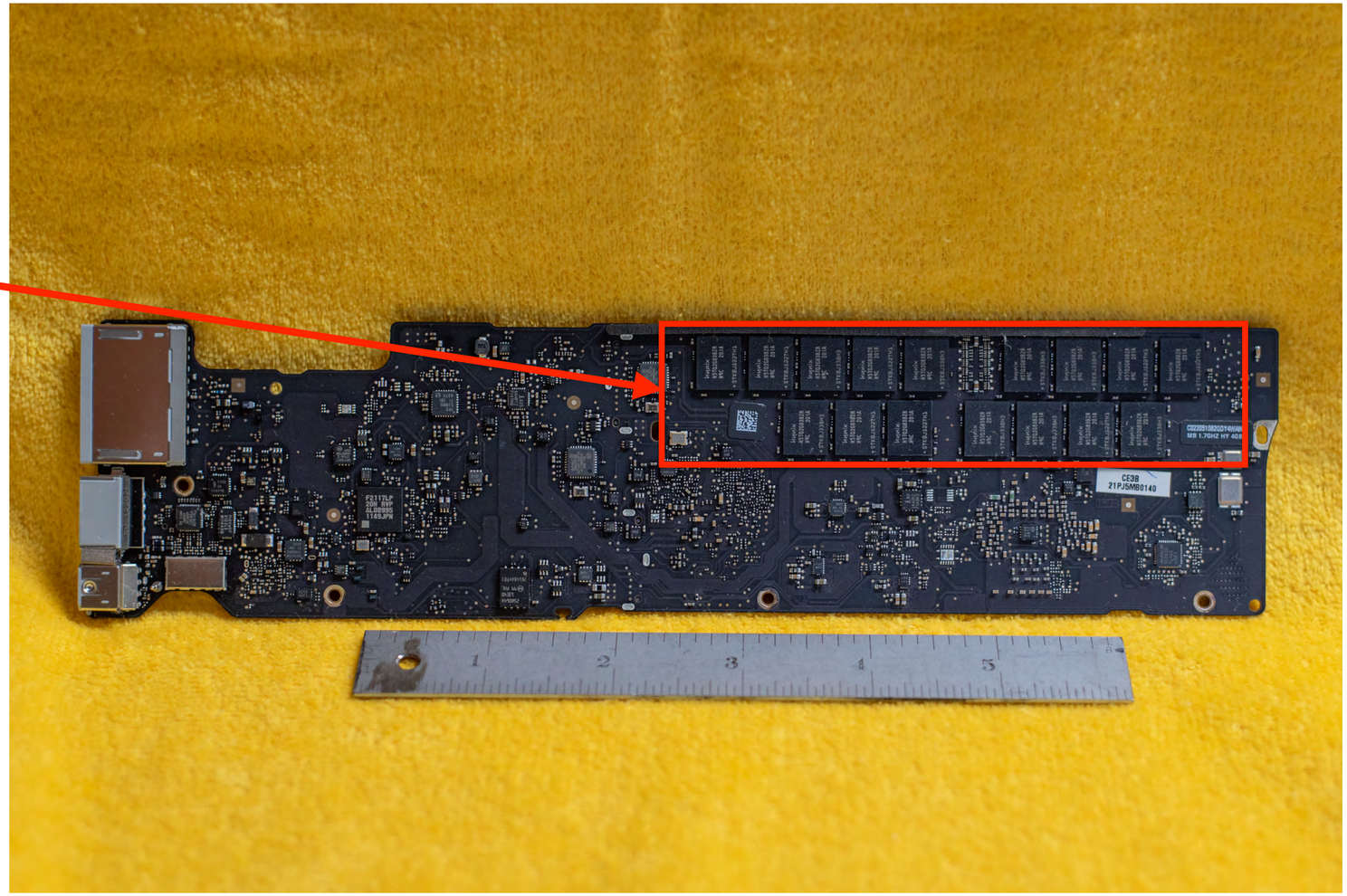
MacBook Air, SSD, circa 2011

SSD
NAND Flash

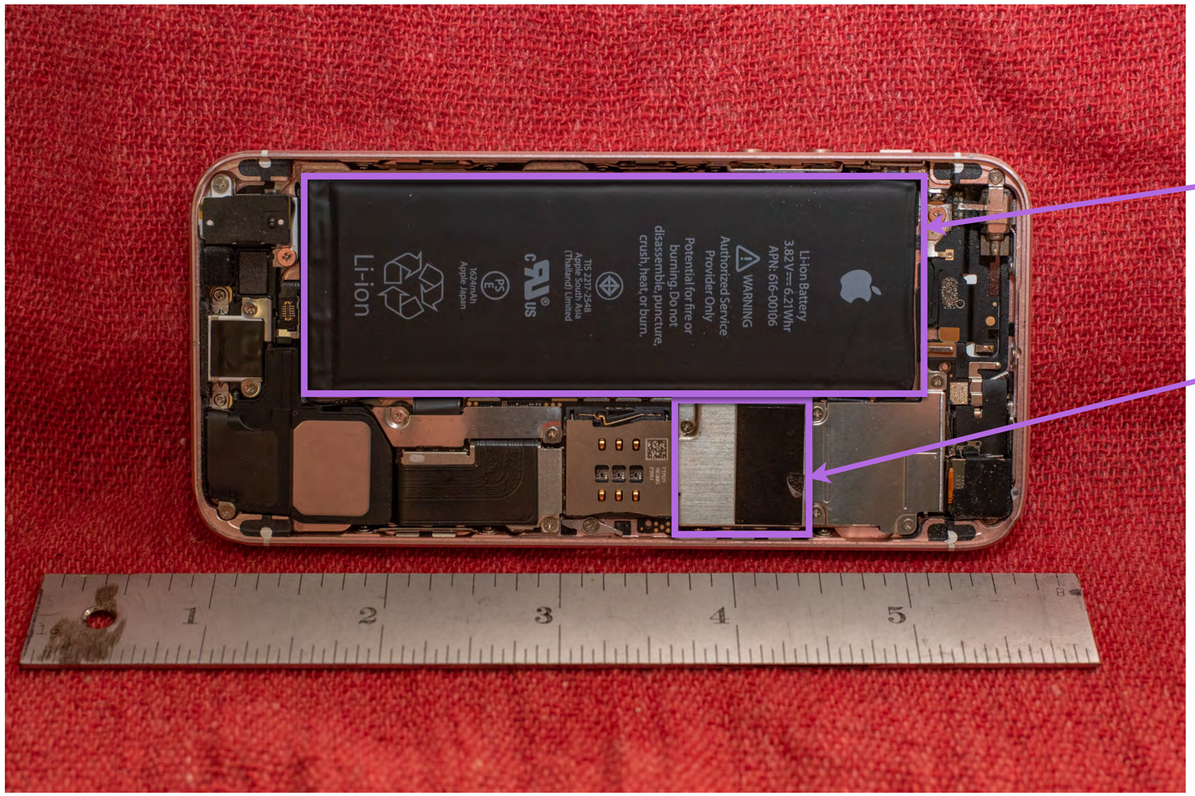


MacBook Air, motherboard, back, circa 2011

Memory
DRAM



iPhone SE, circa 2016



Battery

Processor

Classic HDD, 1 Tbyte, circa 2015