OPCUG Computer – It is all about the bits

STEPHANE RICHARD

Questions that will be answered in this presentation

- Why are bits so important in computing?
- How does a computer store text, number and images?
- What are the major components in a computer?
- How does a Central Processing Unit (CPU) work?
- What is the assembly language?
- What is miniaturization?
- If you have a question during the presentation, please interrupt

First thing first – what is a bit?

- A bit or binary digit is the smallest unit of data that a computer can process and store.
- A bit is always in one of two physical states, similar to an on/off light switch.
- The state is represented by a single binary value, usually a 0 or 1. However, the state might also be represented by yes/no, on/off or true/false.
- For example. the Transistor-Transistor Logic (TTL) hardware is powered with 5 V and uses signal level < 0.8 V as False and > 2 V as True. If the signal level is between 0.8 and 2 V, the result is not guaranteed.
- One bit only has limited utility, but many bits are very useful!
  - What they mean depends on the reader, for example

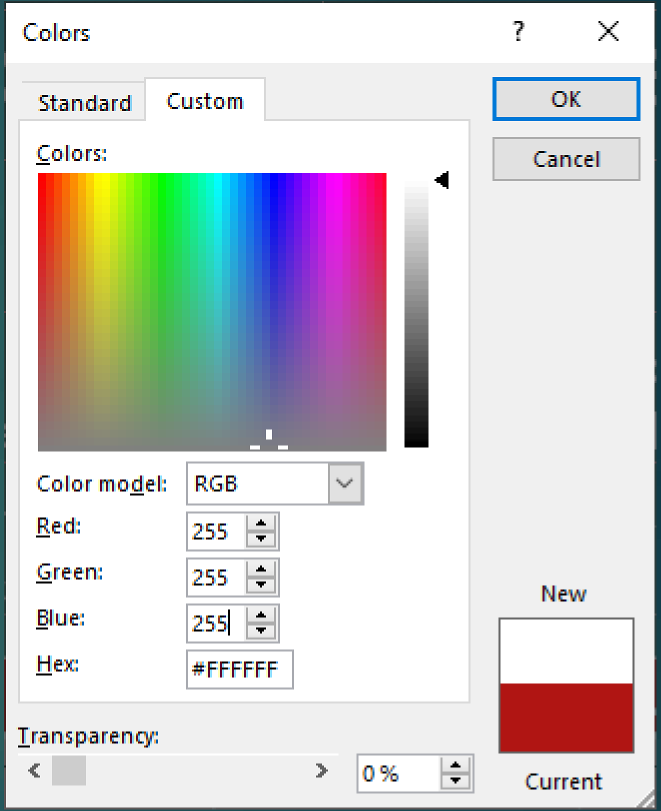
What could this string of bits 01001100 mean?

- It could represent the decimal number 76 or hexadecimal 4C.
- It could represent the letter L in the American Standard Code for Information Interchange (ASCII) character code.
- It could tell a Motorola 6800 Microprocessor to increment the accumulator A by 1.
- It could tell an Intel 8080 Microprocessor to move the content of register C to register H.
- The Red Green Blue (RGB) color model use 8 bits to code each of the red, green, and blue colors:
  - It could represent this red color in the RGB color model.
  - It could represent this green color in the RGB color model.
  - It could represent this blue color in the RGB color model.
- It could mean peace for the Gray Alien.
- So who knows what it mean, it is the reader of the string of bits that will decide!

How does a computer store color?

- There are various models available to encode color. The most common is the Red Green Blue (RGB) color model.
- It uses 8 bits to encode of the each of the red, green, and blue colors for a total of 24 bits.

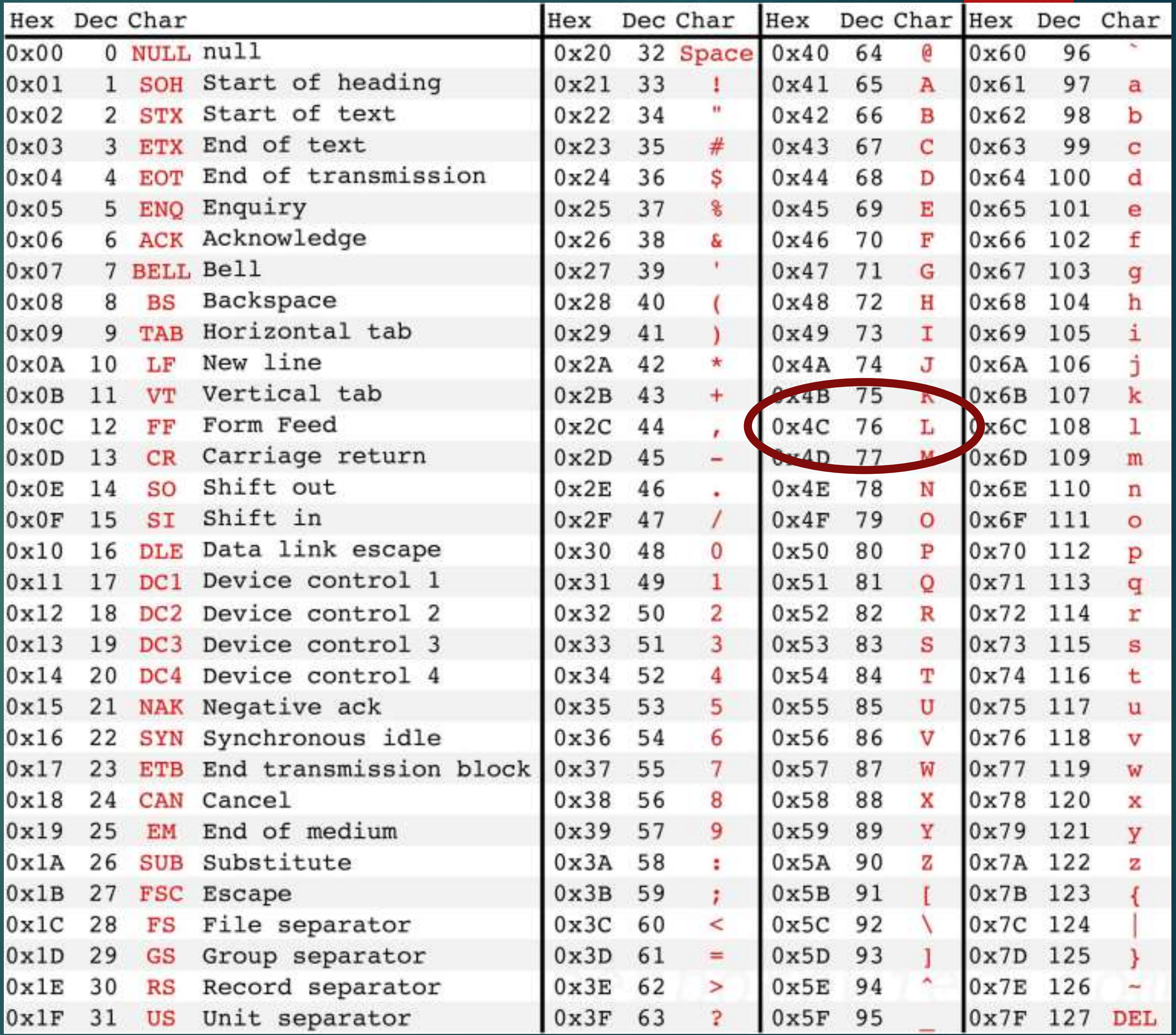
Color	Red	Green	Blue
⬜	11111111 (255)	11111111 (255)	11111111 (255)
⬛	00000000 (0)	00000000 (0)	00000000 (0)
🟥	11111111 (255)	00000000 (0)	00000000 (0)
🟩	00000000 (0)	11111111 (255)	00000000 (0)
🟦	00000000 (0)	00000000 (0)	11111111 (255)
🟦	00000000 (0)	00000000 (0)	01001100 (76)
🟨	11111111 (255)	11111111 (255)	00000000 (0)
🟪	11001010 (202)	00110110 (54)	11000100 (196)



Color model: RGB
Red: 255
Green: 255
Blue: 255
Hex: #FFFFFF
Transparency: 0 %

How does a computer store and process text?

- By using standardized character coding such as ASCII (shown on the right) and other character coding tables.
- Letters, numbers, and symbols become string of bits.



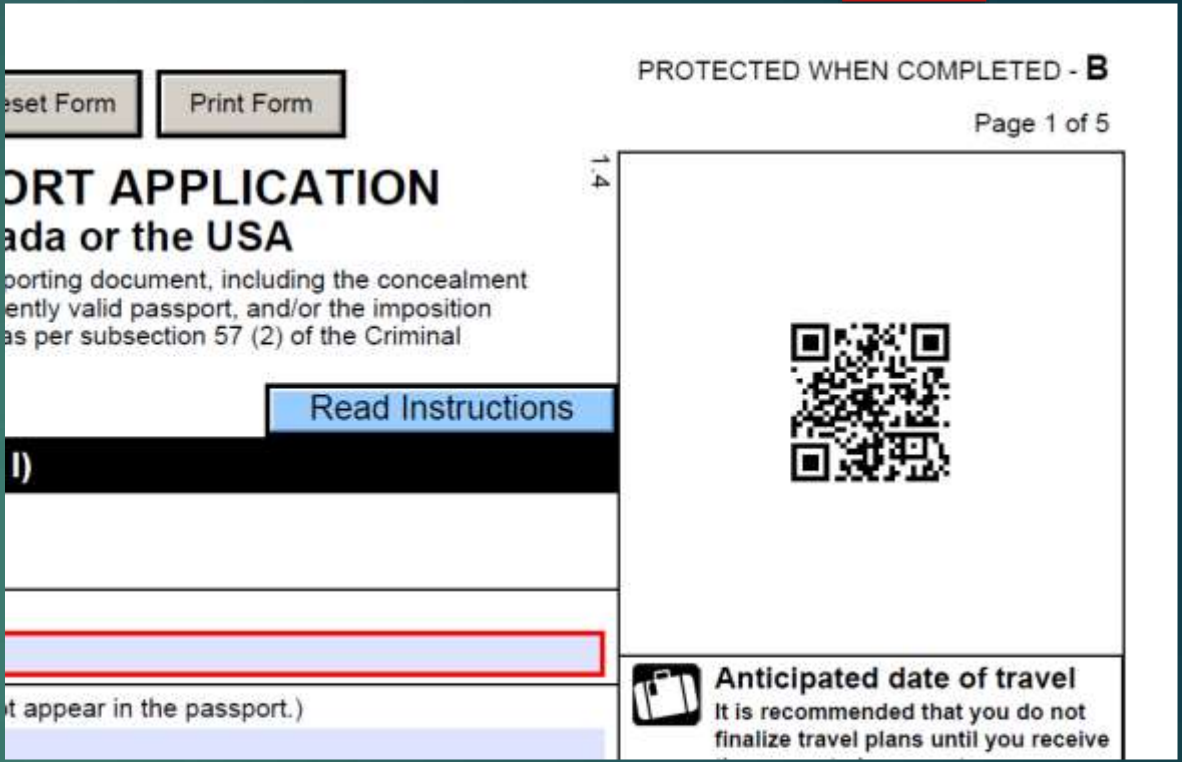
Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Sample Character	ASCII (0-127)	Extended ASCII (0-255) Windows-1252	Notes
a	97, 0x61	97, 0x61	
A	65, 0x41	65, 0X41	Difference of 32, 0x20 between lower and upper case code
b	98, 0x62	98, 0x62	
B	66, 0x42	66, 0x42	
c	99, 0x63	99, 0x63	
C	67, 0x43	67, 0x43	
<	60, 0x3C	60, 0x3C	
=	61, 0x3D	61, 0x3D	
>	62, 0x3E	62, 0x3E	
è		232, 0xE8	Can be entered by pressing the Alt key and using the number keypad to enter decimal value with a leading 0, i.e. è is Alt-0232
é		233, 0xE9	
É		201, 0xC9	

The complete list can be found at https://www.ascii-code.com/

Funny story – reading 2D bar codes

- Getting character coding to work can be hard!
- I was the Director of the Project Management Bureau in Passport Canada and one of the project I was responsible for was to add a 2D bar code to Passport Canada forms.
- We needed to buy about 1,300 2D scan guns.
- The contract included testing of the proposed 2D scan gun.
- The proposed gun did not read the code correctly. A technician showed up with a 3 inches binder of 2D bar code to configure the gun. He started to configure and test…
- He used all the configurations in the binder and none worked. We move to the next bidder.



PROTECTED WHEN COMPLETED - B

Page 1 of 5

ORT APPLICATION

ada or the USA

porting document, including the concealment
ently valid passport, and/or the imposition
as per subsection 57 (2) of the Criminal

t appear in the passport.)

Anticipated date of travel

It is recommended that you do not
finalize travel plans until you receive

How does a computer store and process numbers?

- There are various types of number each with a different way to store and process.
- For human being:
  - Stored using character coding, such as ASCII, composing the number.
  - Very inefficient but easy for human being to read the number.
  - To process number, need to be converted to a binary number, more on that in the next slides.
  - Depending on the character coding used, each digits take 8 to 16 bits.
  - Example: the number 19,534 encoded in ASCII would be using 6 bytes of memory (48 bits):
    - 0x31, 0x39, 0x2C, 0x35, 0x33, 0x34

Binary Coded Decimal (BCD)

- In computing and electronic systems, Binary Coded Decimal (BCD) is a class of binary encodings of decimal numbers where each digit is represented by a fixed number of bits, usually four or eight. Sometimes, special bit patterns are used for a sign or other indications (e.g. error or overflow).
- There are many BCD coding table, the most common is shown on the right. Its advantage is the easy conversion to ASCII by adding 0x30, but still need to be converted to binary to be processed.

Decimal Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Integer number

- An integer is a whole number that does not include decimal parts or fraction.
- The number can be positive, negative, or zero.
- Examples of integers are: -5, 1, 5, 8, 97, and 3,043.
- Examples of numbers that are not integers are: -1.43, 1 3/4, and 3.14.
- For signed number (i.e. negative), a bit is selected to indicate a negative number and special coding is used so that adding -X with X gives 0 without any other processing.

Number of bits	Range
8	0 to 255
8 signed	-128 to 127
16	0 to 65,535
16 signed	-32,768 to 32,767
32	0 to 4,294,967,295
32 signed	-2,147,483,648 to 2,147,483,647
64	0 to 18,446,744,073,709,551,615
64 signed	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating Point Number

- If you want to store and process a number that does not fit in the integer range OR has a decimal parts or fraction, you need to use a floating point data type.
- The floating point, also known as the scientific notation, has the following form mantissa * base$^{\text{exponent}}$.
- Example the mass of the proton can be expressed as $1.67262192369 \times 10^{-27}$ kg.
- For a computer, the base 2 is used, an overall number of bits is selected, then bits are reserved for:
  - sign and value of the mantissa. The first bit has always a value 1 and it is not stored.
  - sign and value of the of the exponent.
  - Some exponent and mantissa value have special meaning such as infinity, Not a Number, overflow, etc.

Standard Floating Point representations

# bits	Name	Range
32	Single or float	-3.4028237 E+38 to -1.175494 E-38 0 1.175494 E-38 to 3.4028237 E+38 up to 7 significant digits
64	Double	-1.797693134862315 E+308 to -2.225073858507201 E-308 0 2.225073858507201 E-308 to 1.797693134862315 E+308 With up to 15 significant digits
80	Extended	-1.18 E+4932 to -3.65 E−4951 0 3.65 E−4951 to 1.18 E+4932 With up to 21 significant digits

I did a Radar Signal Processor project that used single data type, but the answers were wrong. We had to change all the variables to double precision which gave the correct answers but double the memory needs.

Next, how can a bug in a CPU be problematic...

1994 Pentium Floating Point Division bug

- The Pentium FDIV bug is a hardware bug affecting the floating-point unit (FPU) of the early Intel Pentium processors.
- In order to improve the speed of floating-point division calculations on the Pentium chip, Intel opted to replace the shift-and-subtract division algorithm with the Sweeney, Robertson, and Tocher (SRT) algorithm.
- Missing values in a lookup table used by the FPU's floating-point division algorithm led to calculations acquiring small errors.
- Because of the bug, the processor would return incorrect binary floating point results when dividing certain pairs of numbers. Intel replaced the chips for free.

The correct value of the calculation is:

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.333820449136241002$$

The bug in the FPU resulted in the wrong answer in the fifth significant figure!

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.333739068902037589$$

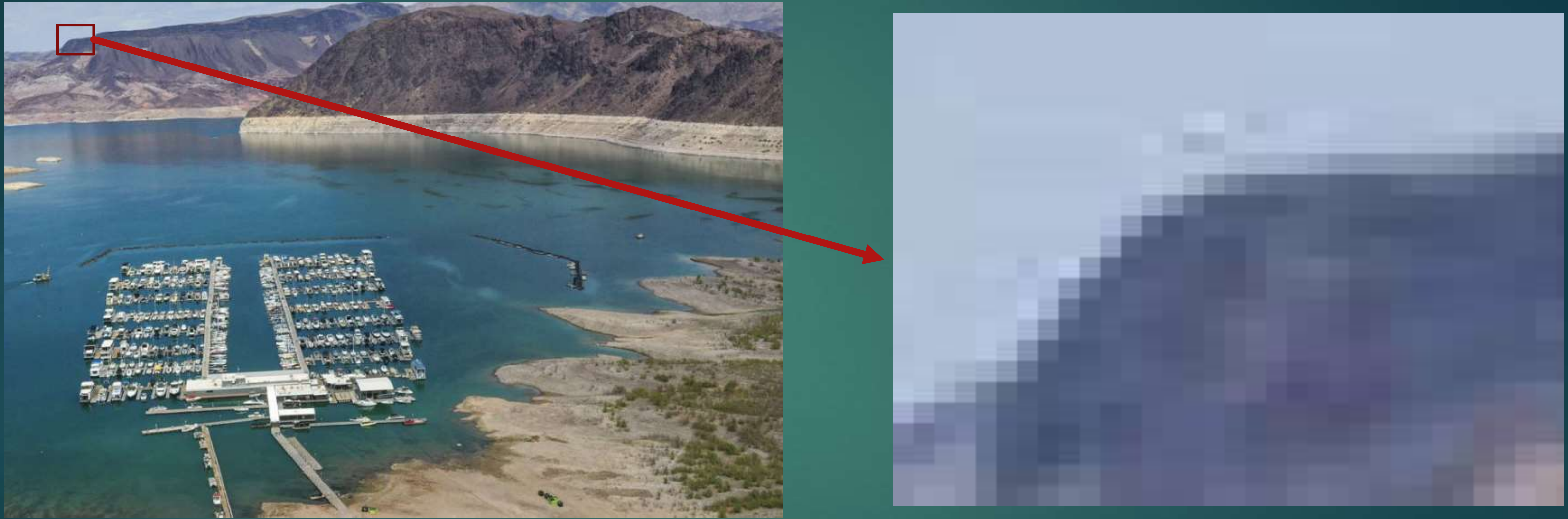
How does a computer store image?

- There are two main ways:
  - Bitmap
  - Vector
- They both have advantages and disadvantages.
- But both end-up as string bits to be displayed.

Bitmap images

- Bitmap images are broken down into tiny elements called pixels (short for “picture element”).
- Each pixel has a color coded using a color code such as Red Green Blue (RGB).
- The number of pixels in an image is called the image resolution.
- Image resolution = Width * High.
- Image size = resolution * number of bits per color.
- Bitmap image can be very large (e.g. 5184 X 3456 X 24 bits for the color = 53 GB!), but can be compressed, with or without loss depending on the algorithm, to reduce size.

Bitmap example zoomed 800%

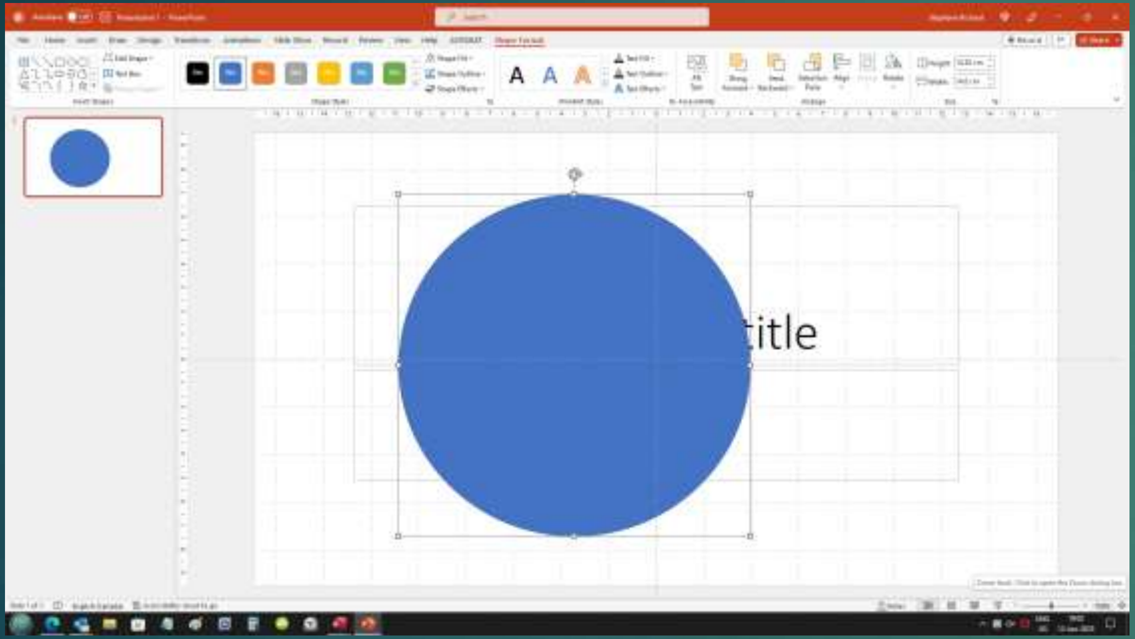


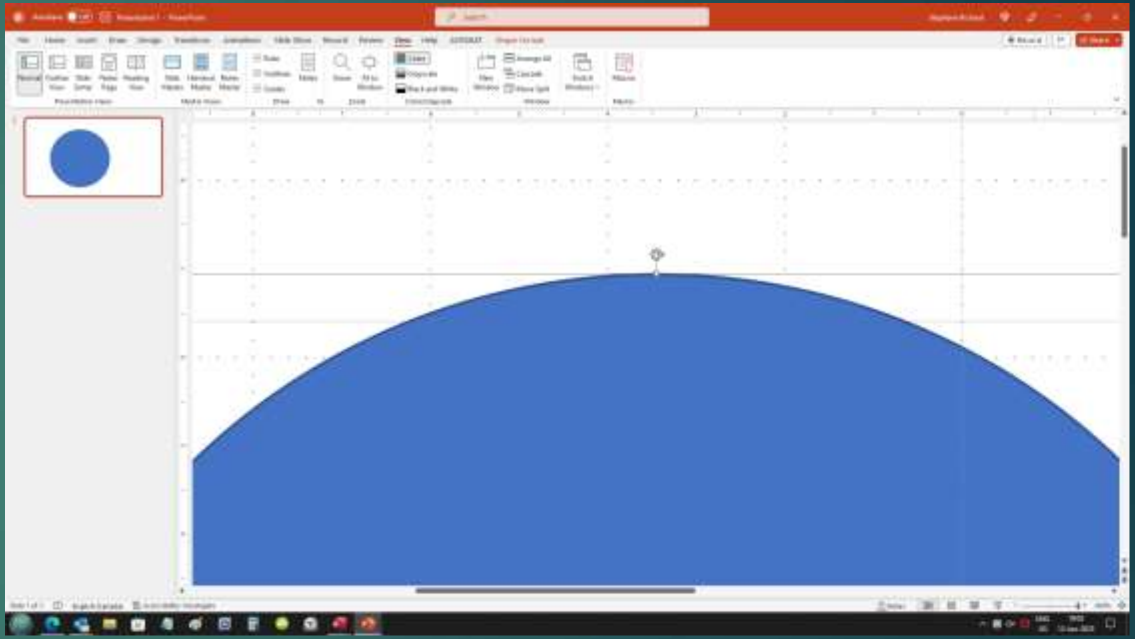
860 X 568 = 488,480 pixels
File save as BMP format (no compression) 679 KB
File save as JPG format (compression with loss) 83.8 KB

Vector images

- A vector image is made out of shapes, straight lines, and curves.
- In vector images, coordinates and geometry define different parts of the image.
- Vector images can be scaled, moved, rotated, filled, etc. as it is redrawn from the shapes.
- Vector are more efficient than bitmaps at storing section with the same color because they do not need to store every pixel, they just store the object type, coordinates, and color.
- To be displayed or printed, vector image must be rendered into bitmap.
- Most 3D game are polygon based, and much of the Graphic Processing Unit pipeline is dedicated to rendering polygons.

Vector image example – from PowerPoint – zoom 400%





So, we have seen how computer store all kind of information using string of bits. Now, how does a computer process string of bits?

Computers process string of bits

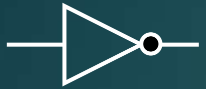
- By reading and writing string of bits from and to memory, network card, secondary storage such as hard disk drive, solid state drive, optical drive, tape drive, etc.
- By moving or copying string of bits between the computer main memory and registers in the Central Processing Unit (CPU).
- By setting the string of bits to a specific value.
- By testing the string of bits and deciding what to do next.
- By manipulating the string of bits with arithmetic and logical operators (e.g., add, subtract, and, or, shifting, rotating, etc.).
- So, it is all about how many bits and how fast bits can be processed in the CPU, stored in memory or secondary storage, displayed, etc.

Comparing Matter and Computer

- Matter is made of:
  - Molecules which are made of
  - Atoms which are made of
  - Proton(s), Neutron(s), and Electron(s)

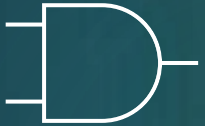
- Computer is made of:
  - Circuits and Boards which are made of
  - Logic gates which are made of
  - Transistors and other electronic components

Elementary Logic Gates



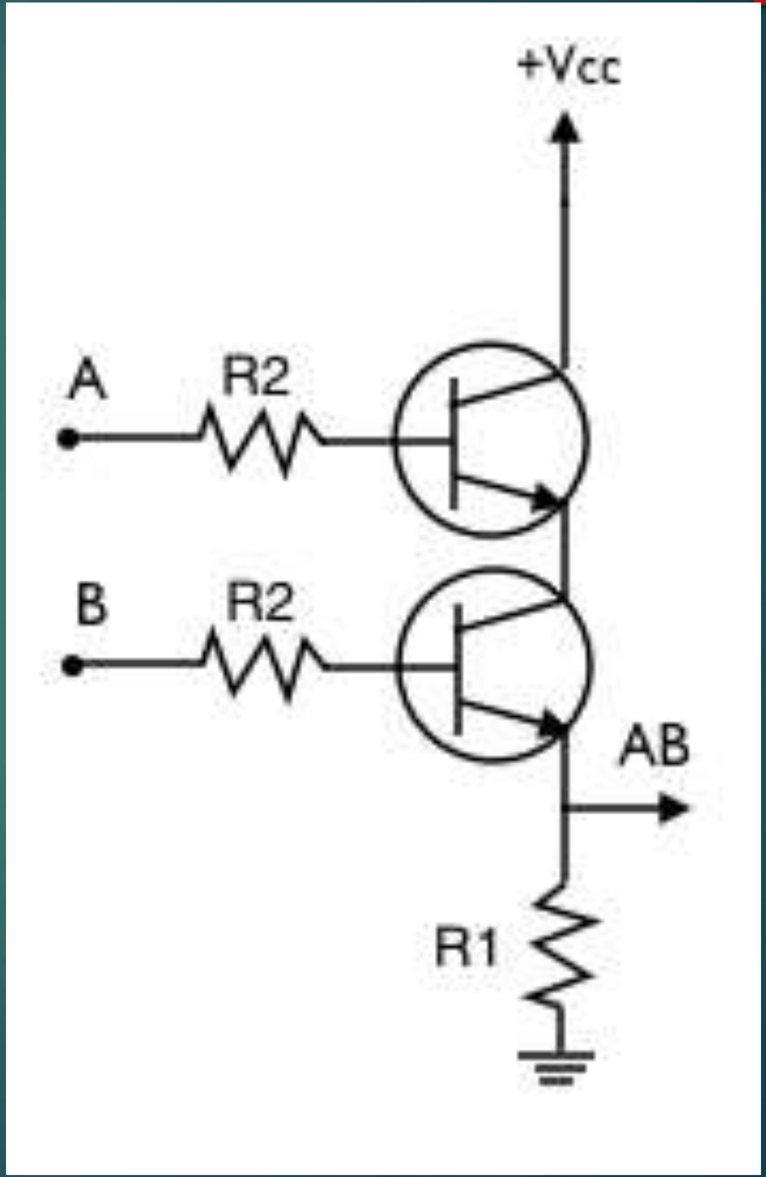
NOT

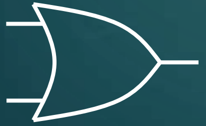
In	Out
0	1
1	0



AND

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

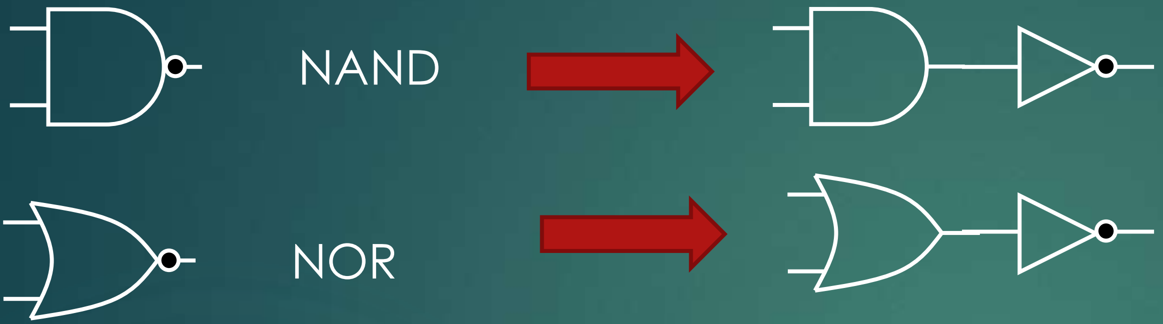


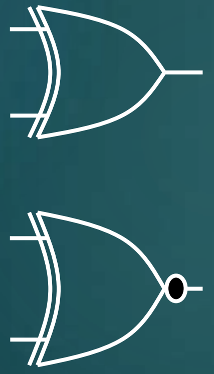


OR

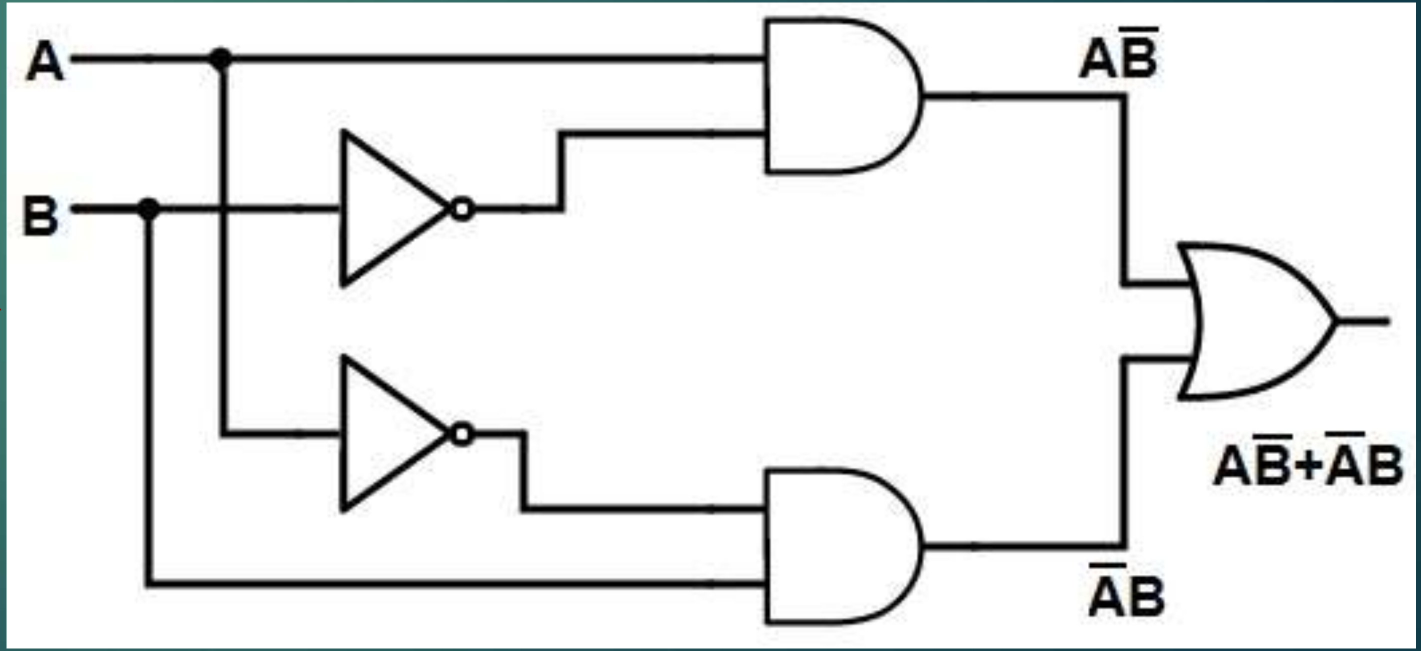
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

Gates built from Elementary Gates





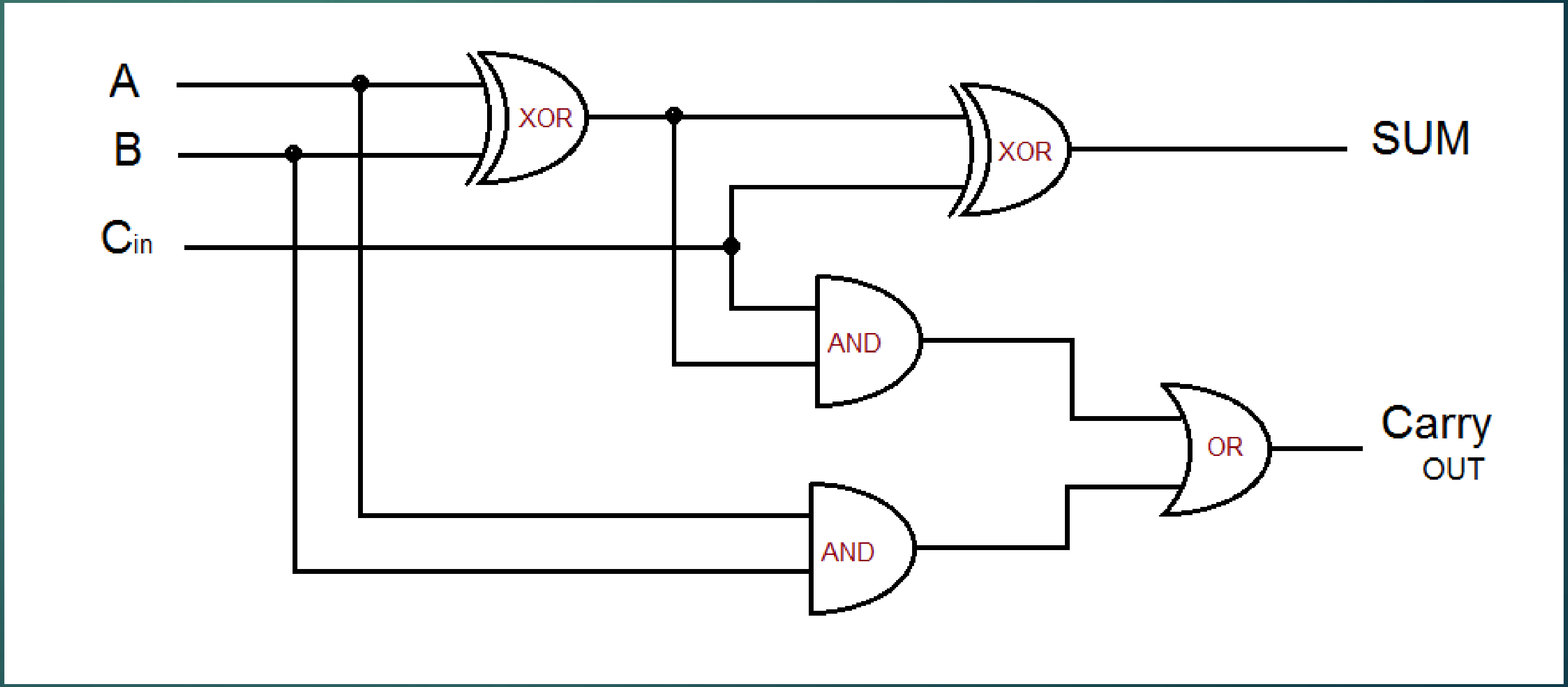
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0



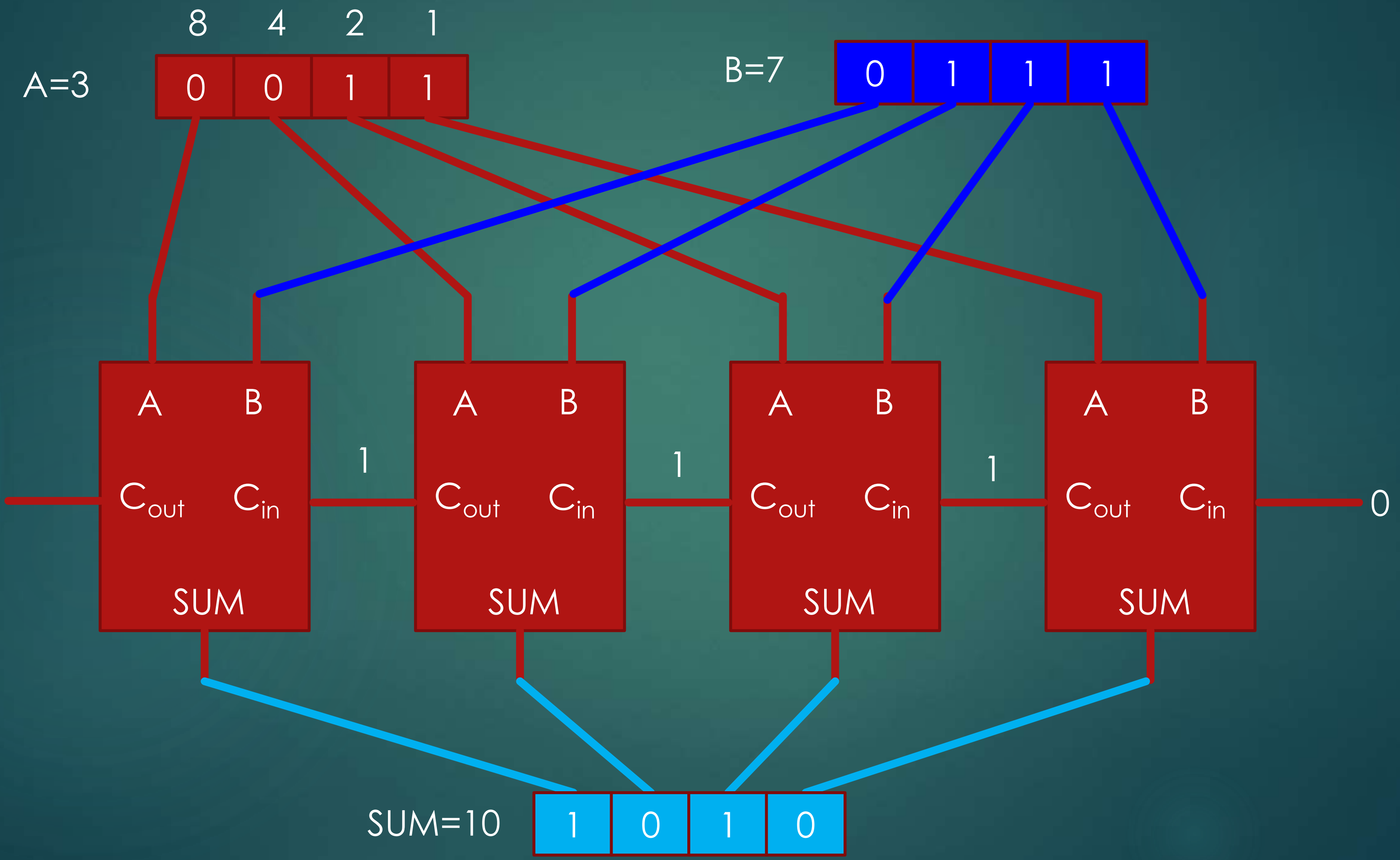
2 bits Full Adder

- Inputs: A, B, Carry-in
- Outputs: Sum of A and B, Carry-on

A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



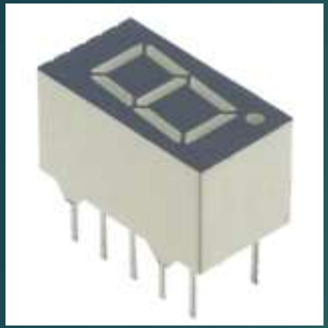
Example adding 3 and 7

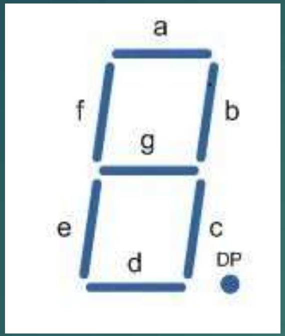


Wonder what the last C_{out} is used for? It is used to indicate an overflow. With 4 bits, the maximum number is 15.

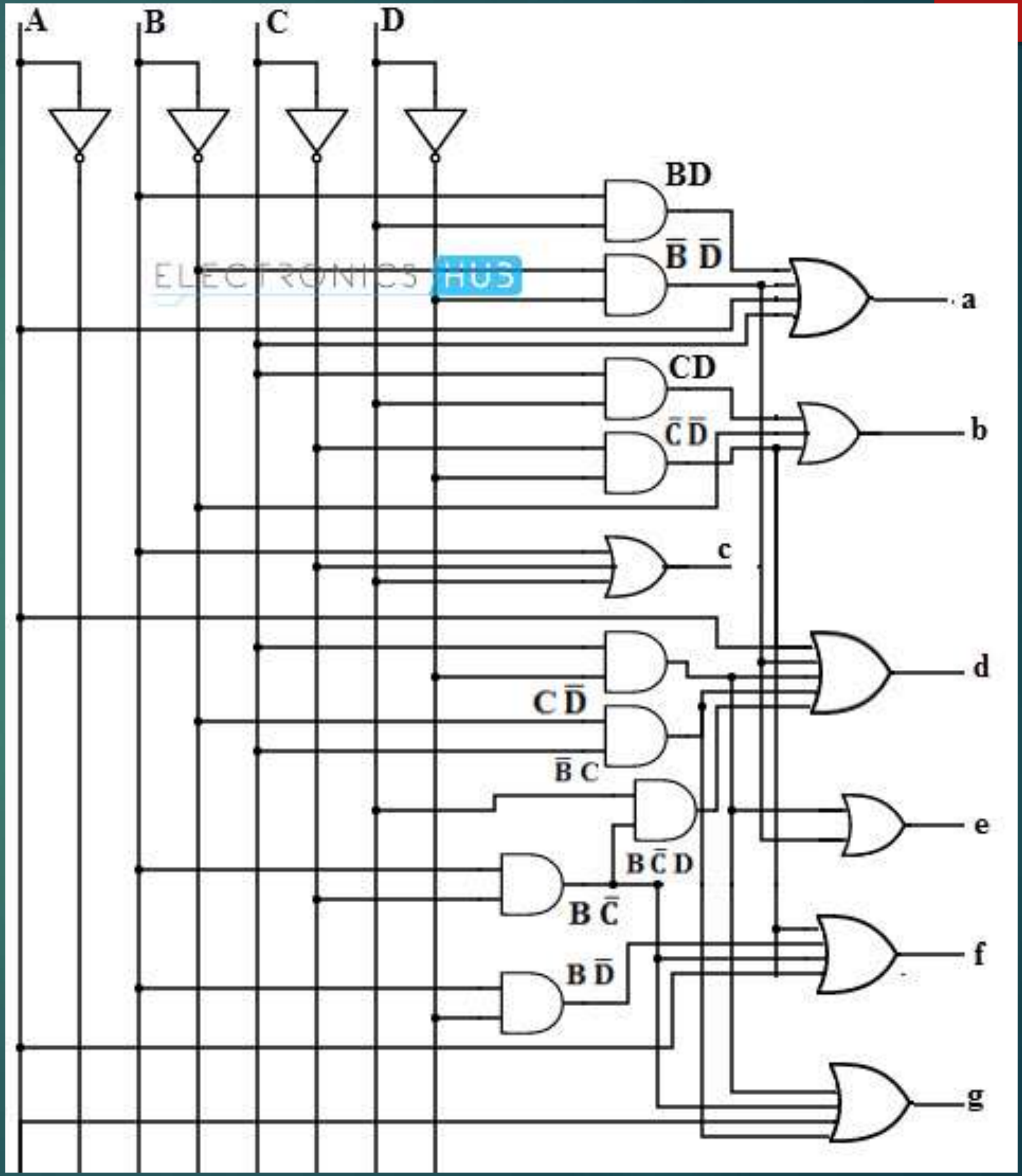
While this design works, we need to wait for the carry to propagate through the circuit. A Carry Look Ahead circuit is used to resolved this problem.

Example of more complex circuits



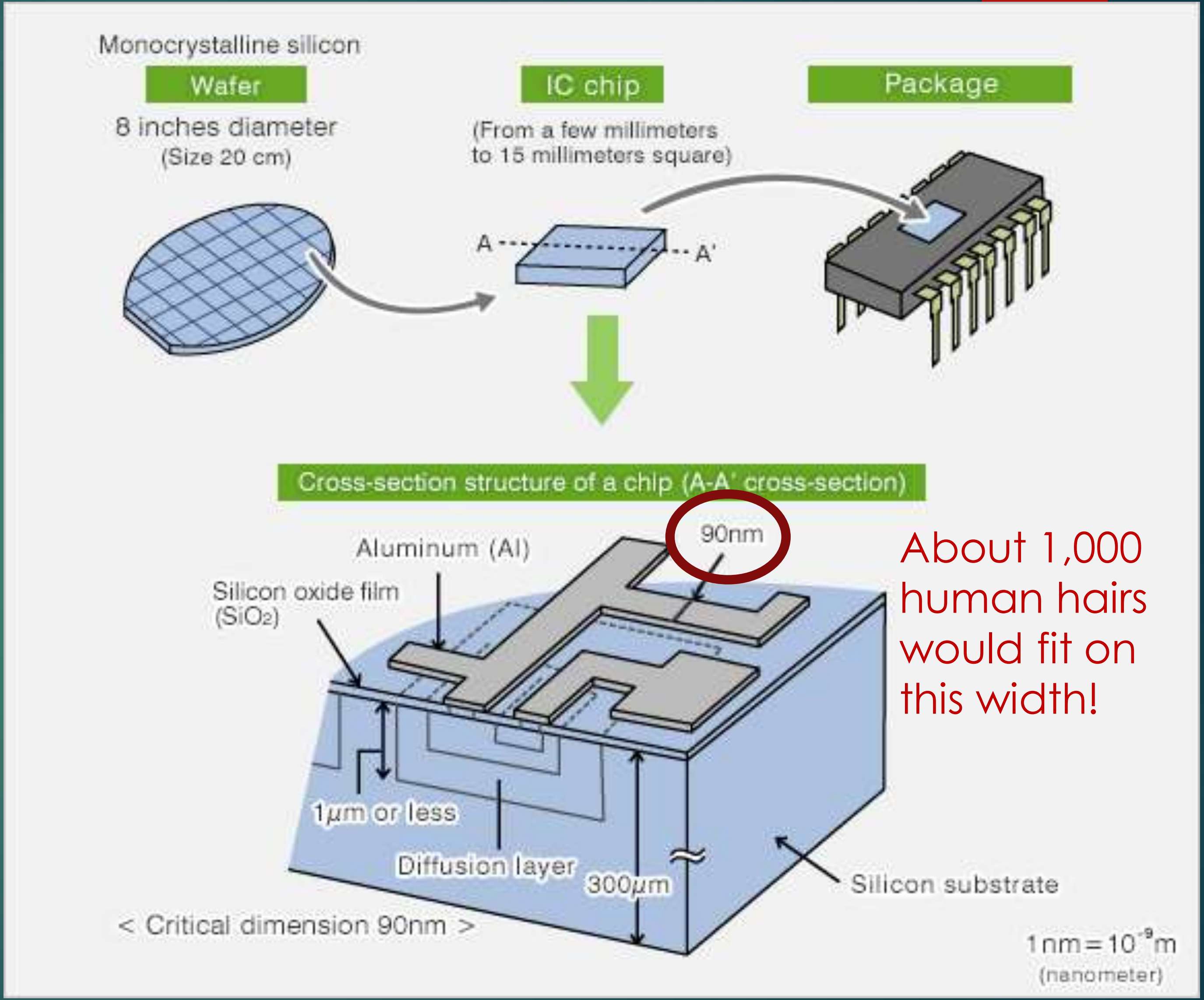


A	B	C	D	Display	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1



How do you fit millions of transistors and electronic components in about 1 inch square?

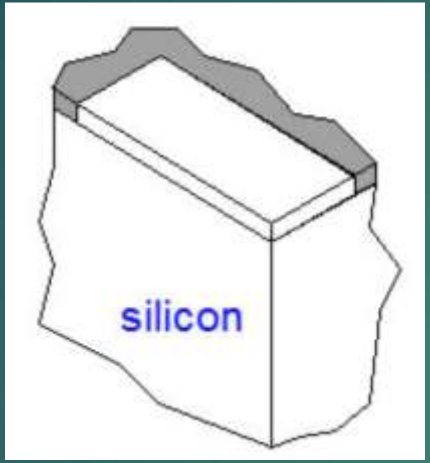
- Miniaturization!

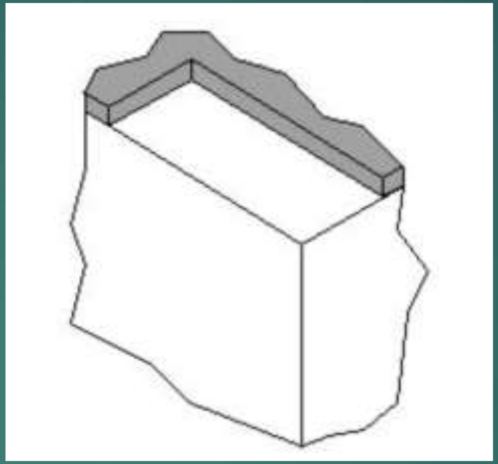
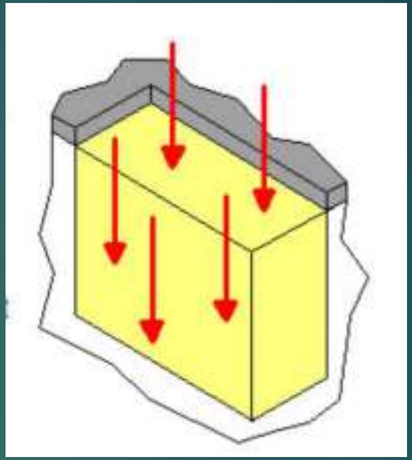
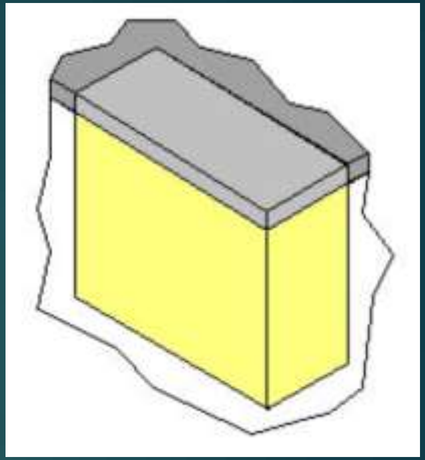
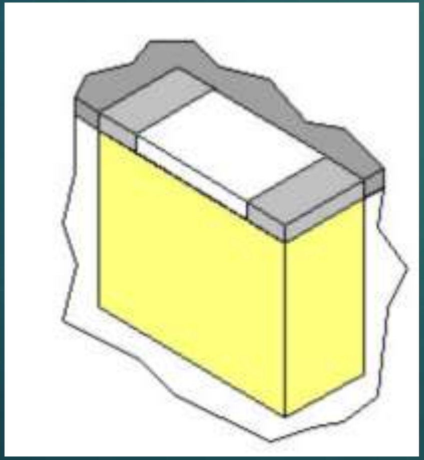
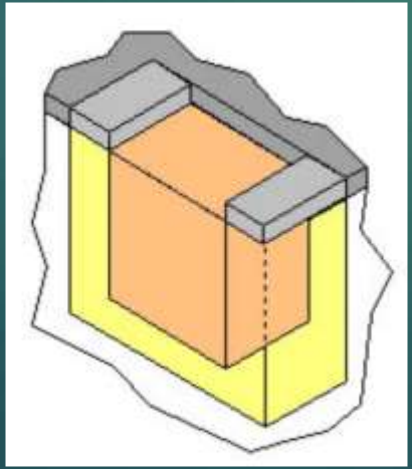
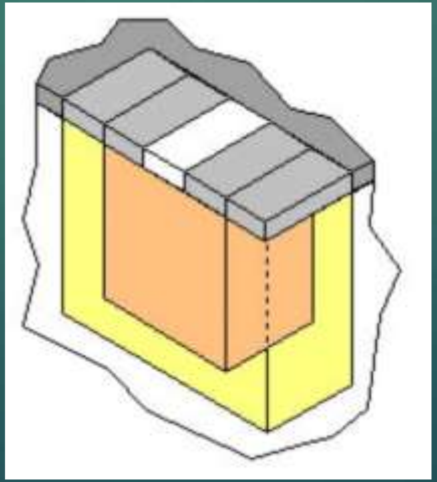
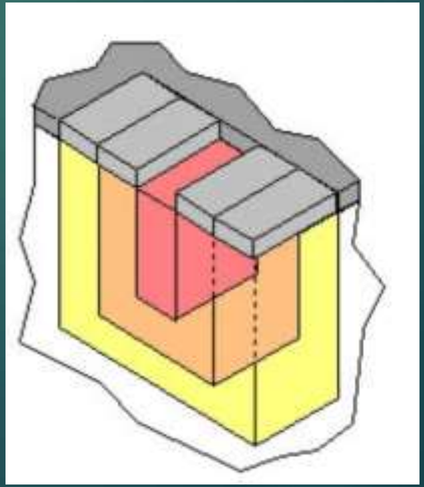
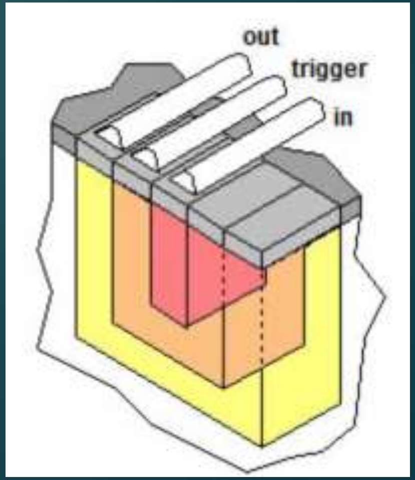


About 1,000 human hairs would fit on this width!

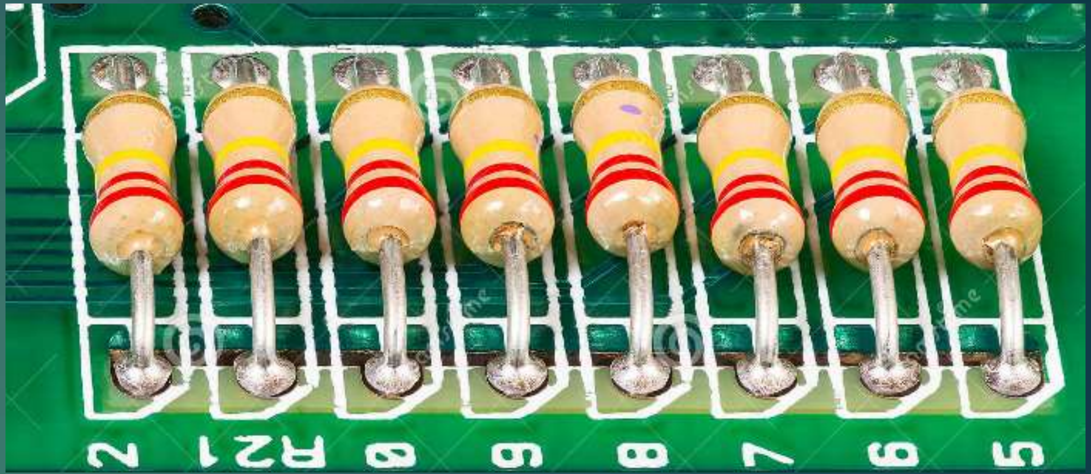
How is a semiconductor wafer built?

Through multiple stages of masking, etching, and diffusion, the sublayers on the chip are created. The final stage lays the top metal layer (usually aluminum), which interconnects the transistors to each other and to the outside world.

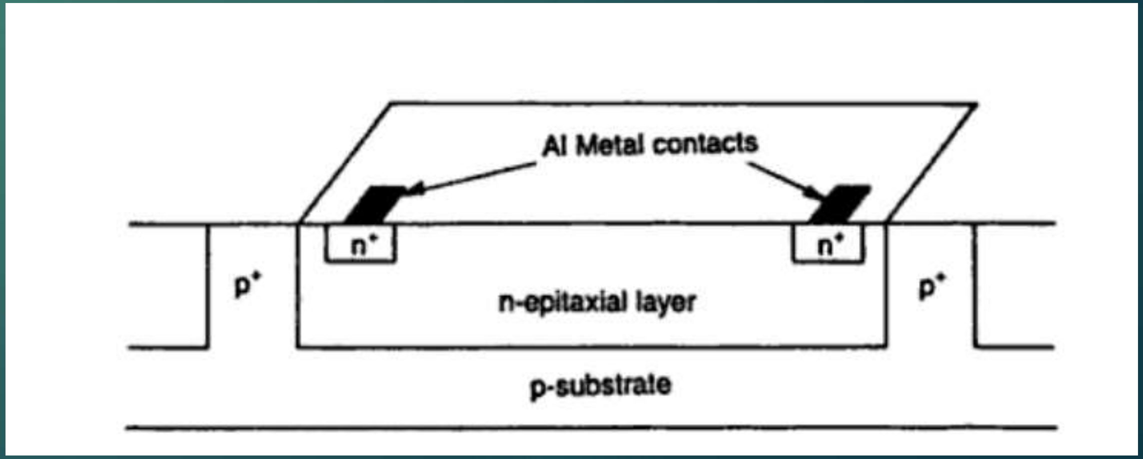


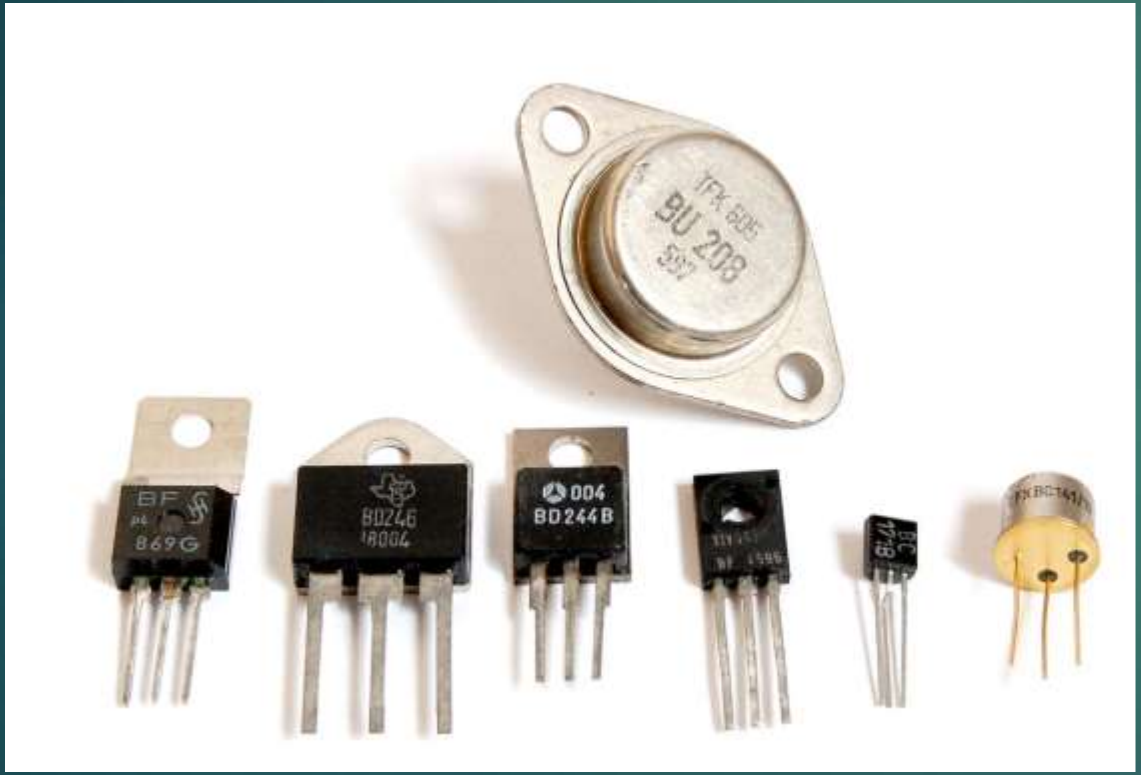
Component Miniaturization - Resistor

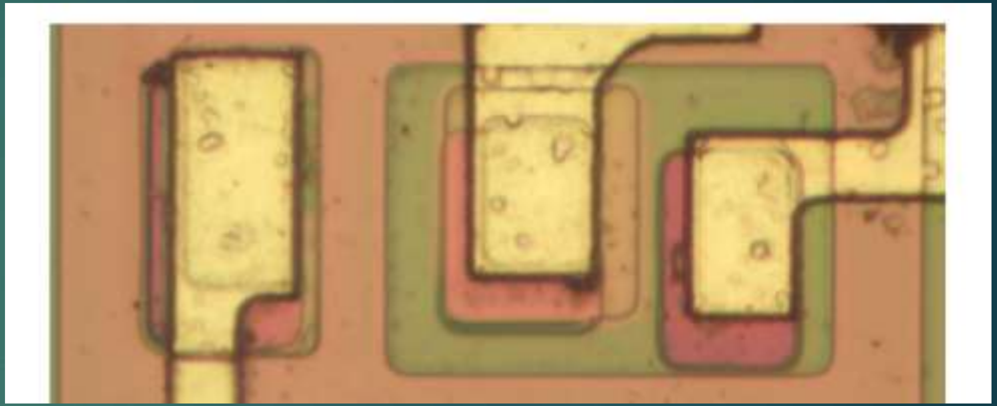


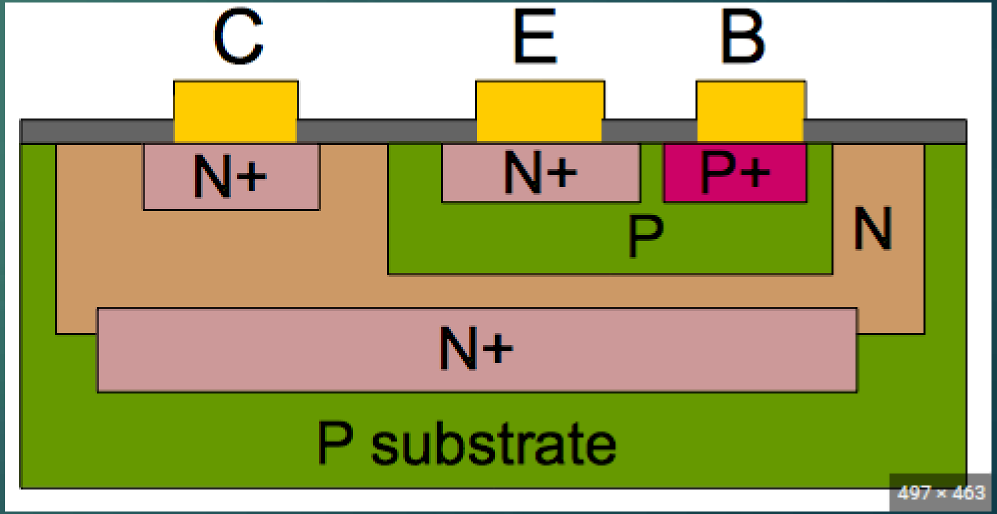
Size and shape will vary depending on the power requirements



Component Miniaturization - Transistor

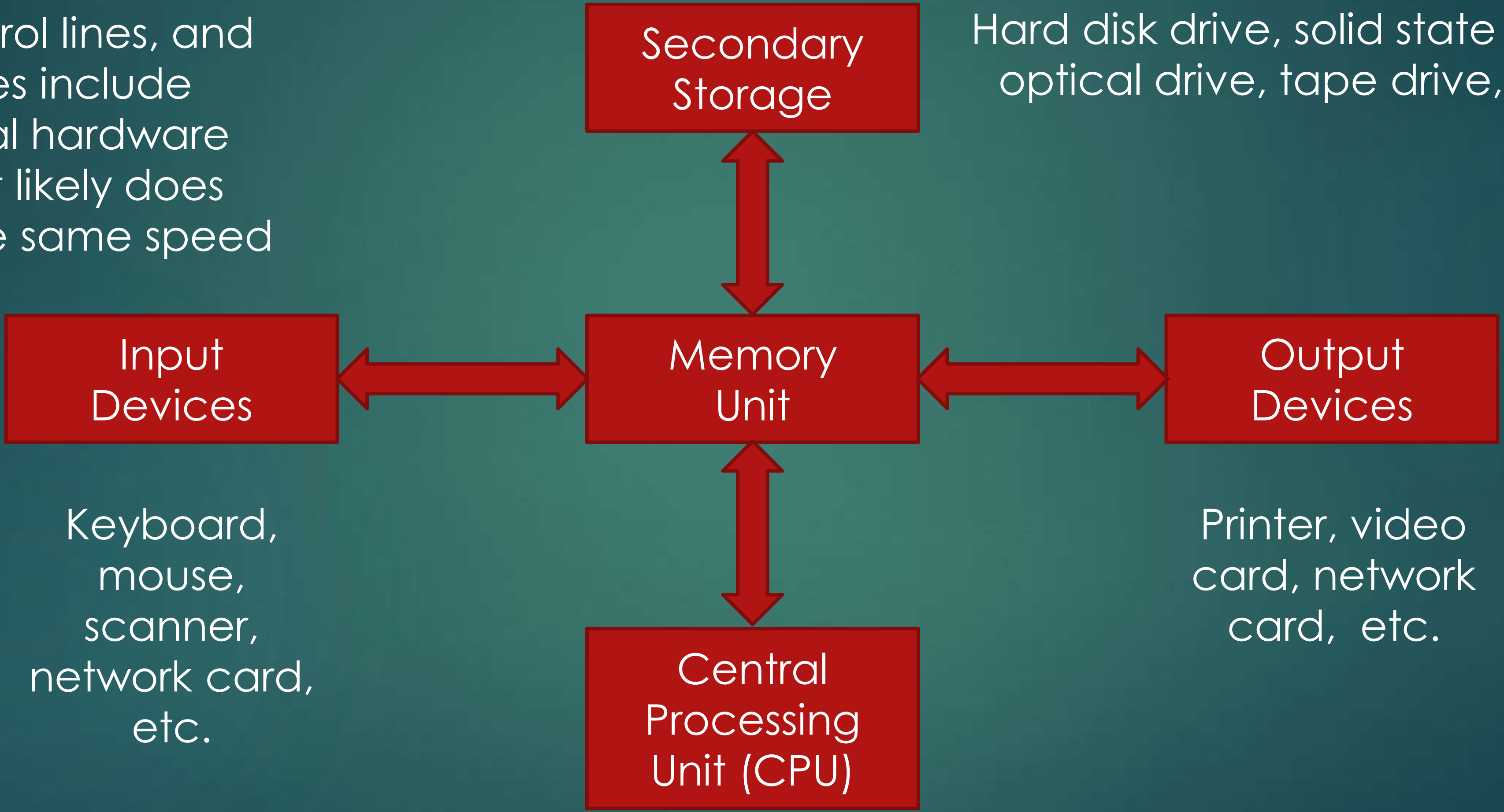






Basic Computer Block Diagram

All the arrows are data and control lines, and sometimes include additional hardware and most likely does run at the same speed



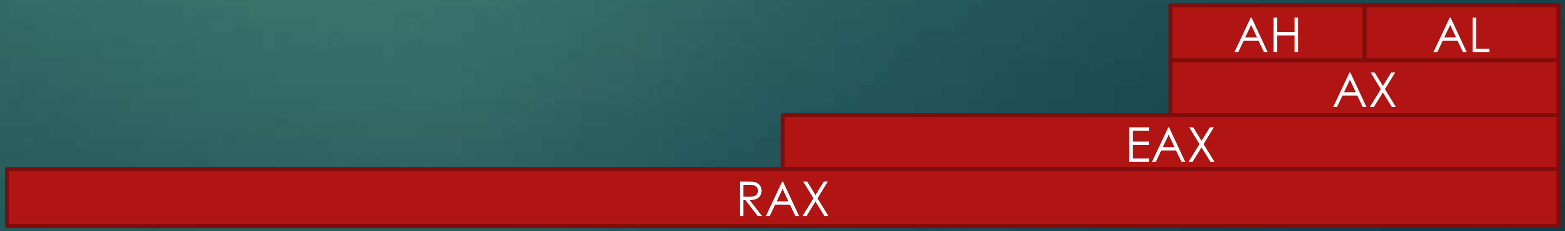
Hard disk drive, solid state drive, optical drive, tape drive, etc.

Keyboard, mouse, scanner, network card, etc.

Printer, video card, network card, etc.

Intel x86 CPU Architecture - General-Purpose Registers (GPR)

- Inside a CPU, register are storage locations and are accessed by instructions to read, modify and store values.
- General Purpose Registers:
  - AX: Accumulator register, used in arithmetic operations.
  - BX: Base register, used as a pointer to data.
  - CX: Counter register, used in shift/rotate instructions and loops.
  - DX: Data register, used in arithmetic operations and I/O operations.
- Names varies depending on how many bits:
  - R-X, 64 bits, for example RAX is a 64 bits accumulator register
  - E-X, 32 bits
  - -X, 16 bits
  - -H and -L, 8 bits



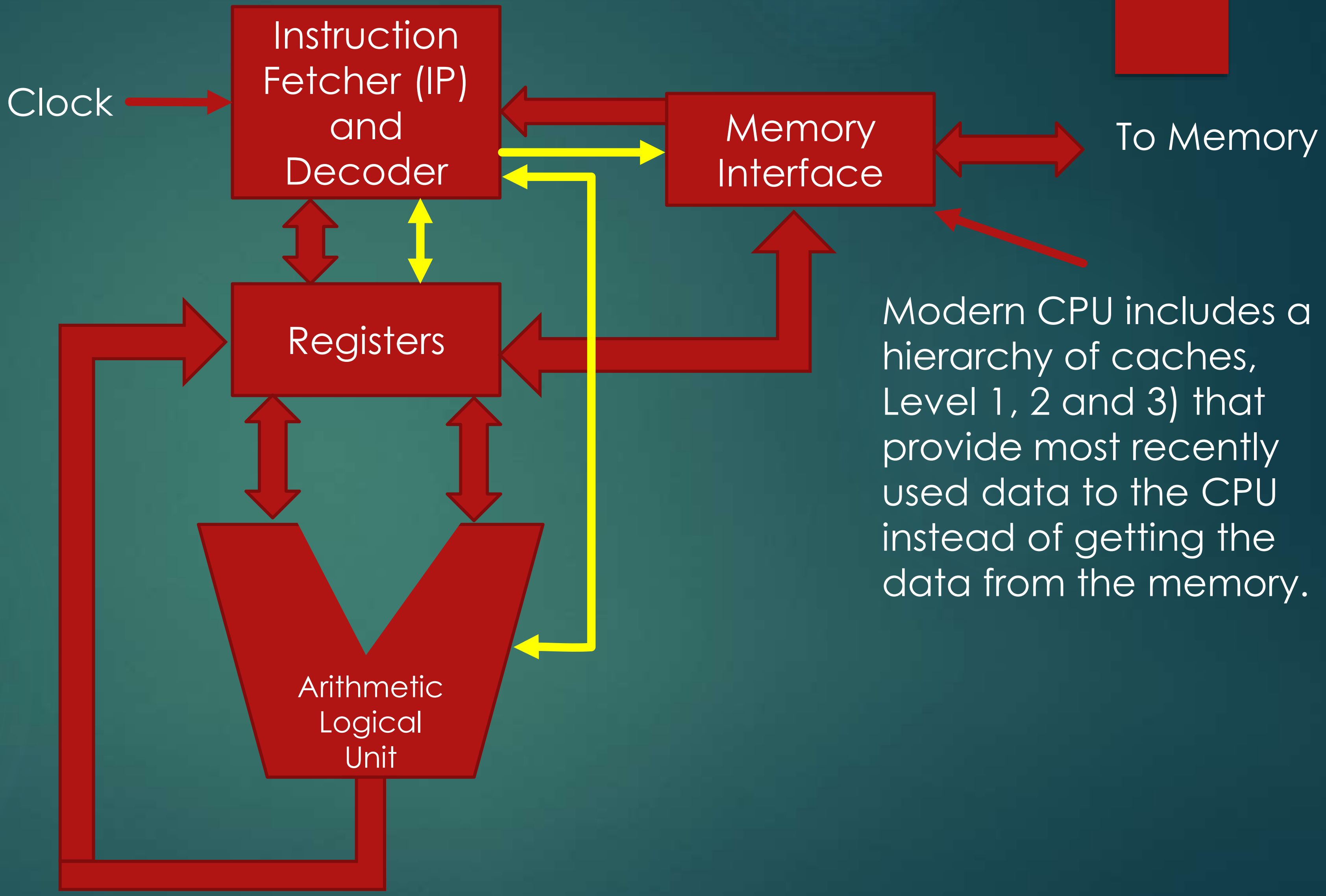
Intel x86 Architecture – Other Registers

- Instruction Pointer (IP): contains the address of the next instruction to be executed if no branching is done.
- EFLAGS: a 32-bit register used as a collection of bits representing Boolean values to store the results of operations and the state of the processor.
- Segment Registers for memory access:
  - Segment Stack Pointer register (SP): Pointer to the top of the stack.
  - Stack Base Pointer register (BP): Used to point to the base of the stack.
  - Source Index register (SI): Used as a pointer to a source in stream operations.
  - Destination Index register (DI): Used as a pointer to a destination in stream operations.

Central Processing Unit Basic Block Diagram

🟥 Data

🟨 Control



Modern CPU includes a hierarchy of caches, Level 1, 2 and 3) that provide most recently used data to the CPU instead of getting the data from the memory.

The opcode is shorter than mov eax, 0, only 2 bytes, and the processor recognizes the special case and treats it as a mov eax, 0 without a false read dependency on eax, so the execution time is the sa

Memory, Machine Code, Assembly Language Example for Intel Pentium

Instruction Pointer (IP)

Memory		Machine Code	Assembly Language
B8	+4	B8 22 11 00 FF	movl $0xFF001122, %eax
22			
11			
00			
FF			
01	+1	01 CA	addl %edx, %ecx
CA			
31	+1	31 F6	xorl %esi, %esi
F6			
53	+0	53	pushl %ebx

High level computer languages such C, C++, Pascal, Visual Basic, etc. generate Assembly Code statements which are translated to Machine Code unique to a CPU type.

This XOR would always result in 0, but op code shorter than doing mov esi, 0

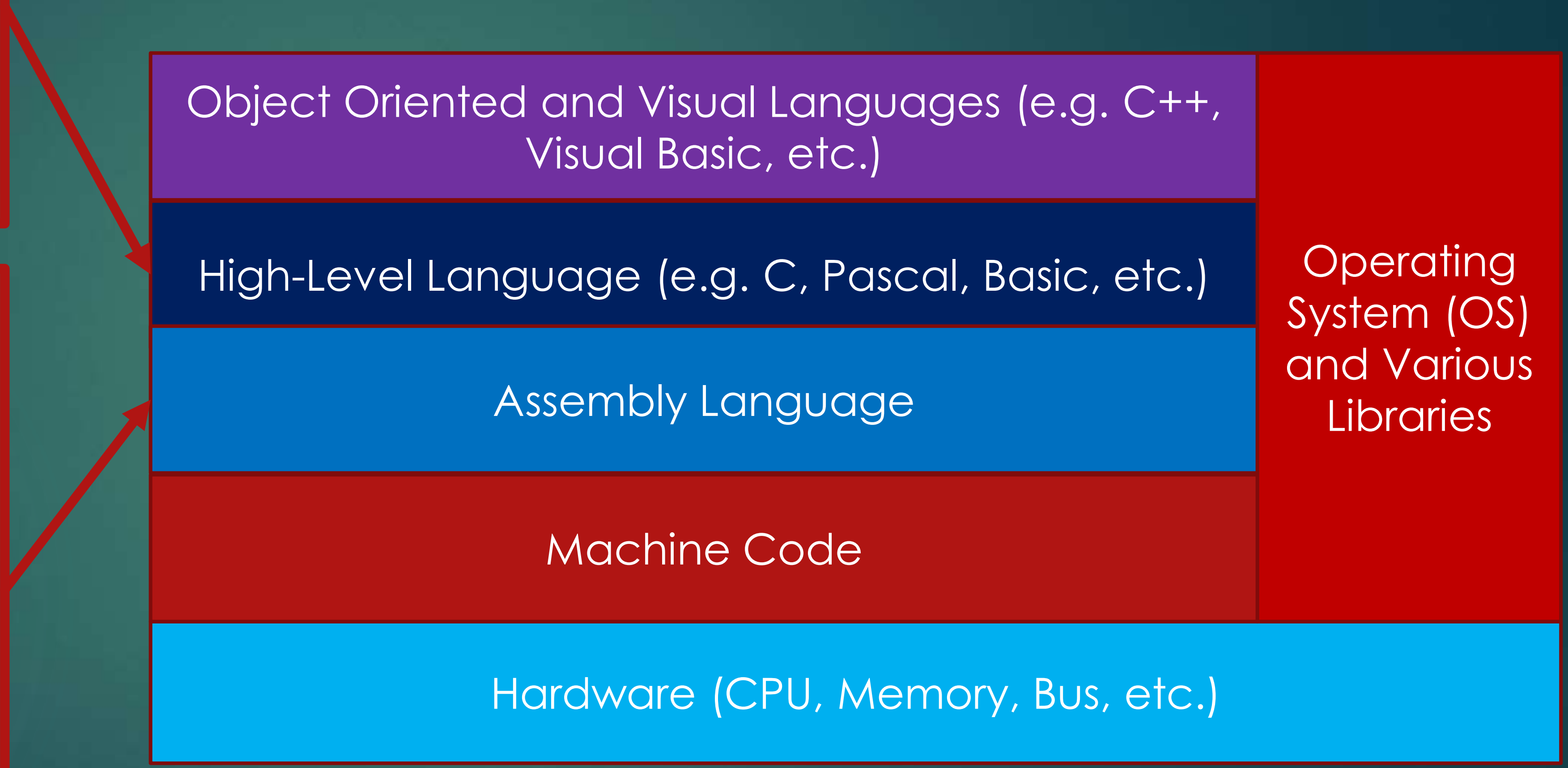
Hardware and Software Stack

C language

```
#include  <stdio.h>
int main()
{
   printf("hello world\n");
   return 0;
}
```

Assembly

```
.LC0: .string "hello world\n"
.globl main
.type main, @function
main:
  pushq %rbp
  movq %rbp, %rsp
  movl %edi, $.LC0
  call printf
  movl %eax, $0
  ret
```


Object Oriented and Visual Languages (e.g. C++, Visual Basic, etc.)
High-Level Language (e.g. C, Pascal, Basic, etc.)
Assembly Language
Machine Code
Operating System (OS) and Various Libraries
Hardware (CPU, Memory, Bus, etc.)

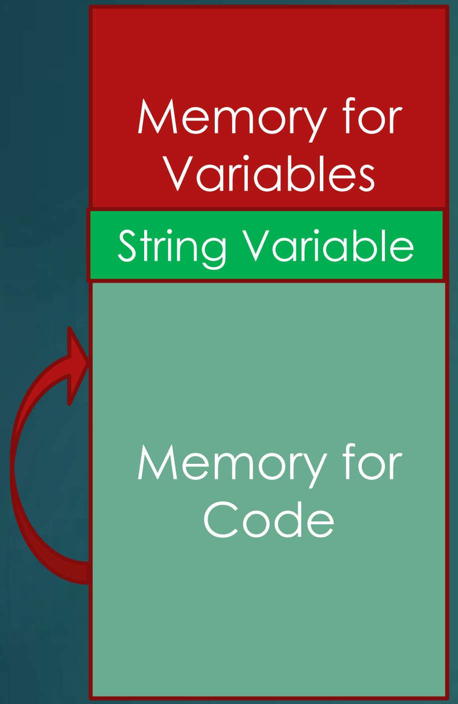
Intel CPU addressing bug

- In 1990, one of my friend was creating software for submarine detection.
- However, its program froze the Intel 80286 CPU.
- He looked at the assembly code generated by the C compiler and started to remove instructions until he was left with one instruction that froze the CPU.
- He called Intel and explained the problem. The Intel representative did not believe him.
- He said – start Debug and enter the following code and run it.
- The Intel representative said – Hum, will call you back...
- Intel never did call him back, but Intel told all the compiler producers to never generate this instruction.
- Problem solved!

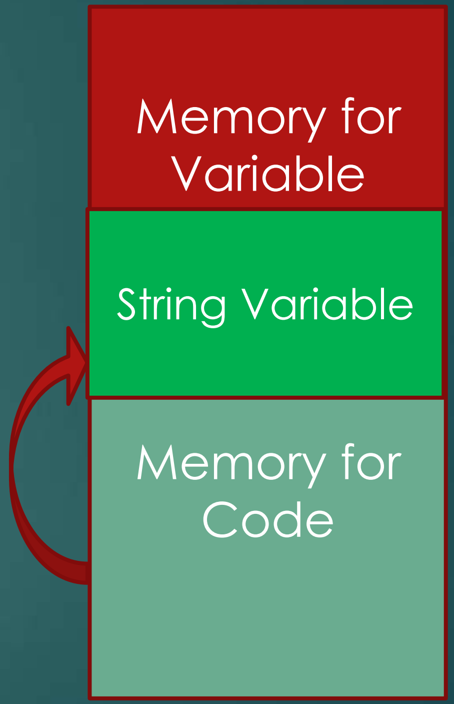
The execution of a program by a CPU can be interrupted

- In digital computers, an interrupt (sometimes referred to as a trap) is a request for the processor to interrupt currently executing code, so that the event can be processed in a timely manner.
- If the request is accepted, the processor will suspend its current activities, save its state (i.e. registers) in memory, and execute a function called an interrupt handler to deal with the event.
- This interruption is often temporary, allowing the software to resume normal activities after the interrupt handler finishes, although the interrupt could instead indicate a fatal error.
- Interrupts are commonly used by hardware devices to indicate electronic or physical state changes that require time-sensitive attention.
- For example moving the mouse or pressing a key on the keyboard generates interrupt and the data is stored in memory controlled by the Operating System.

When I crashed Windows 3 and the Intel 386 CPU



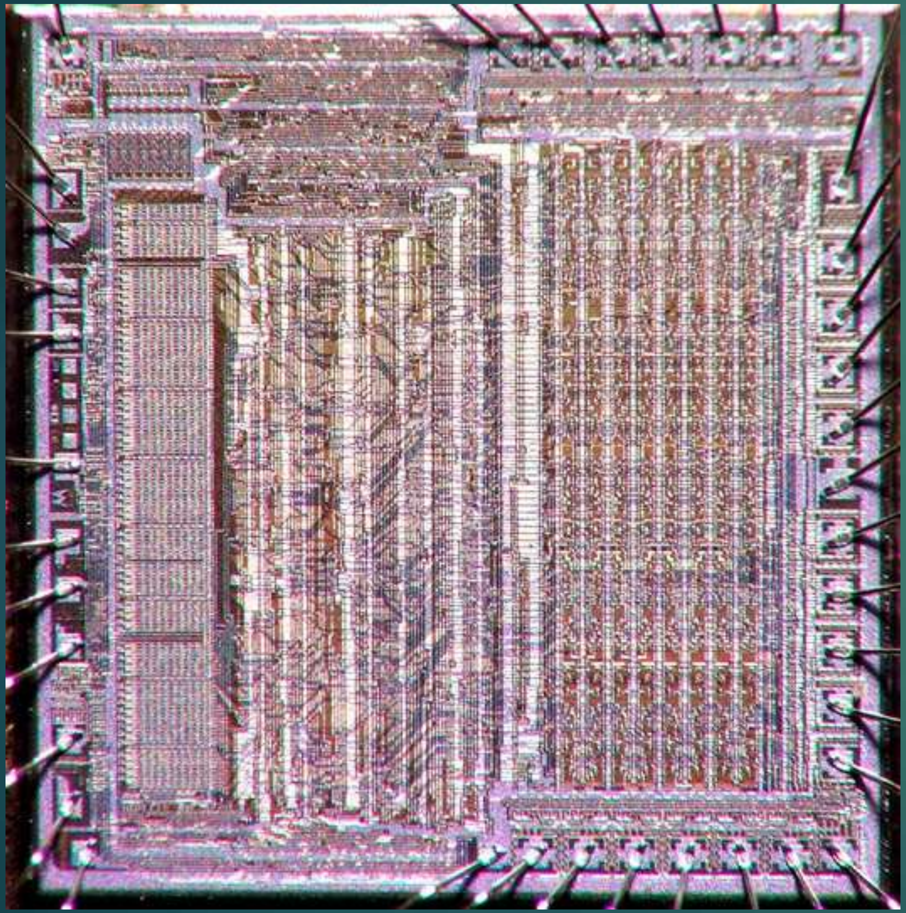
- One of the variable was for a string of characters. However, the programming language I was using, C, does not check if the string of character grows longer then the allocated space.
- This type of error is called a buffer overrun.
- So, program worked OK for short string, however a long string crashed Windows 3 and the Blue Screen of Death said that I tried to execute an instruction that does not exist on the Intel 386 CPU.

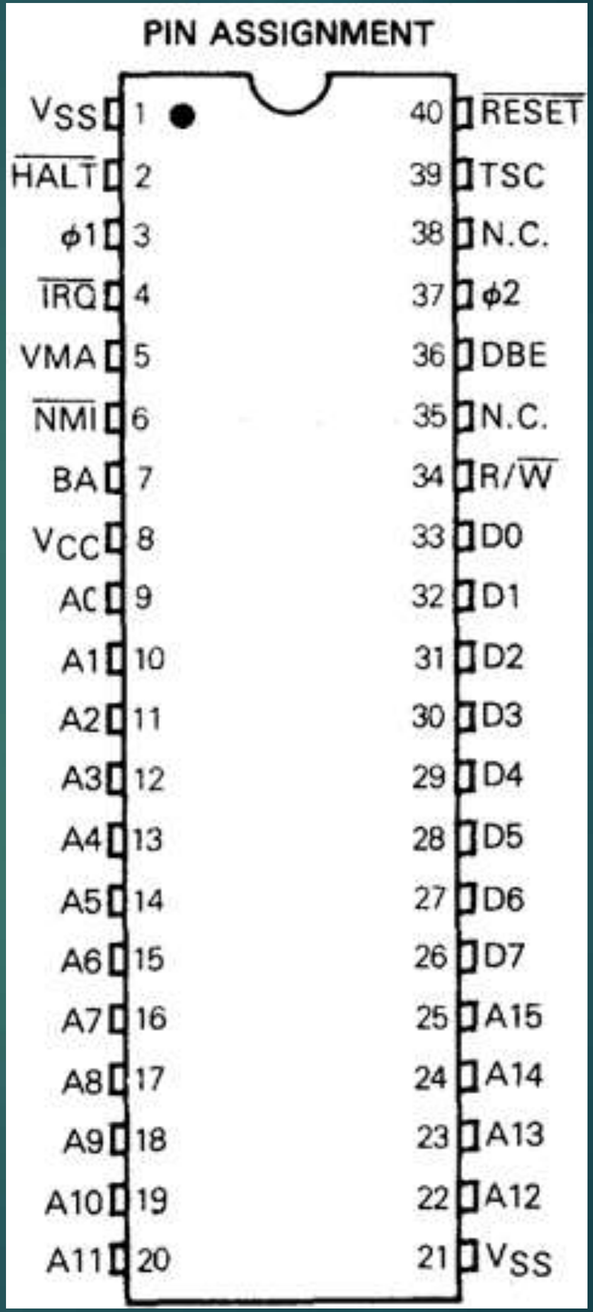


Comparison between 2 Microprocessors released 17 years apart

Motorola 6800	Intel 80486
Released in 1972	Released in 1989
8 bits	32 bits
6 registers (3 8-bit and 3 16-bit)	24 registers (6 16-bit, 10 32-bit, 8 80-bit)
4,100 transistors	1.2M transistors
1 MHz, 40-pin chip	100 MHz, 168-pin chip

Motorola 6800 chip





Interrupt Request

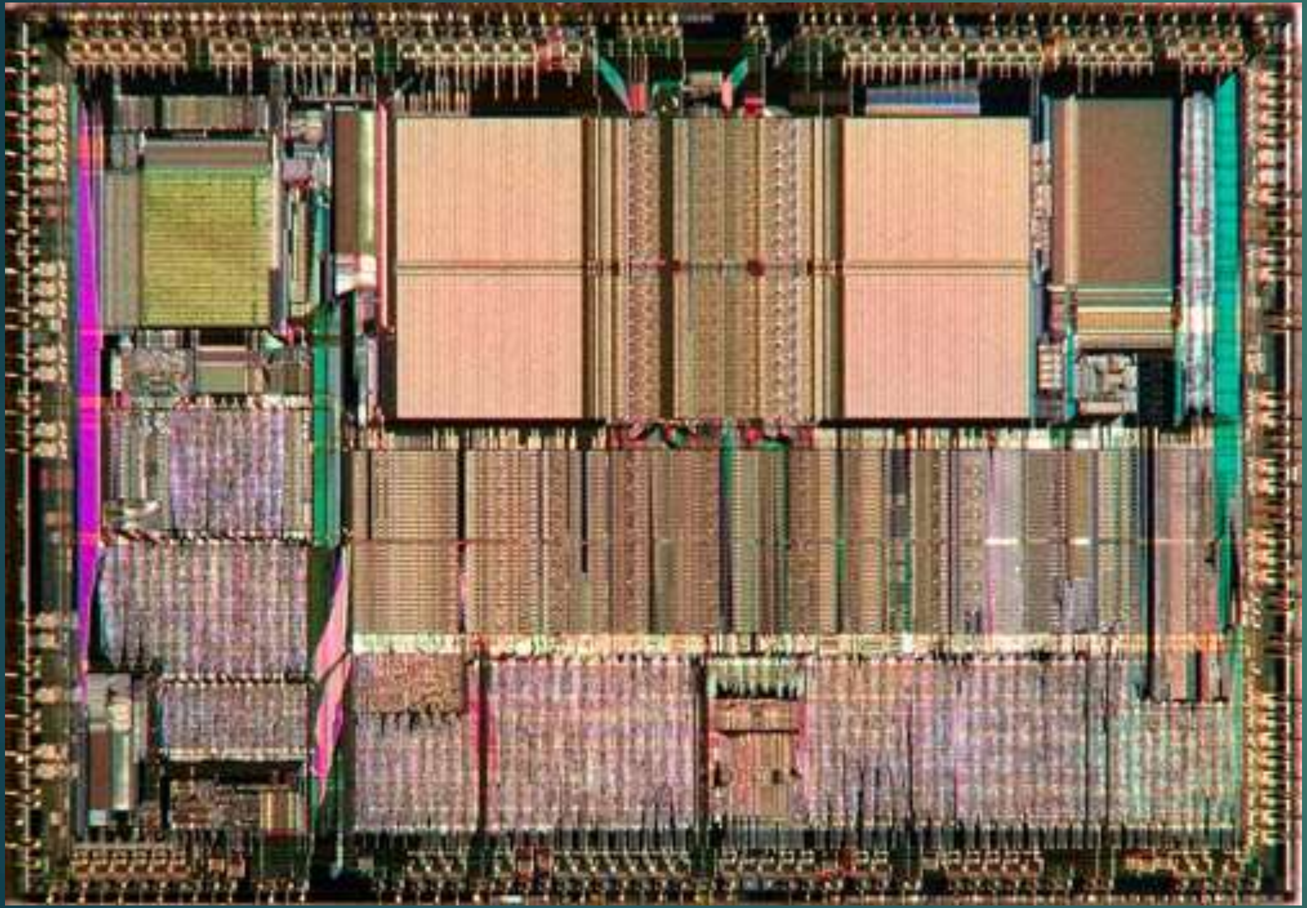
A0 to A15 Memory Address

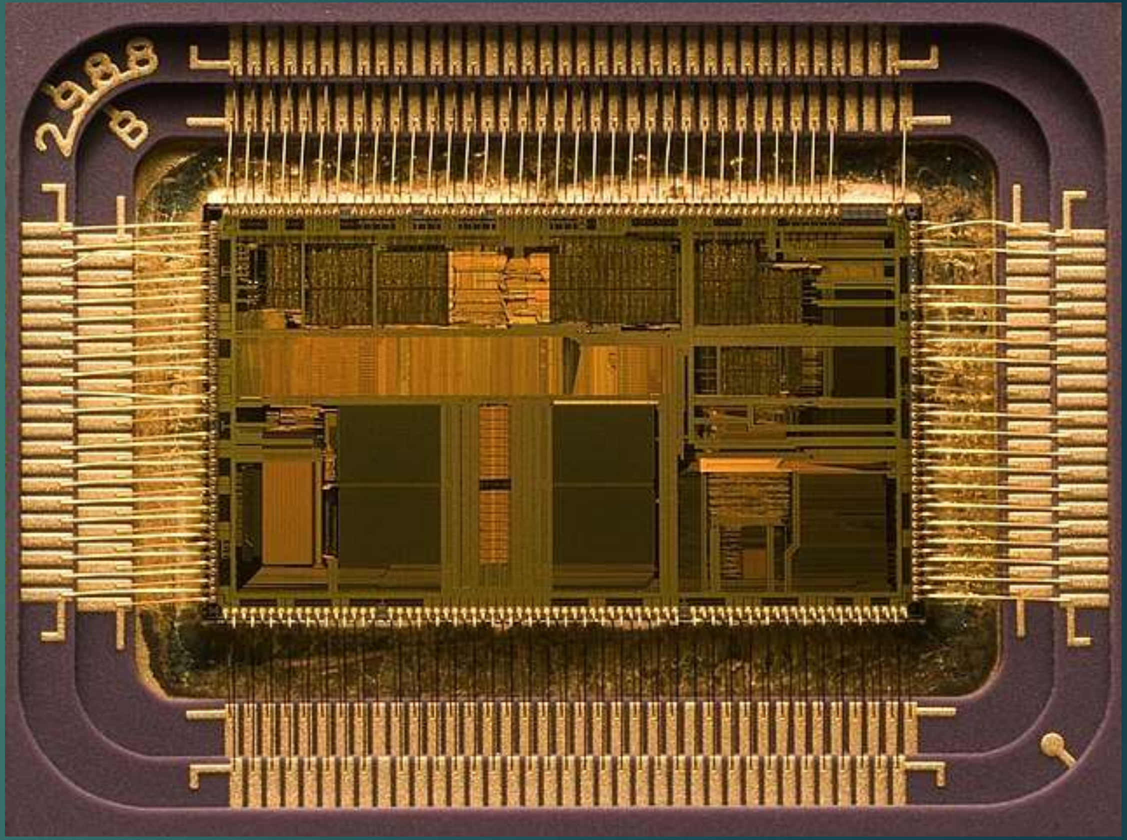
Reset

Memory Read or Write

D0 to D7 Memory Data

Intel 40486 chip





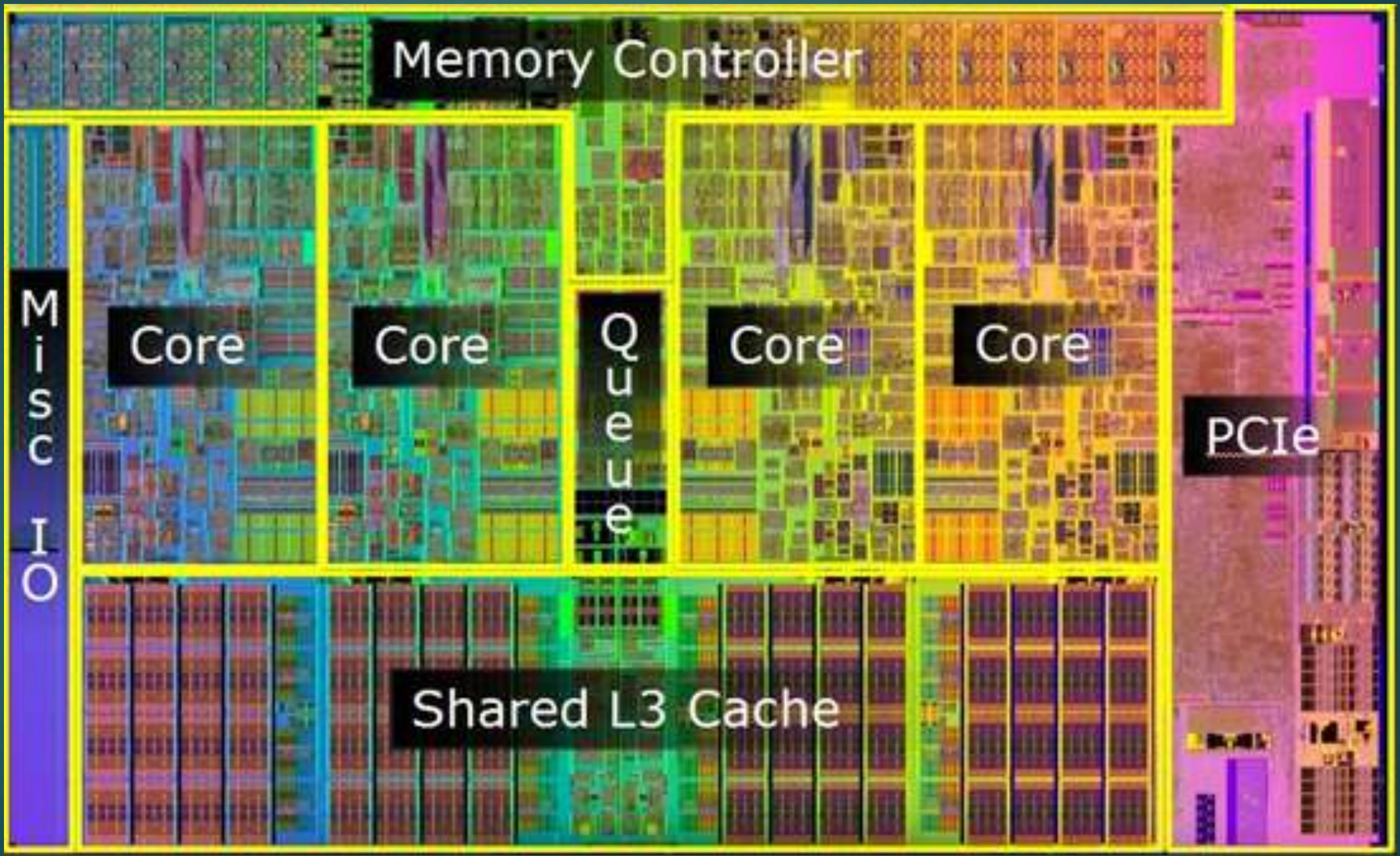
Instructions Grouping Motorola 6800

- Data Transfer Group:
  - Transfer data between registers and memory
- Arithmetic Group:
  - Add and subtract (NO MULTIPLY OR DIVIDE – NEED TO DO WITH CODING)
  - Increment and decrement
- Logical Group:
  - Compare, XOR, OR, AND, NOT, Rotate
- Branch Group:
  - Test and branch
  - Call and return from subroutine
- Miscellaneous Instructions:
  - No Operation, Wait for Interrupt

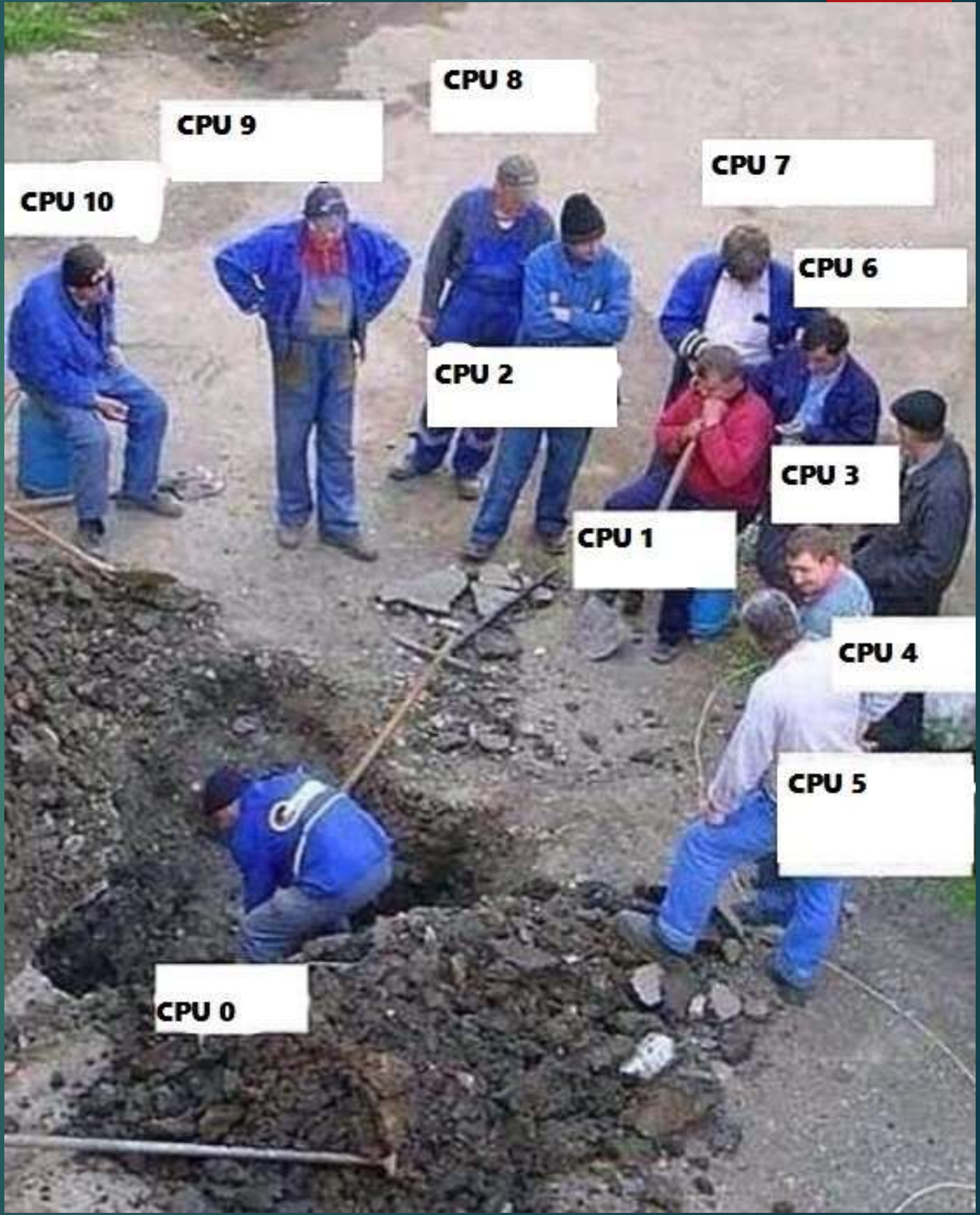
Instructions Grouping Intel 80486

- Integers instructions (32 bits):
  - Transfer data and exchange
  - Binary arithmetic (add, subtract, multiply, and divide)
  - Decimal arithmetic (BCD to binary)
  - Logical (AND, OR, XOR, NOT, etc.)
  - Shift and Rotate
  - Bit and Byte (set, test, etc.)
  - Control transfer
  - String instructions
  - Flag control instructions
  - Segment registers
- Floating-Point Instructions (80 bits):
  - Data Transfer and exchange
  - Basic Arithmetic (add, subtract, multiply, and divide)
  - Comparison
  - Transcendental (e.g., sine, cosine, etc.)
  - Load Constants (e.g. , pi, etc.)
  - Floating Point Unit Control
- System Instructions:
  - Support for Operating Systems (e.g. protect memory)

Today, we have multicore CPU



Multicore CPU are great, but they need to be used properly by the operating system or you end-up with this situation...



Any Questions
?