COL 100: Introduction to Computer Science

2020-21: Semester 2

What is computer science?

MISCONCEPTION 1:

Computer science is the study of computers.

MISCONCEPTION 2:

Computer science is the study of how to write computer programs.

MISCONCEPTION 3:

Computer science is the study of the uses and applications of computers and software.

What is computer science?

- Computer science is the study of *algorithms* including
  - 1. Their formal and mathematical properties
  - 2. Their linguistic realizations
  - 3. Their hardware realizations
  - 4. Their applications

An example of a very simple algorithm

- 1. Wet your hair.
- 2. Lather your hair.
- 3. Rinse your hair.
- 4. Stop.

Observe:

Operations need not be executed by a computer only, but by any entity capable of carrying out the operations listed.

The algorithm begins executing at the top of the list of operations.

A more complex example

- Searching for a word in a dictionary

1. Start with page 1
2. Start with the first word on the page
3. If this word is the same as the target word, then
   read its meaning
   go to step 7
4. If more words on this page
   move to the next word on this page
   go to step 3
5. If reached the end of this page and more pages remain
   move to next page
   go to step 2
6. Declare target word as not found
7. Stop.

A more efficient algorithm

- https://www.youtube.com/watch?v=OUP-F5Oeng8

Formal and mathematical properties of algorithms

- Properties of an algorithm:
  - How efficient is it?
  - What kinds of resources must be used to execute it?
  - How does it compare to other algorithms that solve the same problem?

Linguistic realizations to express algorithms

- **How do we represent algorithms?**
- Pseudocode, high level programming languages, low level programming languages

- Our key focus in this course! Programming in C

```
// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}
```

Course Materials

Please use the lecture slides as your main resource. For C programming, an easy text you can also follow is:

Programming in C, by Stephen G. Kochan. Available here.

Additional books:

1. Programming with C, by Byron Gottfried
2. The C Programming Language, by Brian W Kernighan, Dennis M Ritchie

Linguistic realizations to express algorithms

- **Algorithms operate on data. Closely intertwined is the representation of data**

- Another example: Matching of ads
- Check your Google ad profile! Go to My Account -> Data and personalization -> Ad settings -> Advanced
- How would this be represented in a computer?

Simple data structure example

	I_1	I_2		I_j		I_{m-1}	I_m
U_1			…		…		
U_2			…		…		
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
U_i			…	A_{ij}	…		
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
U_{n-1}			…		…		
U_n			…		…		

- Express ads in terms of an Interest vector
- Show ads which match the user profile
- But sparse matrix with lots of unknown data about users

Simple data structure example

	I_1	I_2		I_j		I_{m-1}	I_m
I_1	1	Sim_{12}	…	Sim_{1j}	…		
I_2		1	…		…		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
I_i		Simi2	…	Sim_{ij}	…		
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
I_{m-1}			…		…	1	
I_m			…		…		1

- Find correlations between interests, use this to “pre fill” the UxI matrix or to reduce the interest dimensions
- At heart: Matrix operations on very large matrices.
- More challenges: Efficient search given many ads to choose from?

Hardware realizations to execute algorithms

- Algorithms need not execute on machines. All we really need are computing entities.
  - Anything that can compute – e.g., a human.
- But, ultimately, most of our interest will lie with algorithms that execute on computing entities called "computers".
- How are these entities constructed?

Applications of algorithms

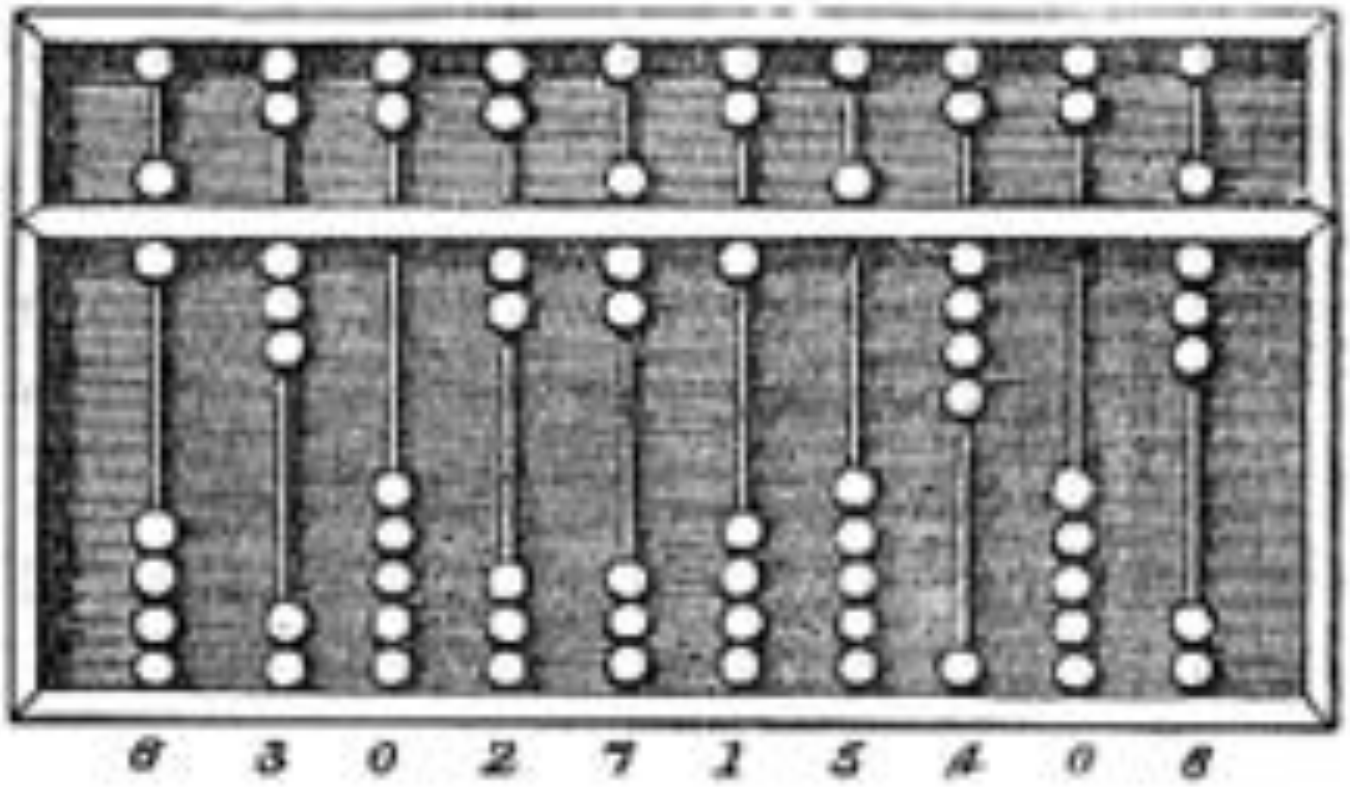
- What are some of the many important and popular applications of computers in current use including:
  - modeling and simulation
  - information retrieval
  - numerical problem solving
  - data analysis
  - artificial intelligence
  - networking
  - graphics

What is computer science?

- Computer science is the study of *algorithms* including
  - 1. Their formal and mathematical properties
  - 2. Their linguistic realizations
  - 3. Their hardware realizations
  - 4. Their applications

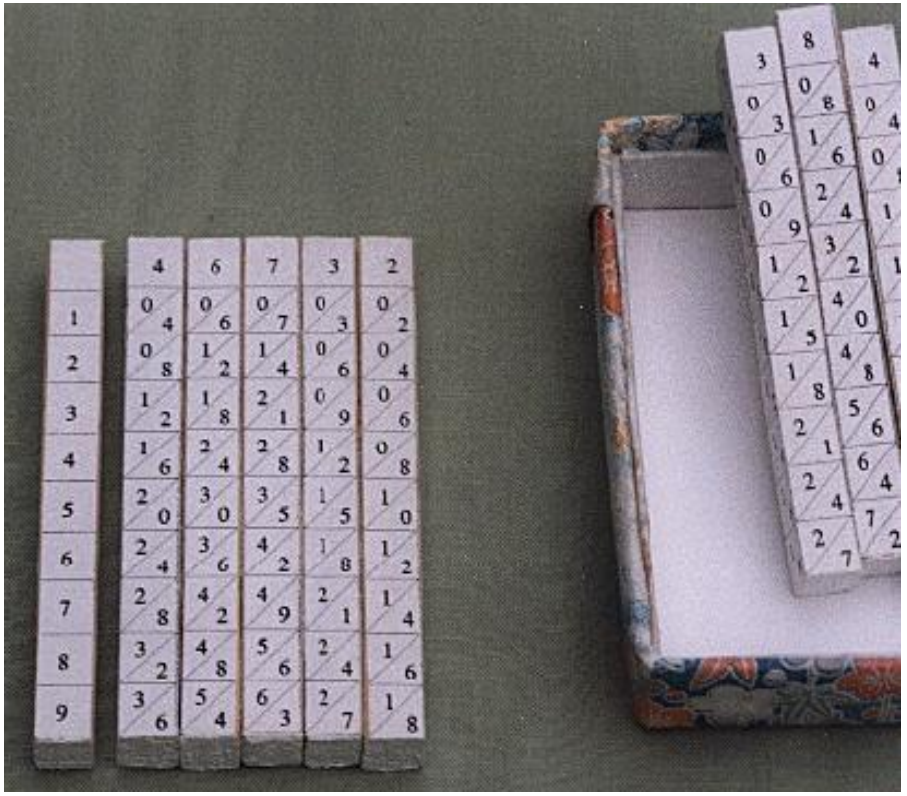
A brief history of computing

Abacus



- 2nd century BC
- Not really a computing device, but a counting device to count large numbers

Napier's Bones



46732 x 5

Add:

10

150

3500

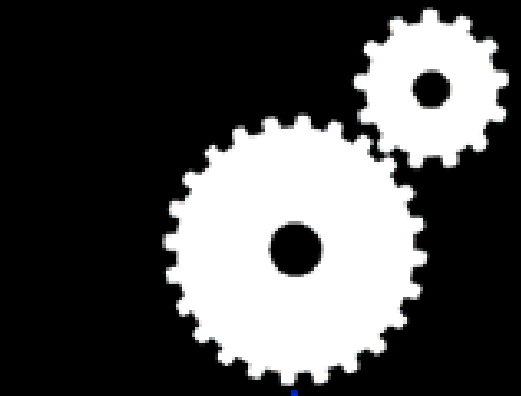
30000

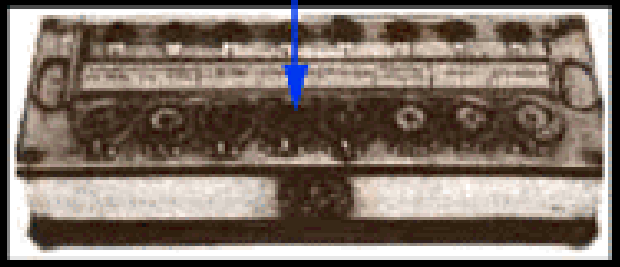
200000

- 1616 AD
- Closer to computing

Pascaline: 1642 AD

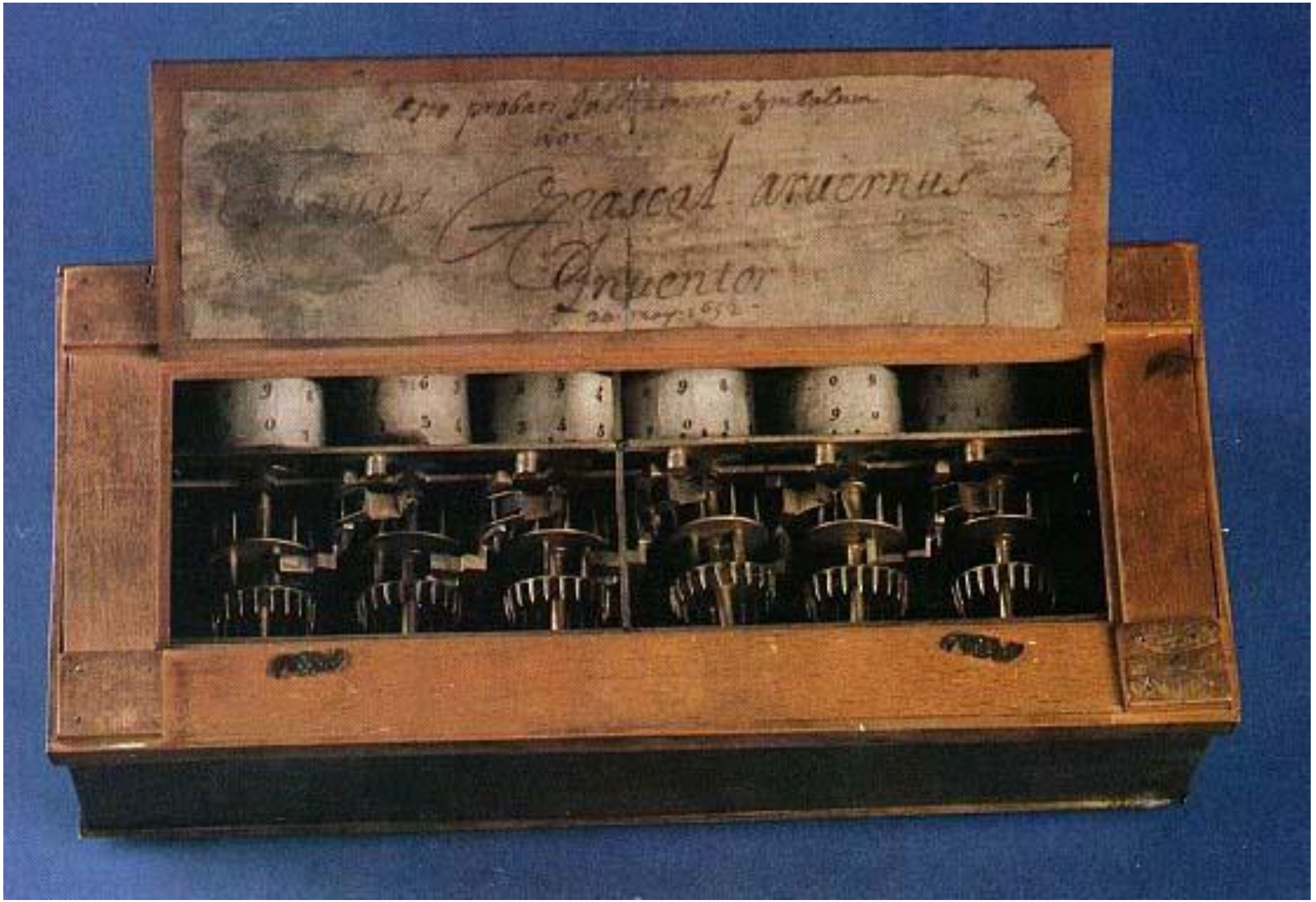
Pascal's Gear System





Pascal's Computer: The Pascaline

A one-tooth gear engages its single tooth with a ten-teeth gear once every time it revolves; the result will be that it must make ten revolutions in order to rotate the ten-teeth gear once.
This is the way that an odometer works for counting kilometers. The one-tooth gear is large enough so that it only engages the next size gear after 1 km has passed.


Inventor

Joseph Jacquard

- In the late 1700s in France, Joseph Jacquard invented a way to control the pattern on a weaving loom used to make fabric.
- Jacquard punched pattern holes into paper cards.
- The cards told the loom what to do.
- Instead of a person making every change in a pattern, the machine made the changes all by itself.
- Jacquard's machine didn't count anything. So it wasn't a computer or even a computing device. His ideas, however, led to many other computing inventions later.

Jacquard's Loom

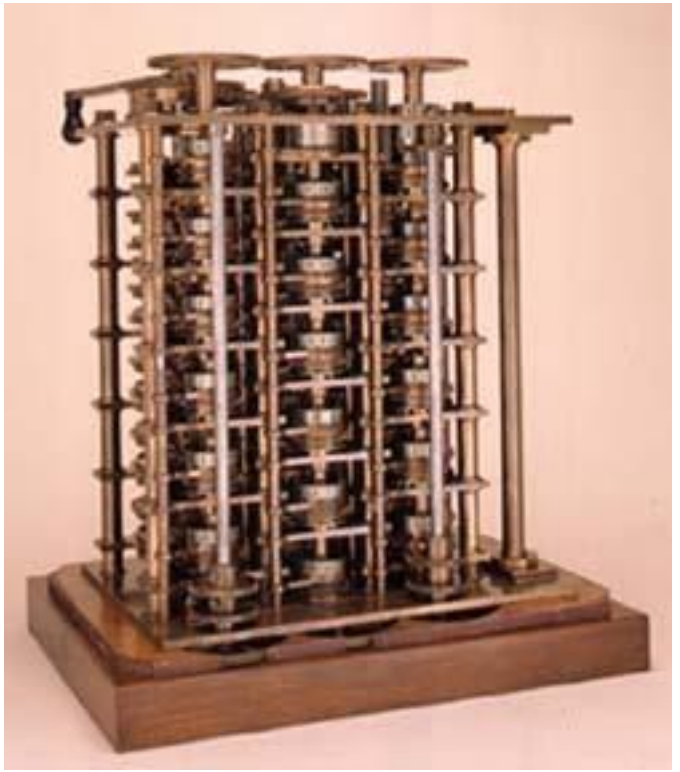
Intricate textile patterns were prized in France in early 1800s. Jacquard's loom (1805-6) used punched cards to allow only some rods to bring the thread into the loom on each shuttle pass.

Jacquard's Loom

http://www.vam.ac.uk/content/videos/j/video-jacquard-weaving/

Charles Babbage

- Babbage is known as the father of modern computing because he was the first person to design a general purpose computing device.
- In 1822, Babbage began to design and build a small working model of an automatic mechanical calculating machine, which he called a "difference engine".



Charles Babbage

- Babbage continued work to produce a full scale working Difference Engine for 10 years, but in 1833 he lost interest because he had a "better idea"--the construction of what today would be described as a general-purpose, fully program-controlled, automatic mechanical digital computer.

- Babbage called his machine an "analytical engine".

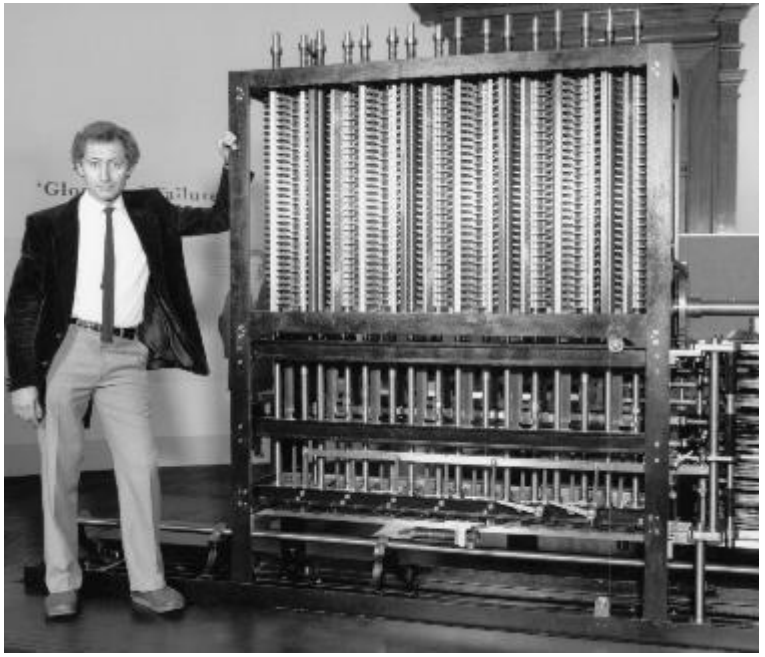
- He designed, but was unable to build, this Analytical Engine (1856) which had many of the characteristics of today's computers:

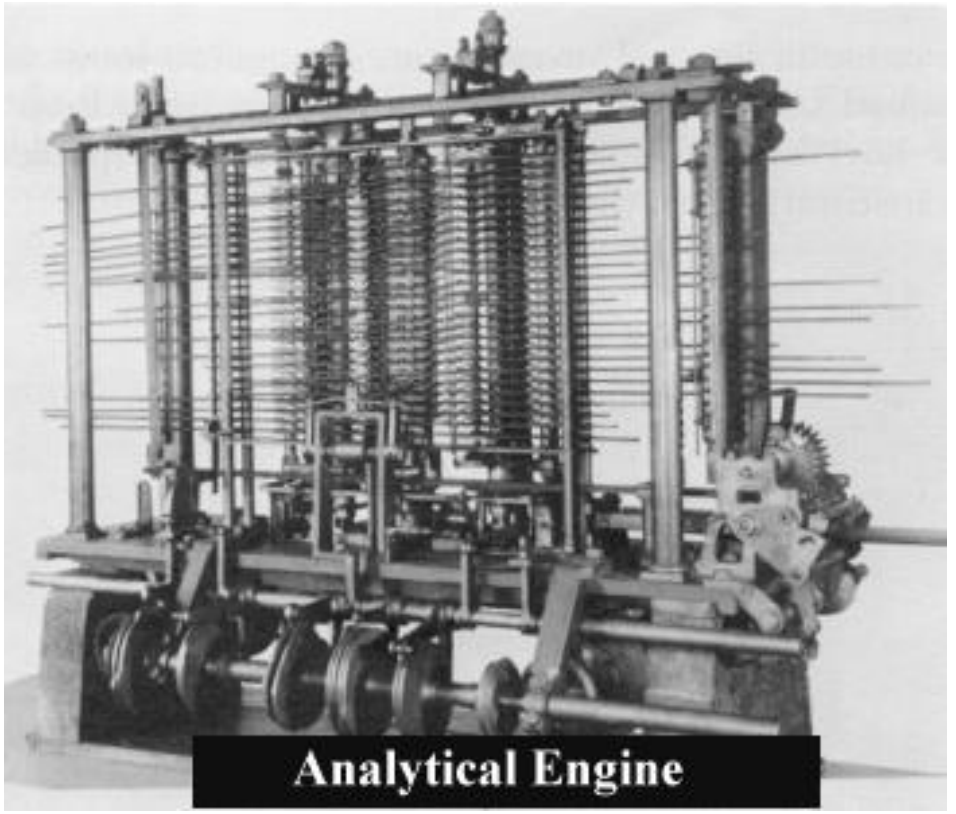
  an input device – punched card reader

  an output device – a typewriter

  memory – rods which rotated into position "stored" a number

  control unit – punched cards that encoded instructions





The machine was to operate automatically, by steam power, and would require only one attendant.

Herman Hollerith

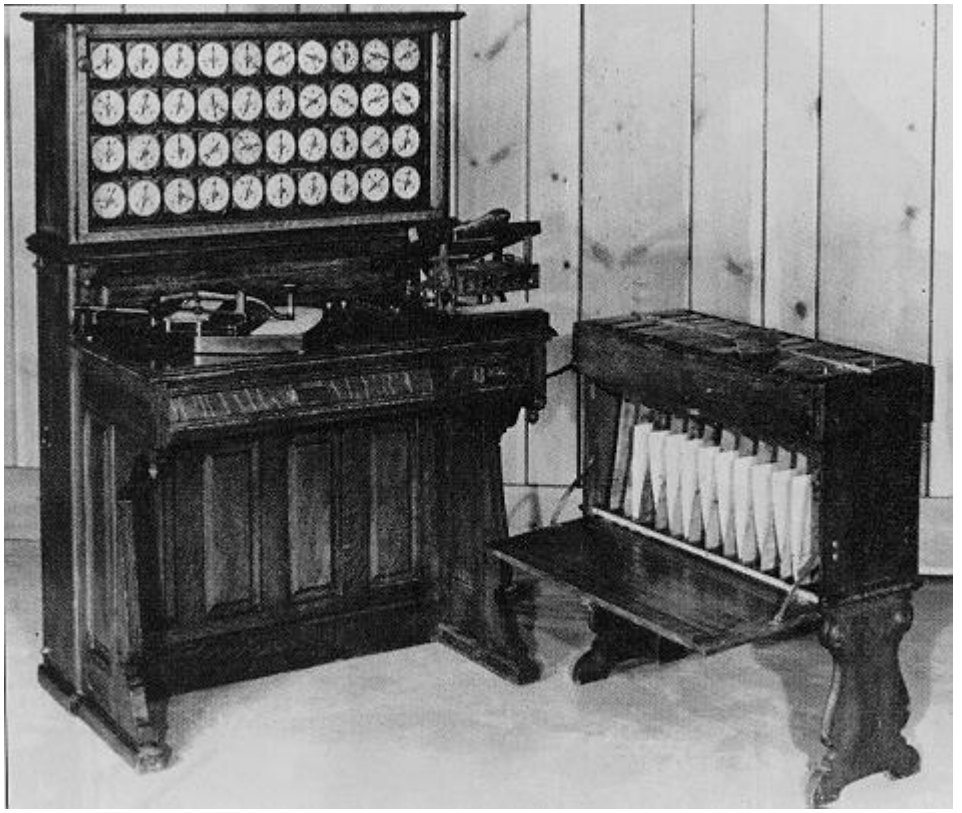
- In 1886, Herman Hollerith invented a machine known as the Automatic Tabulating Machine, to count how many people lived in the United States.
- This machine was needed because the census was taking far too long.
- His idea was based on Jacquard's loom. Hollerith used holes punched in cards. The holes stood for facts about a person; such as age, address, or his type of work. The cards could hold up to 240 pieces of information.
- Hollerith also invented a machine, a tabulator, to select special cards from the millions.
- To find out how many people lived in Pennsylvania, the machine would select only the cards punched with a Pennsylvania hole. Hollerith's punched cards made it possible to count and keep records on over 60 million people.

Hollerith Tabulator

Hollerith founded the Tabulating Machine Company.

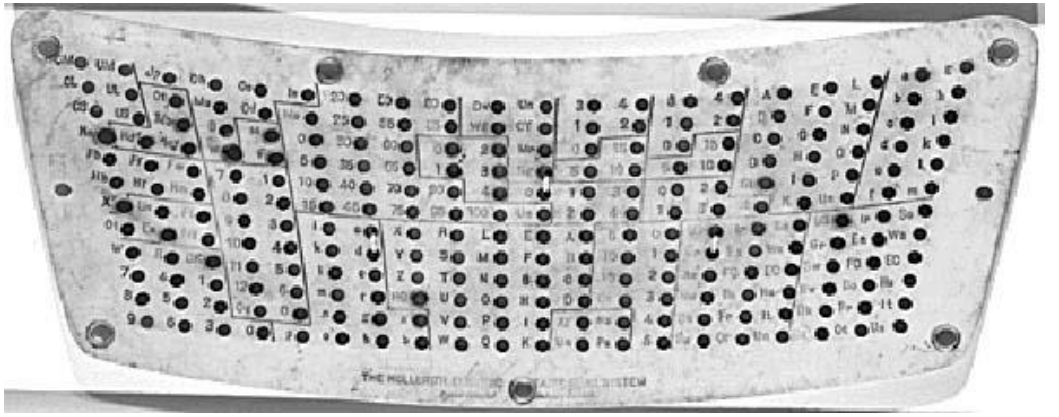
In 1924, the name of the company was changed to International Business Machines Corporation (IBM).

This is the 1890 version used in tabulating the 1890 federal census.



Punched cards

The punched card used by the Hollerith Tabulator for the 1890 US census.



The punched card was standardized in 1928:
It was the primary input media of data processing and computing from 1928 until the mid-1970s and was still in use in voting machines in the 2000 USA presidential election.

The digital revolution: Mark 1

An electro-mechanical computer: 1939-1944

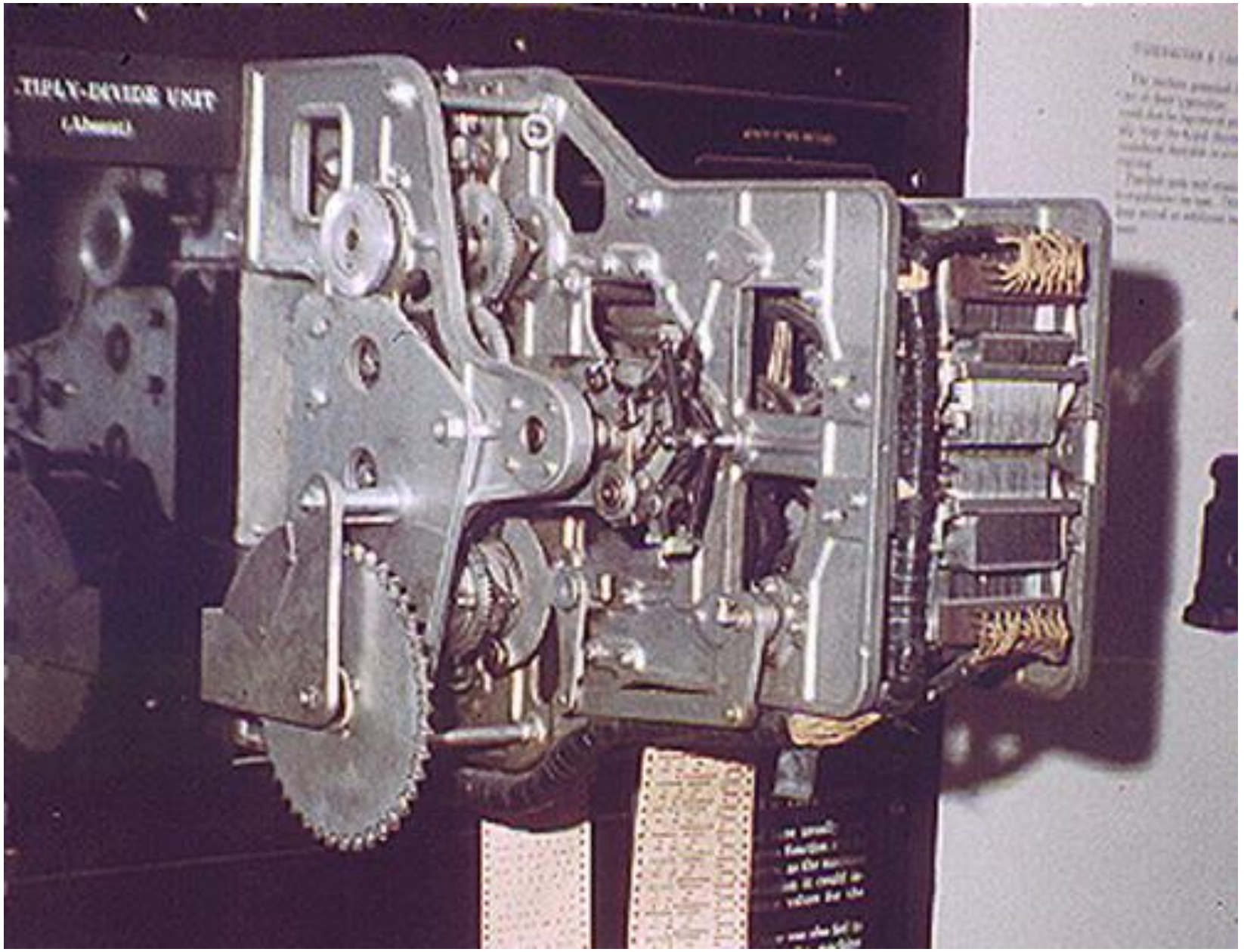
Contains more than 750,000 components, is 50 feet long, 8 feet tall, and weighs approximately 5 tons

Instructions were pre-punched on paper tape

Input was by punched cards

Output was displayed on an electric typewriter.

Could carry out addition, subtraction, multiplication, division and reference to previous results.



One of the four paper tape readers on the Harvard Mark I

9-2

9/9

0800 Arctan started
1000 " stopped - arctan ✓
13'6c (032) MP - MC
(033) PRO 2
correct

{ 1.2700 9.037 847 025
9.037 846 995 correct

~~1.9526447000~~ 2.13047676715 ~~(-3)~~ 4.61592505(-2)
2.13047676715
2.13067676715

Relays 6-2 in 033 failed special speed test
in Relay " " 11.00 test.
Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Mult + Adder Test.

1545 

Relay #70 Panel F
(Moth) in relay.

First actual case of bug being found.

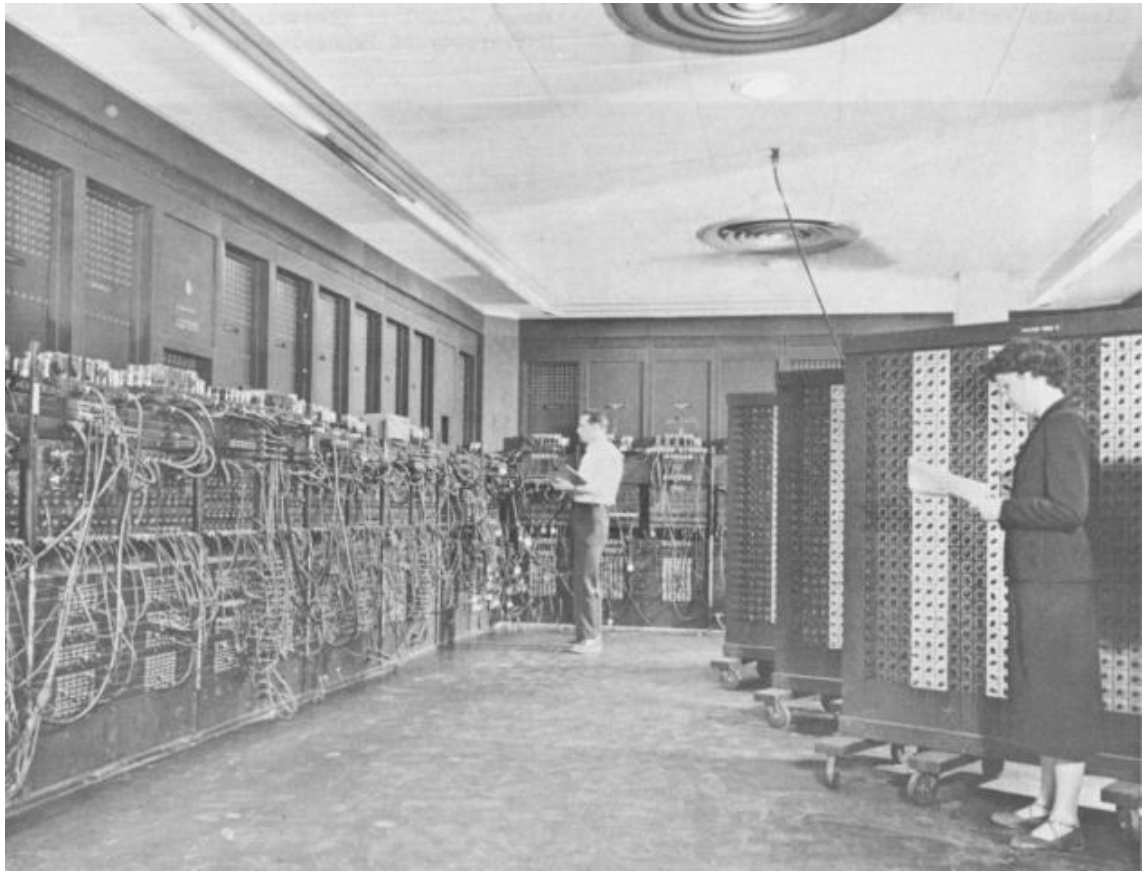
~~1525~~ 1530 Arctangent started.
1700 closed down.

The first computer bug

Computer vs. Computing device

- Most, but not all, people claim a computer must be
  - Digital
  - Programmable
  - Electronic
  - General purpose
- If any characteristic is missing, at best, you have a computing device.

ENIAC – (**E**lectrical **N**umerical **I**ntegrator **A**nd **C**alculator), built by Presper Eckert and John Mauchly at Moore School of Engineering, University of Pennsylvania, 1941-46

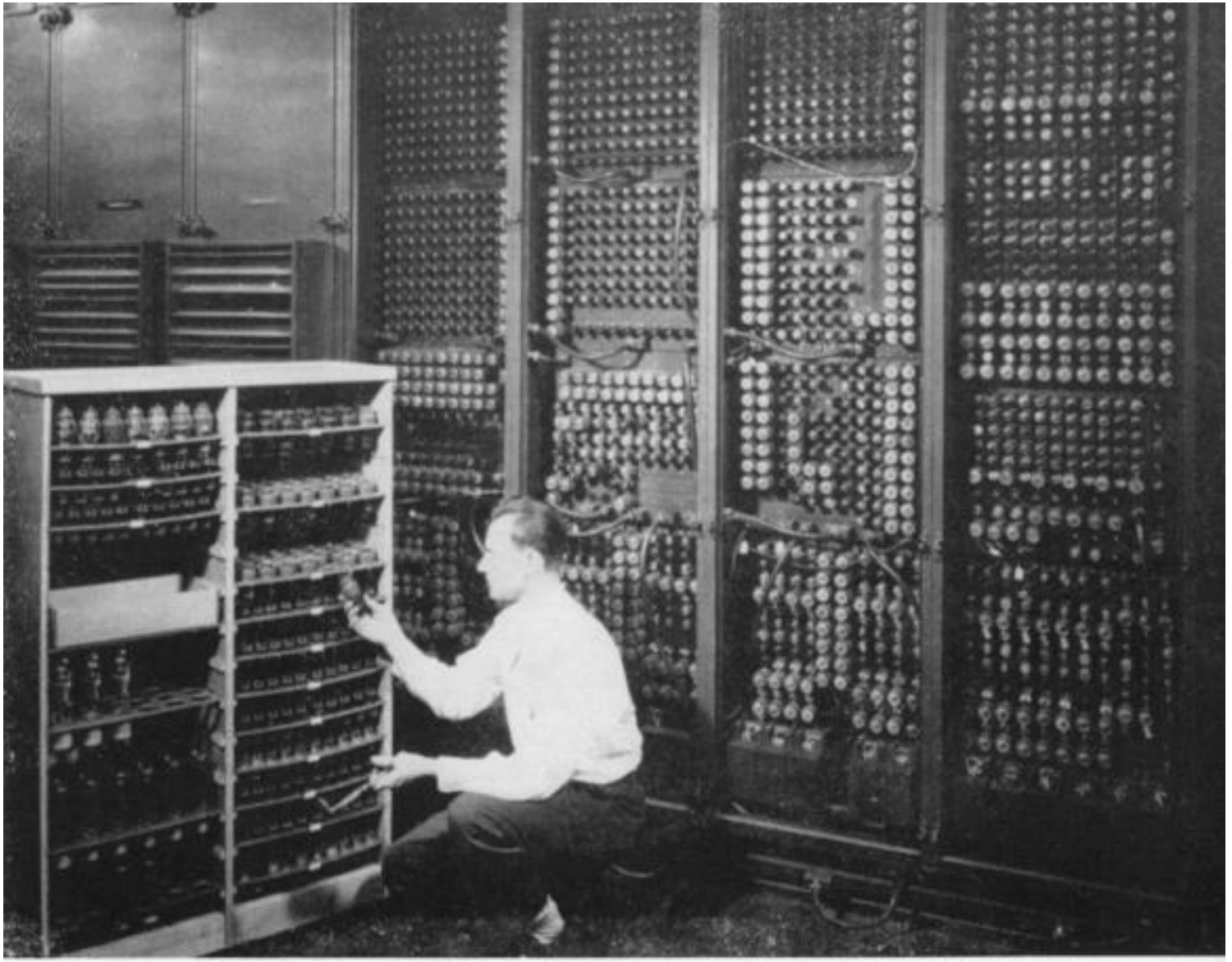


Often called the first computer (that was electronic, programmable, general purpose and digital).

ENIAC

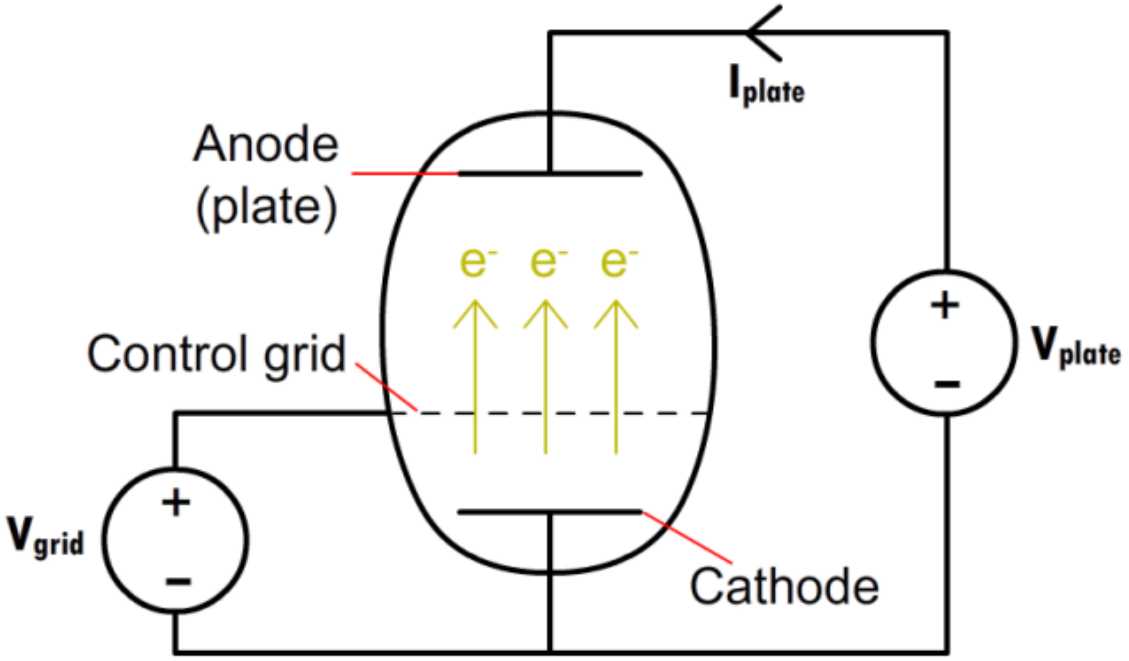
- 19,000 vacuum tubes and weighed 30 **tons**
- Duration of an average run without some failure was only a few hours
- When it ran, the lights in Philadelphia dimmed!
- ENIAC Stored a maximum of twenty 10-digit decimal numbers.
- Input: IBM card reader
- Output: Punched cards, lights

ENIAC's vacuum tubes

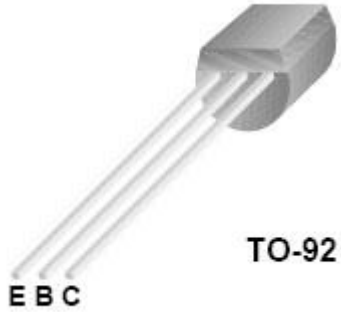


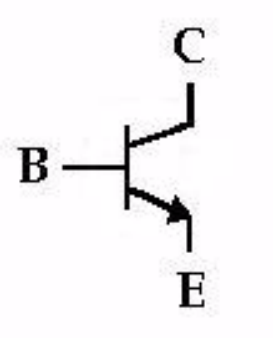
Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

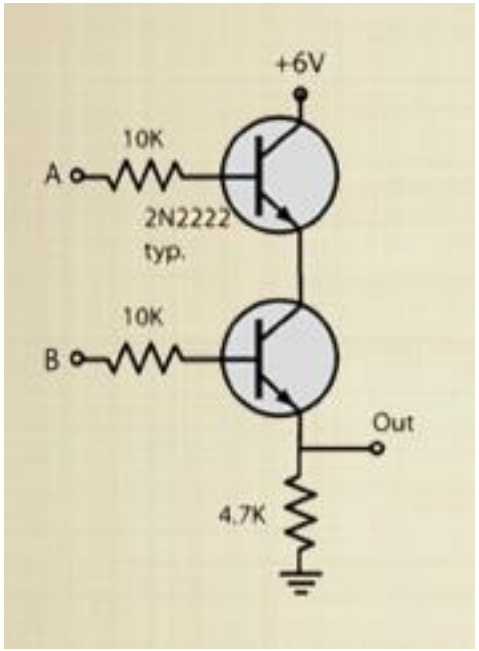
Vacuum tubes



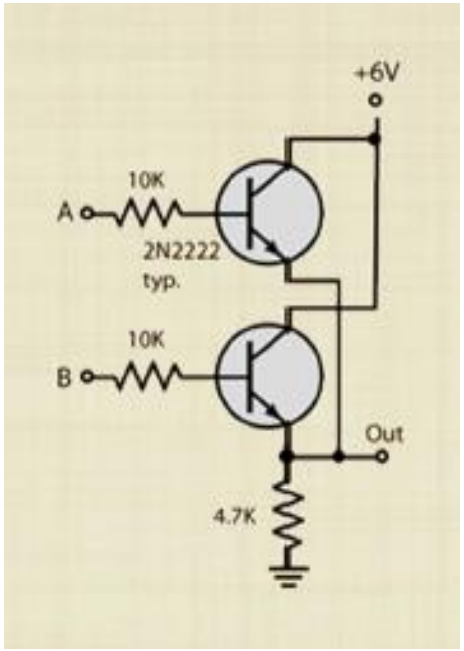
Transistors! 1947 – Shockley, Bardeem, and Brattain @ Bell Labs



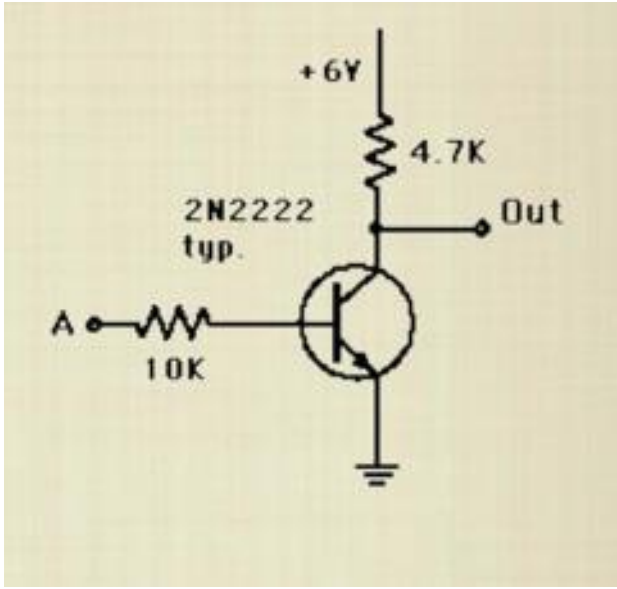




AND gate

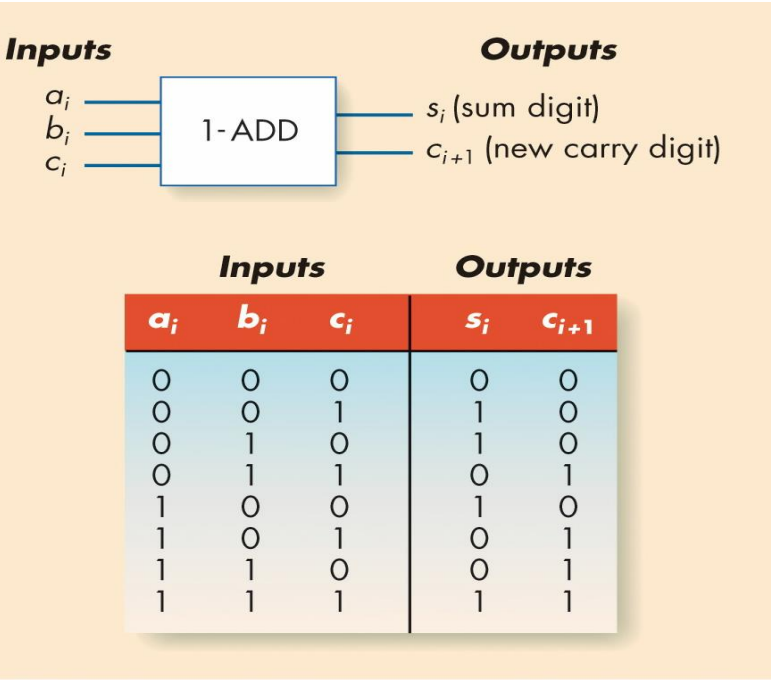


OR gate



NOT gate

From gates to logic circuits



Inputs			Outputs	
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

1 bit adder

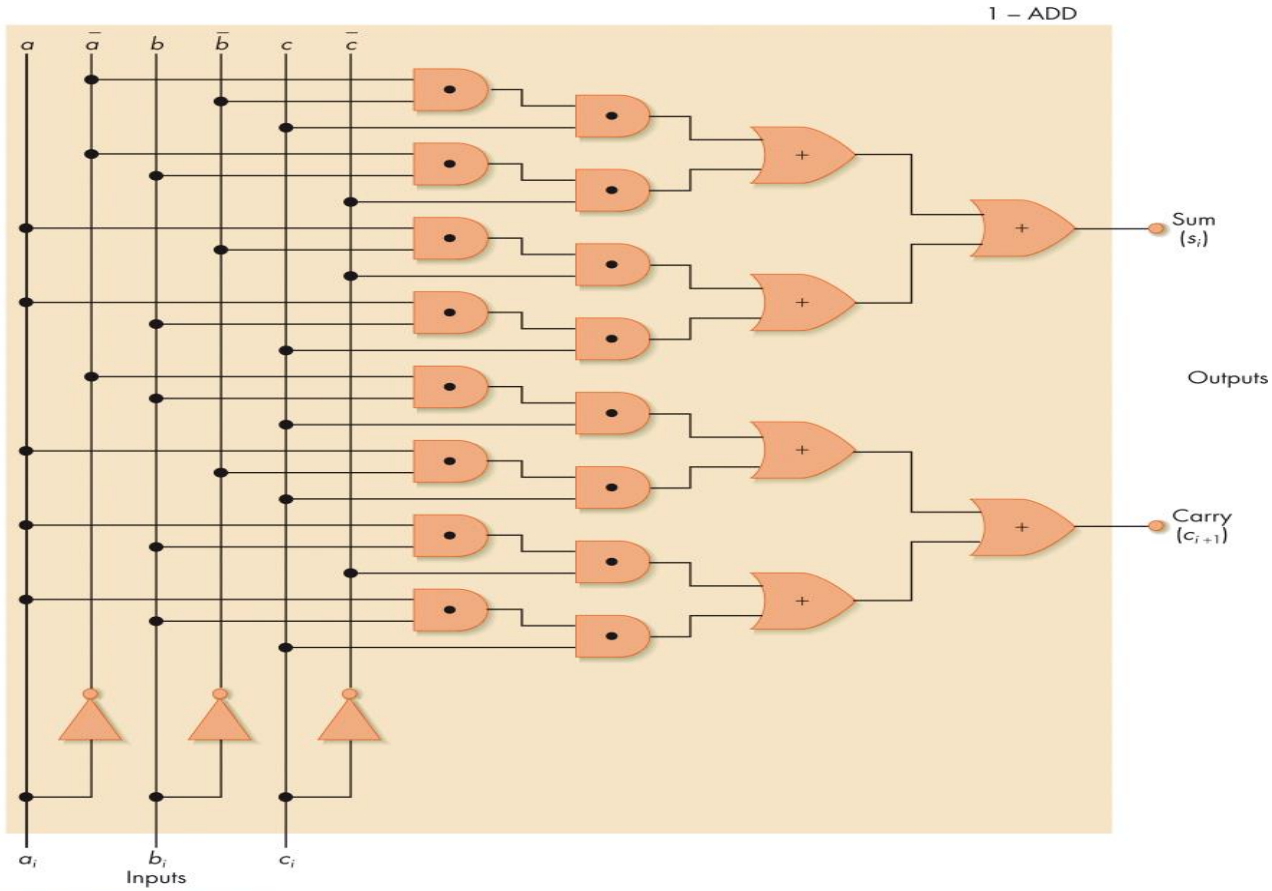


FIGURE 4.26
Complete 1-ADD Circuit for 1-Bit Binary Addition

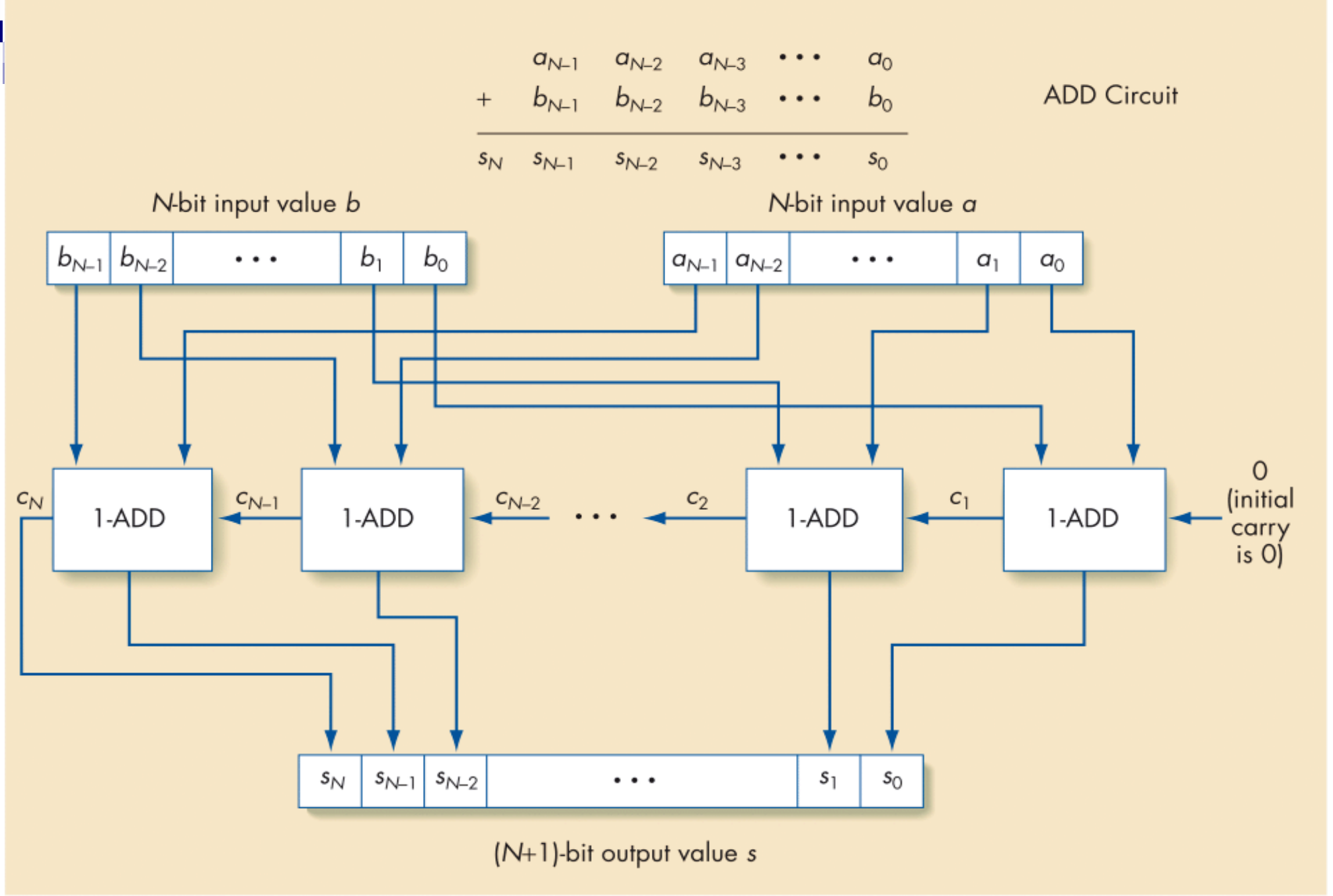
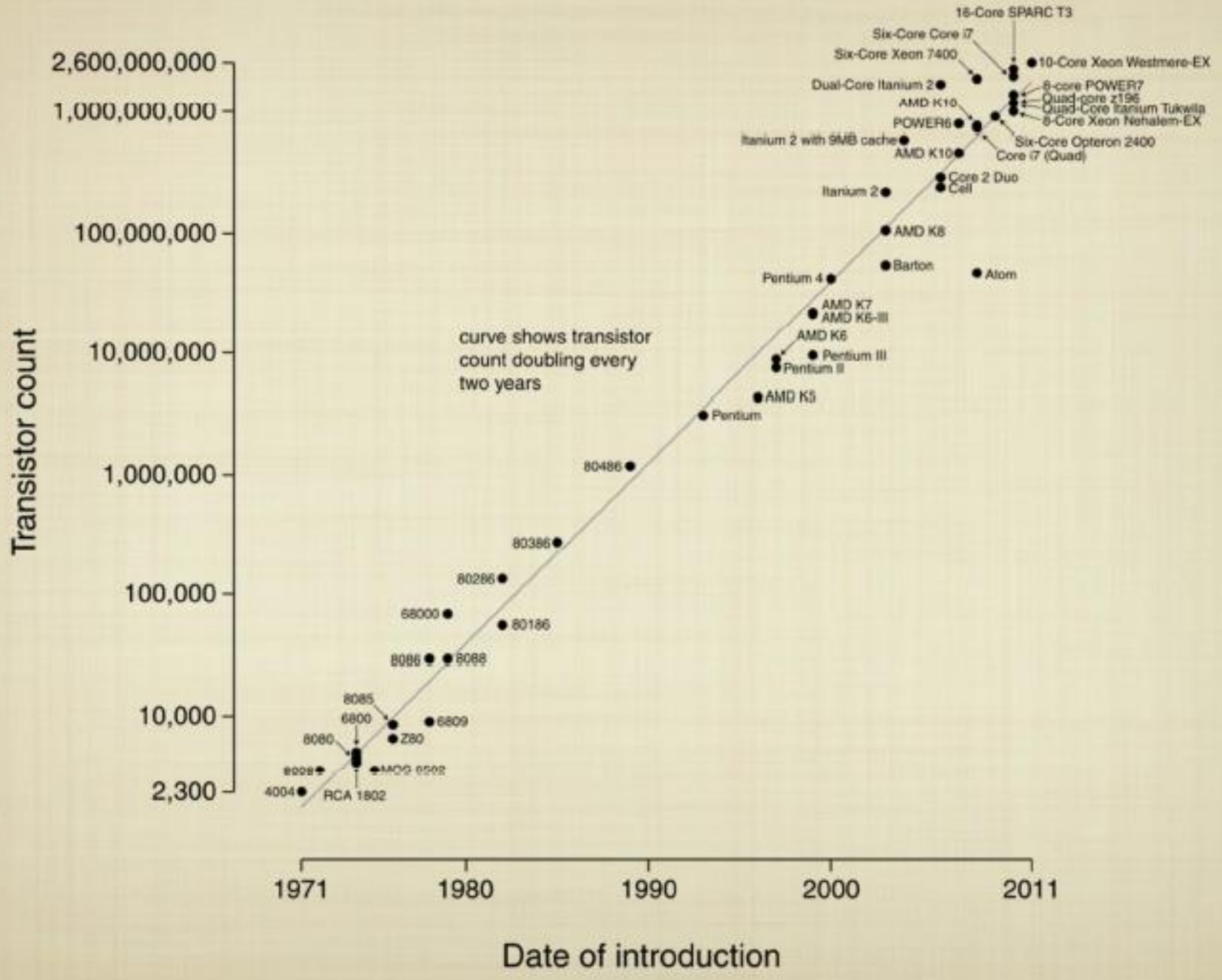
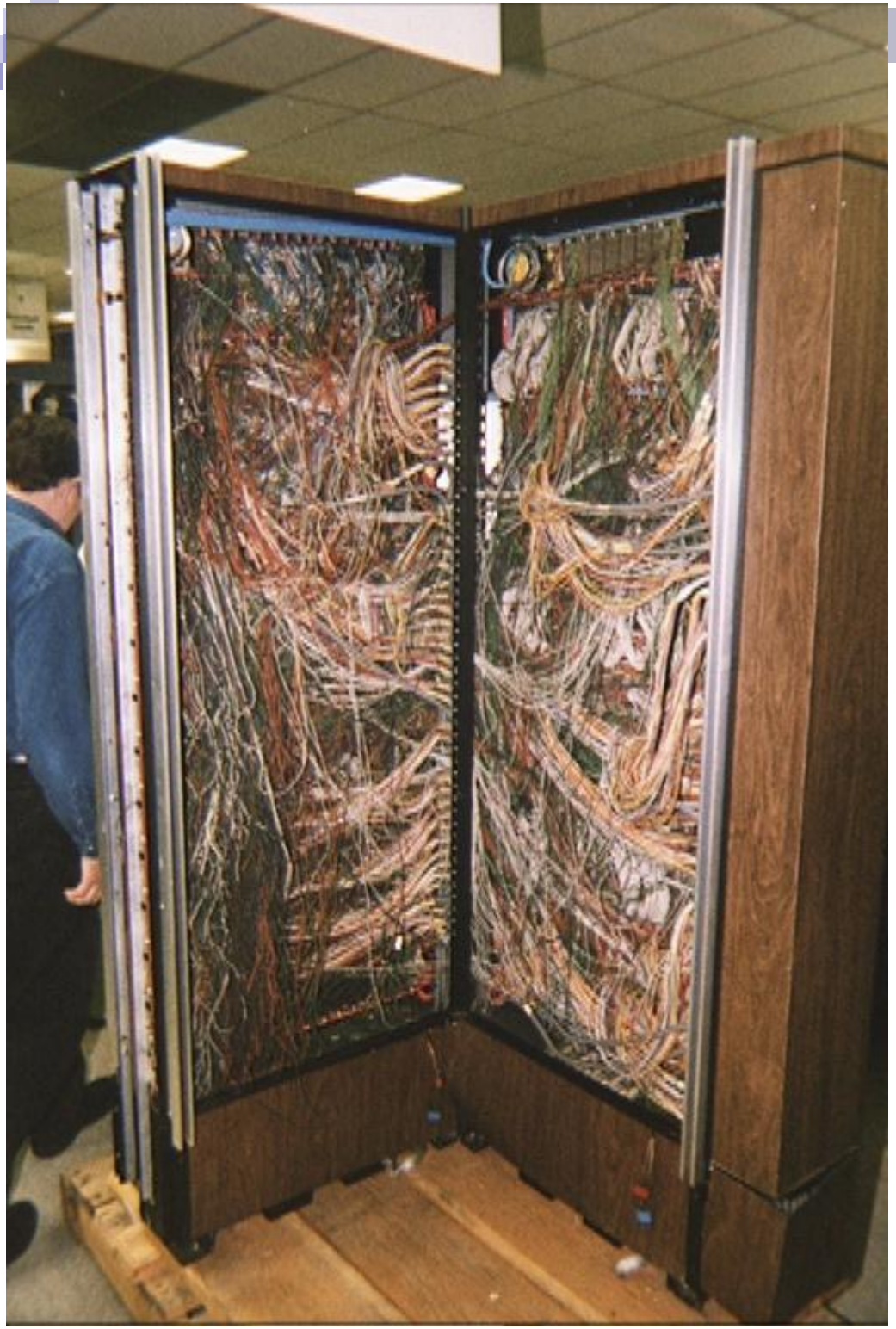


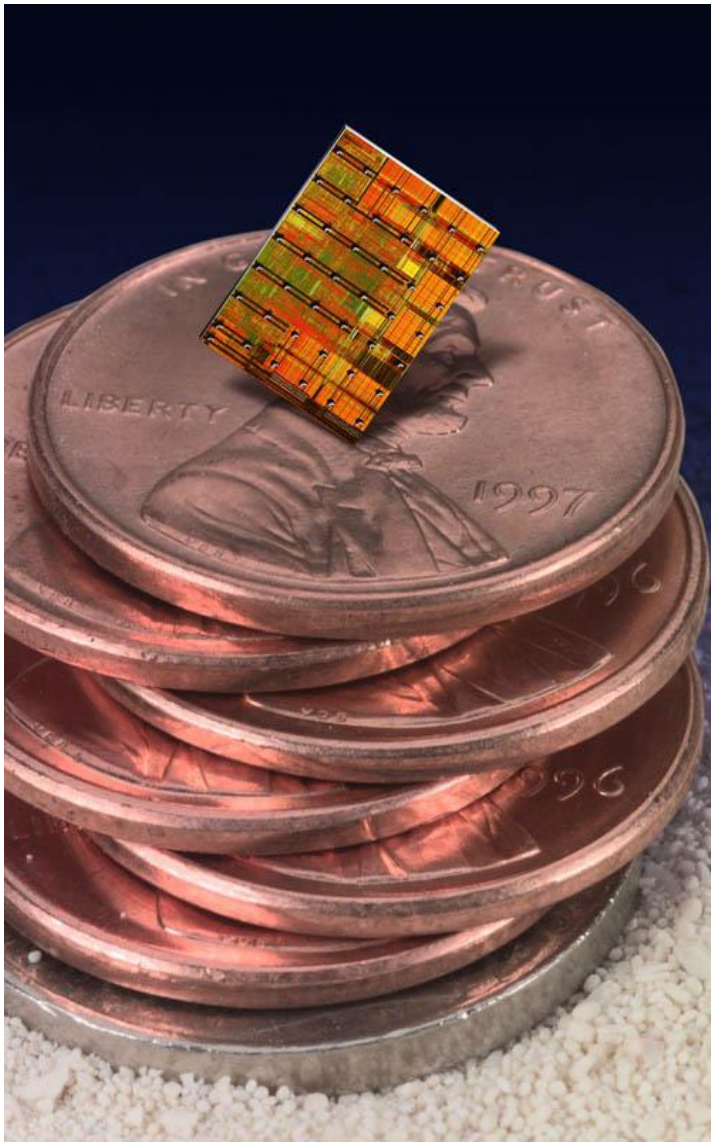
FIGURE 4.27

The Complete Full Adder ADD Circuit

Microprocessor Transistor Counts 1971-2011 & Moore's Law







Left: Typical wiring in an early mainframe computer
Right: An integrated circuit

Von Neumann architecture

There are 3 major units in a computer tied together by buses:

1) Memory: The unit that stores and retrieves instructions and data.

2) Processor: The unit that houses two separate components:

The control unit: Repeats the following 3 tasks repeatedly

Fetches an instruction from memory

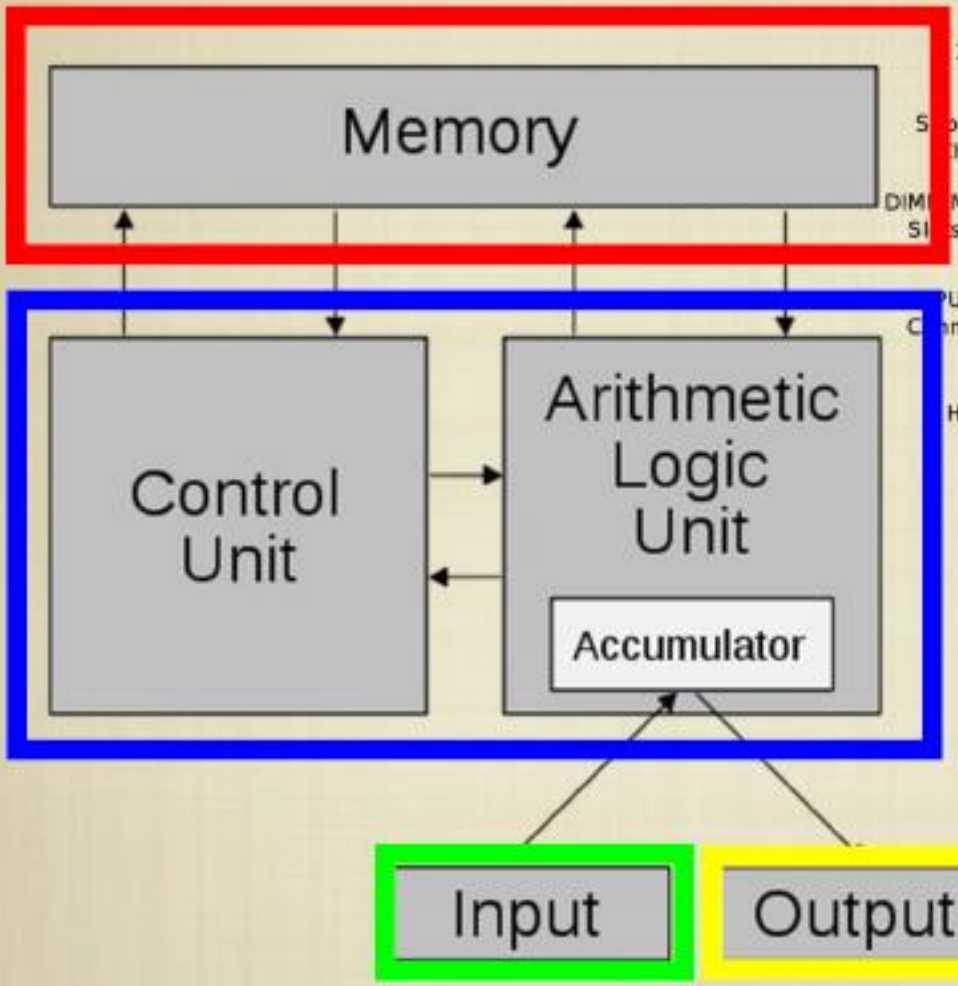
Decodes the instruction

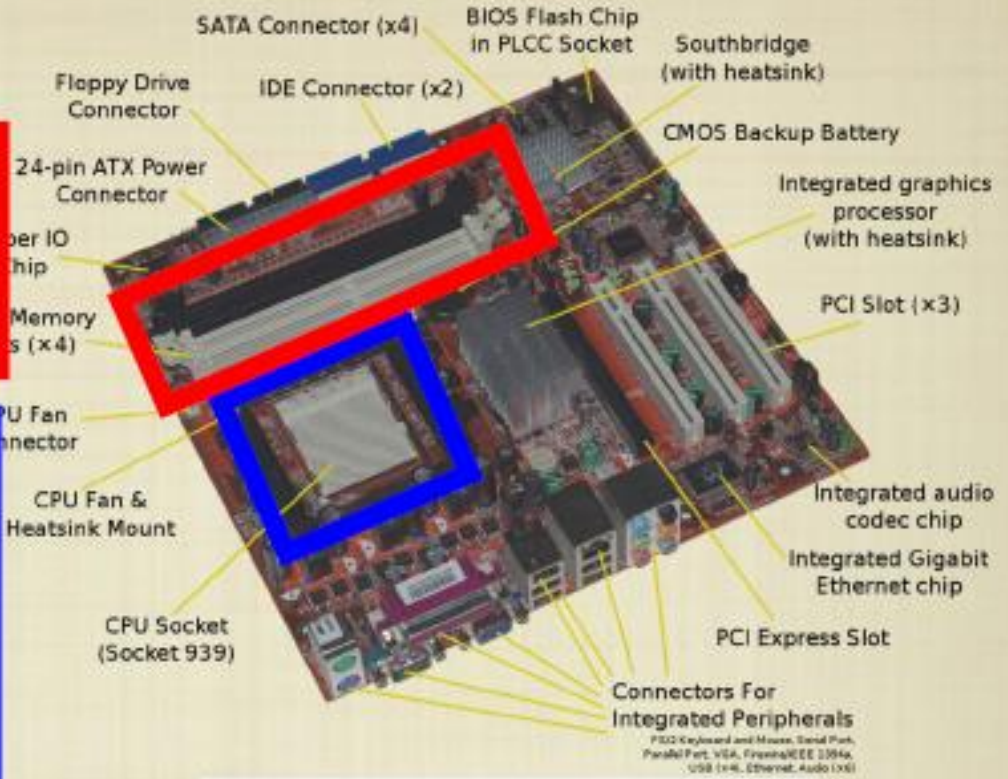
Executes the instruction

The arithmetic/logic unit (ALU): Performs mathematical and logical operations.

3) Input/Output (I/O) Units: Handles communication with the outside world.

von Neumann Architecture

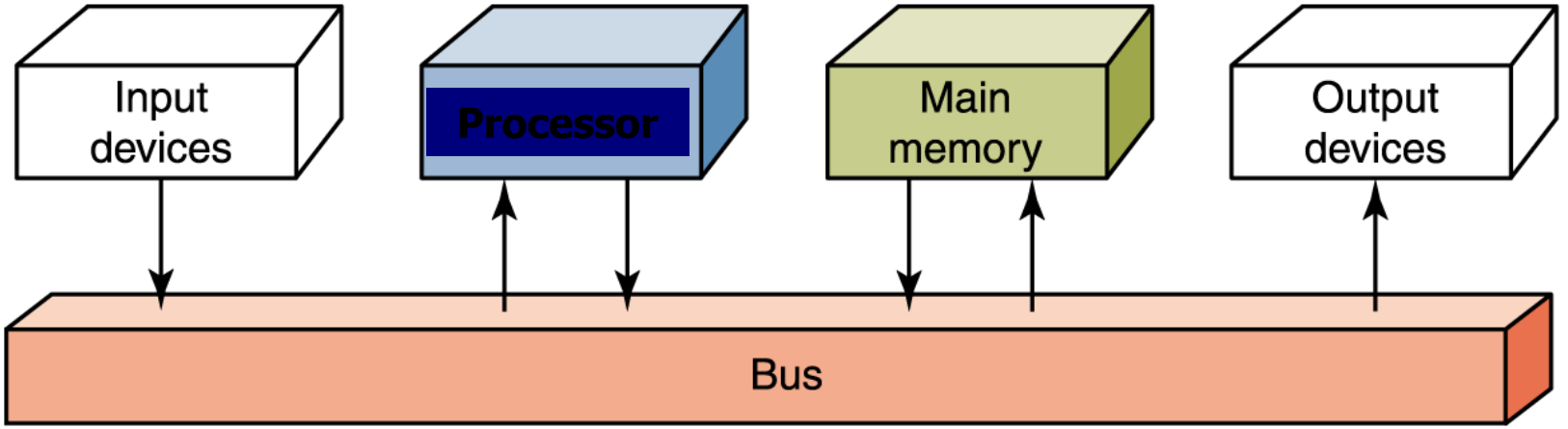




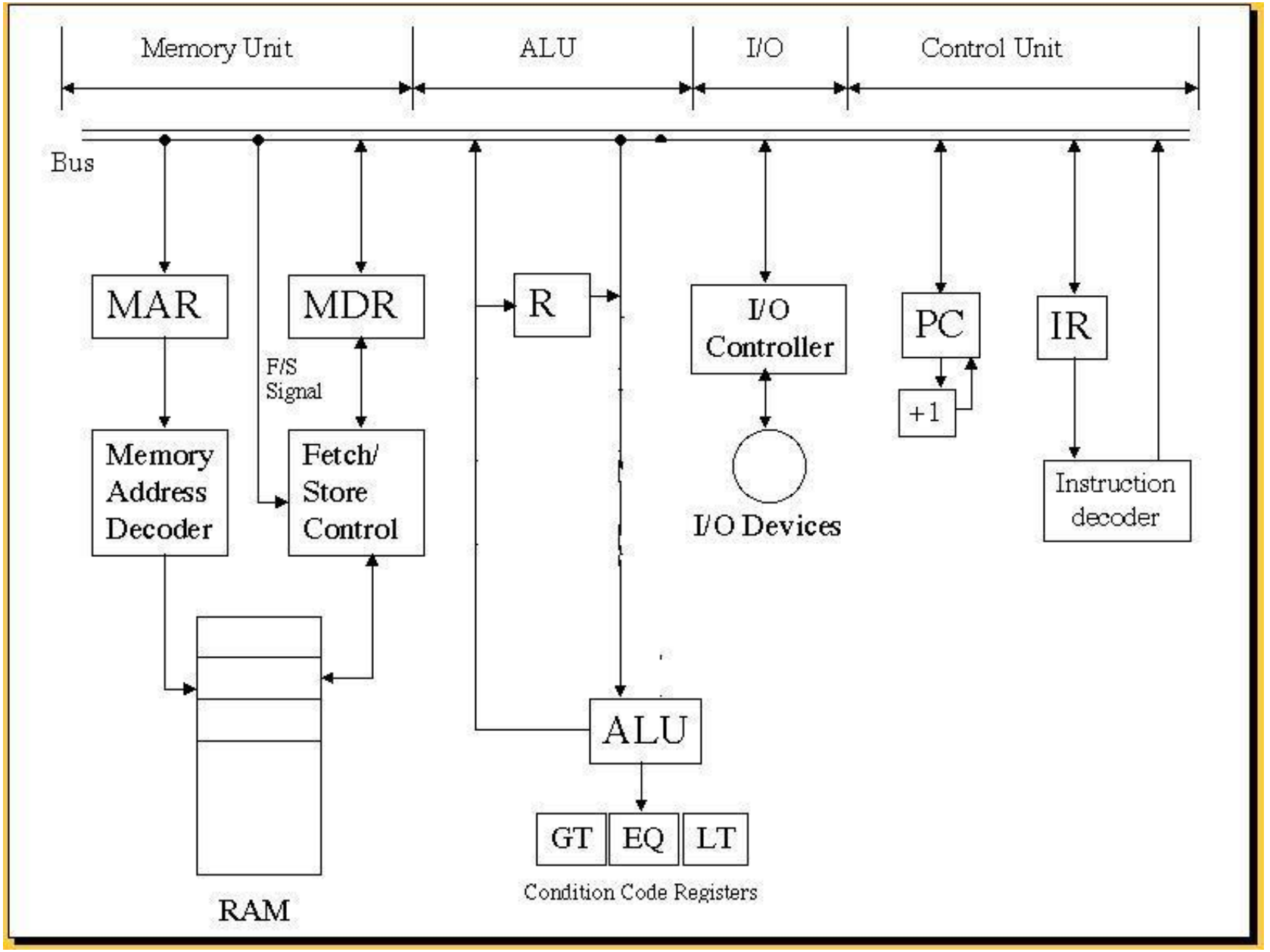
Every modern computational device has a von Neumann architecture.

Flow of information

- The parts are connected to one another by a collection of wires called a bus

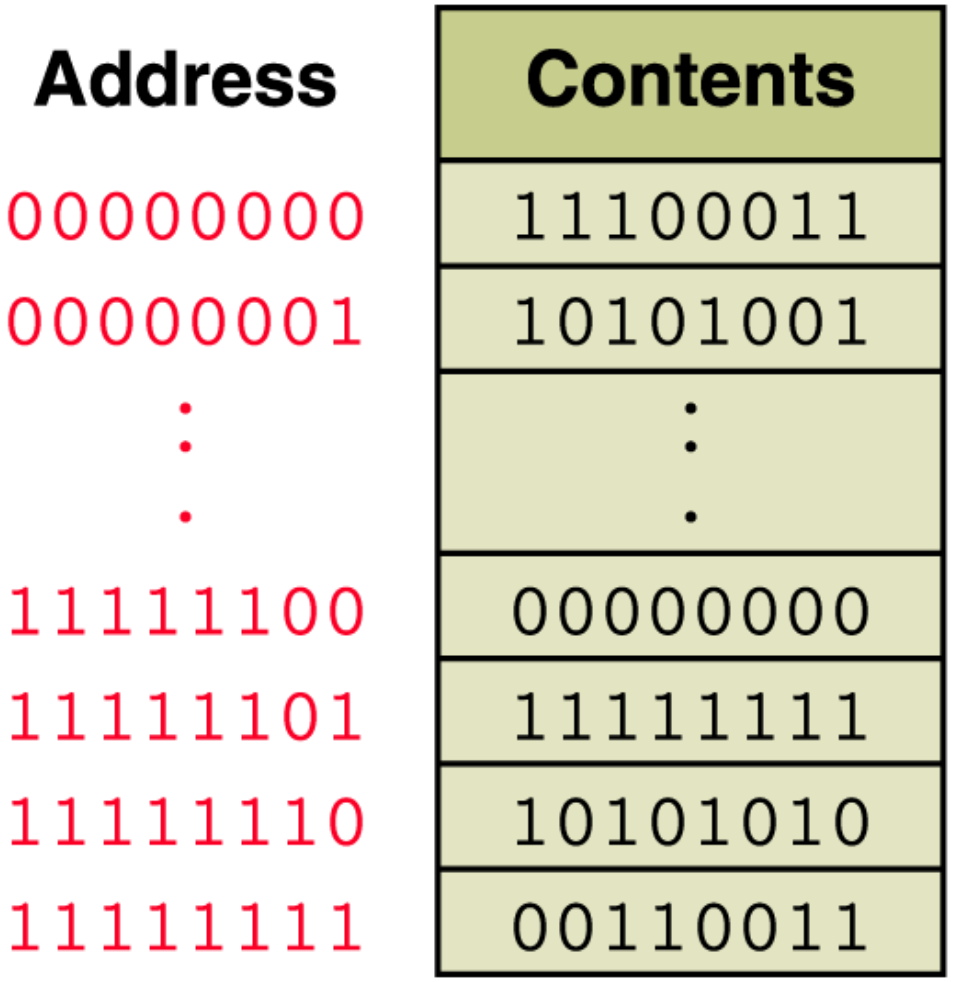


Executing instructions



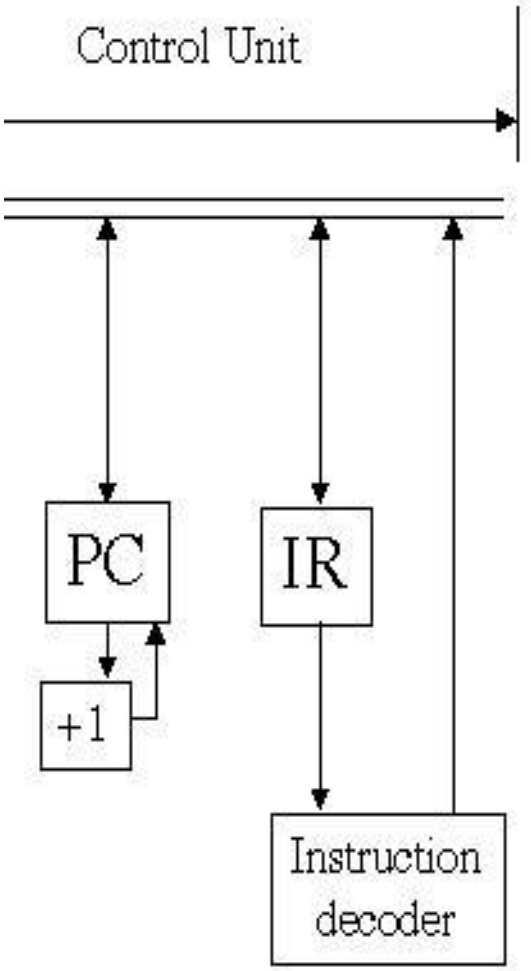
Memory

- Memory is a collection of cells, each with a unique physical address
- The size of a cell is normally a power of 2, typically a byte (8 bites) today.
- RAM: Random Access Memory
- 32-bit vs. 64-bit architecture



Control unit

Once the instruction is fetched, the PC (Program Counter) is incremented.



The PC holds the address of the next instruction to be executed.

Whatever is stored at that address is assumed to be an instruction.

Examples of OpCodes

Arithmetic OpCodes

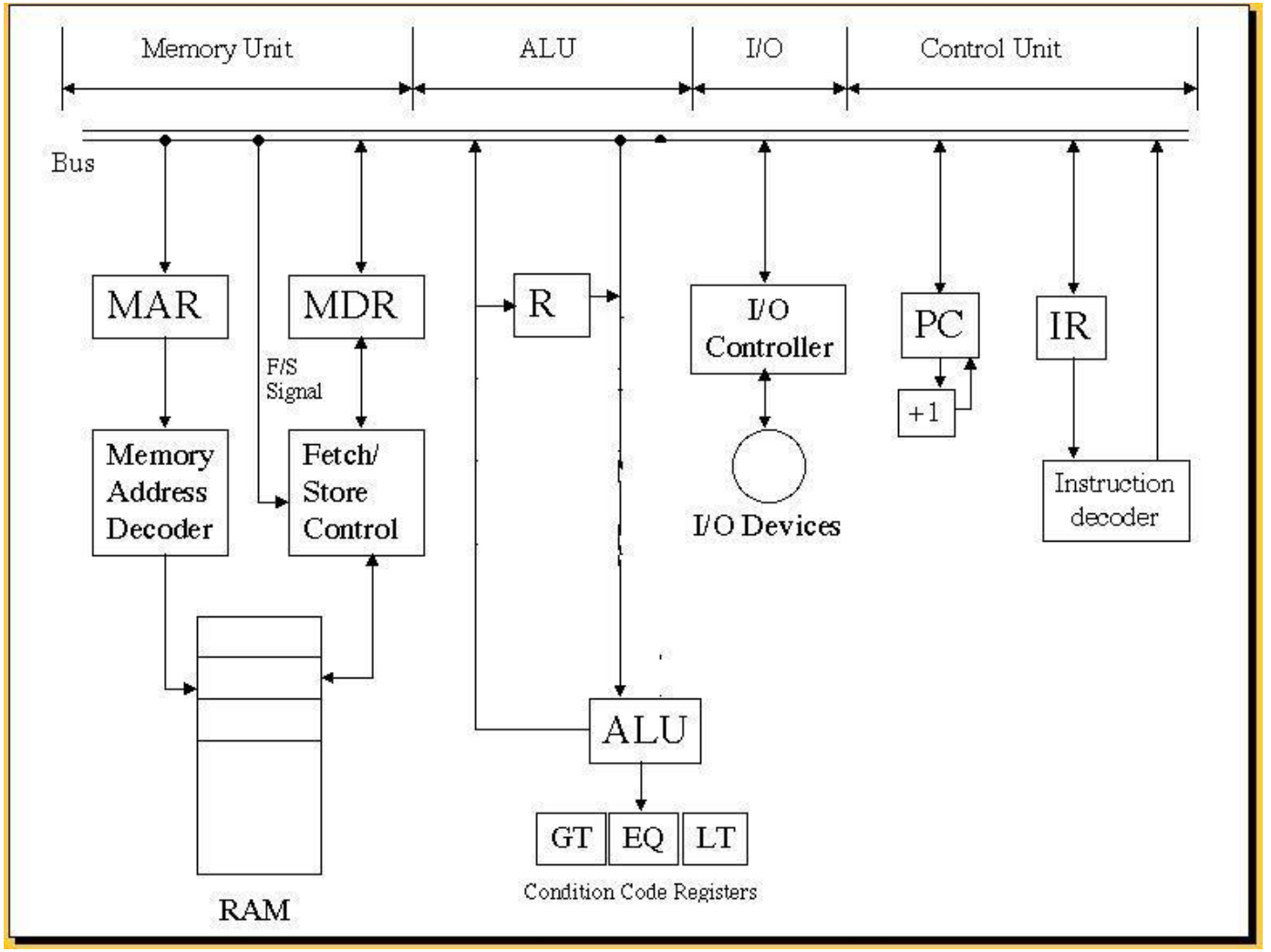
- 0000 load
- 0001 store
- 0010 clear
- 0011 add
- 0100 increment
- 0101 subtract
- 0110 decrement

I/0 OpCodes

- 1101 in
- 1110 out

Logic/Control OpCodes

- 0111 compare
- 1000 jump
- 1001 jumpgt
- 1010 jumpeq
- 1011 jumplt
- 1100 jumpneq

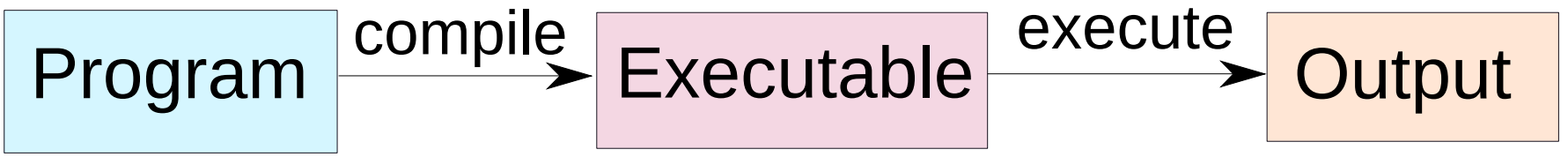


All a computer does is:

- Repeat forever (or until you pull the plug or the system crashes)
- 1) FETCH
- 2) DECODE
- 3) EXECUTE

- Notion of clockspeed

How to Instruct a Computer ?



* Write a program in a high level language – C, C++, Java
* **Compile** it into a format that the computer understands
* Execute the program

What Can a Computer Understand ?

- Computer can clearly NOT understand instructions of the form
  - Multiply two matrices
  - Compute the determinant of a matrix
  - Find the shortest path between Mumbai and Delhi
- They understand :
  - Add *a + b* to get *c*
  - Multiply *a + b* to get *c*

The Language of Instructions

- Humans can understand
  - Complicated sentences
    - English, French, Spanish
- Computers can understand
  - Very simple instructions

The semantics of all the instructions supported by a processor is known as its instruction set architecture (ISA). This includes the semantics of the instructions themselves, along with their operands, and interfaces with peripheral devices. Number of instructions in an x86 architecture?

Features of an ISA

- Example of instructions in an ISA
  - Arithmetic instructions : add, sub, mul, div
  - Logical instructions : and, or, not
  - Data transfer/movement instructions

- Also called low level instructions …
  - Most programmers avoid writing programs in low level instructions. Very complicated, and difficult.

Problems with programming using instruction sets directly

- Instruction sets of different types of CPUs are different
  - Need to write different programs for computers with different types of CPUs even to do the same thing
- **Solution**: High level languages (C, C++, Java,…)
  - CPU neutral, one program for many
  - Compiler to convert from high-level program to low level program that CPU understands
  - Easy to write programs

Fundamentals: Binary Numbers

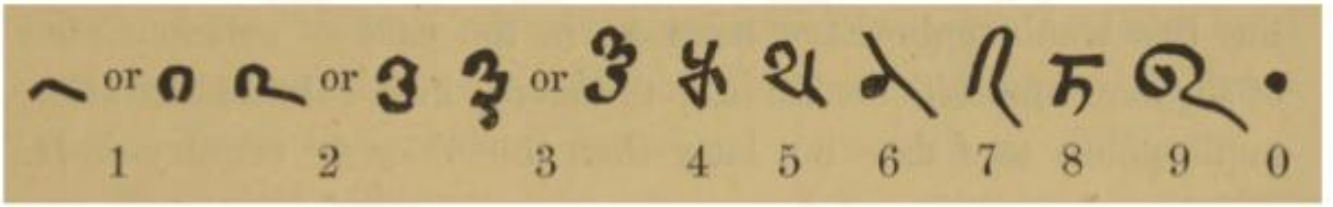
Representing Positive Integers

* Ancient Roman System

Symbol	I	V	X	L	C	D	M
Value	1	5	10	50	100	500	1000

* Issues :
  * There was no notion of 0
  * Very difficult to represent large numbers
  * Addition, and subtraction (very difficult)

Indian System



Bakshali numerals, 7th century AD

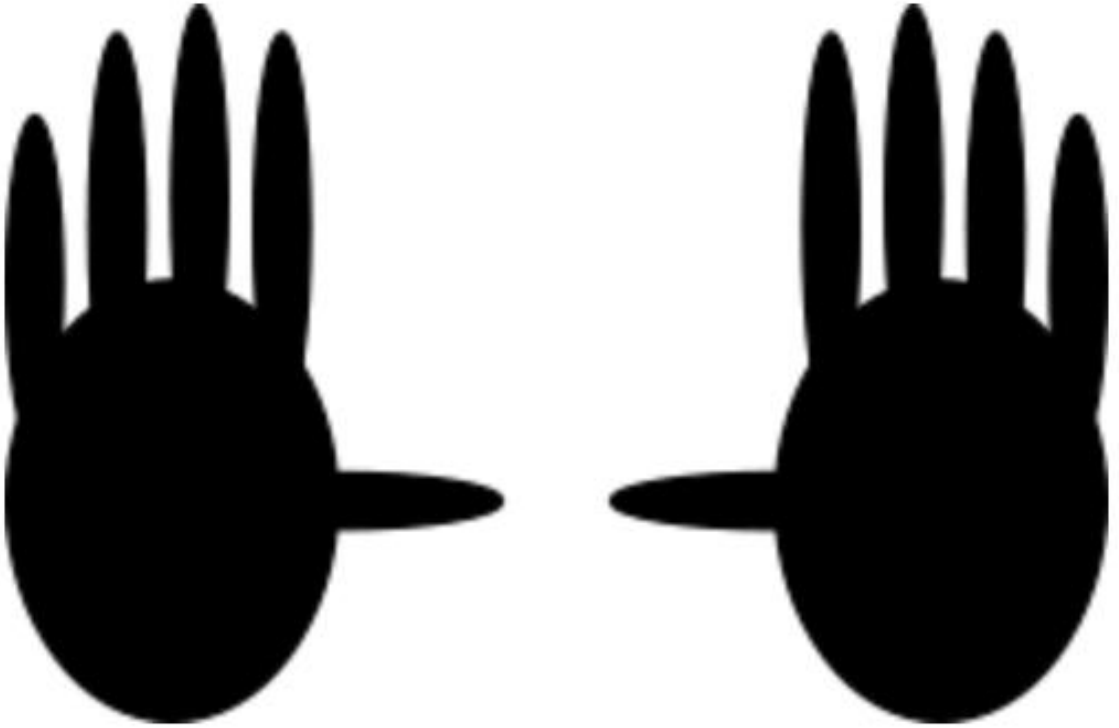
* Uses the place value system

$$5301 = 5 * 10^3 + 3 * 10^2 + 0 * 10^1 + 1*10^0$$

Example in base 10

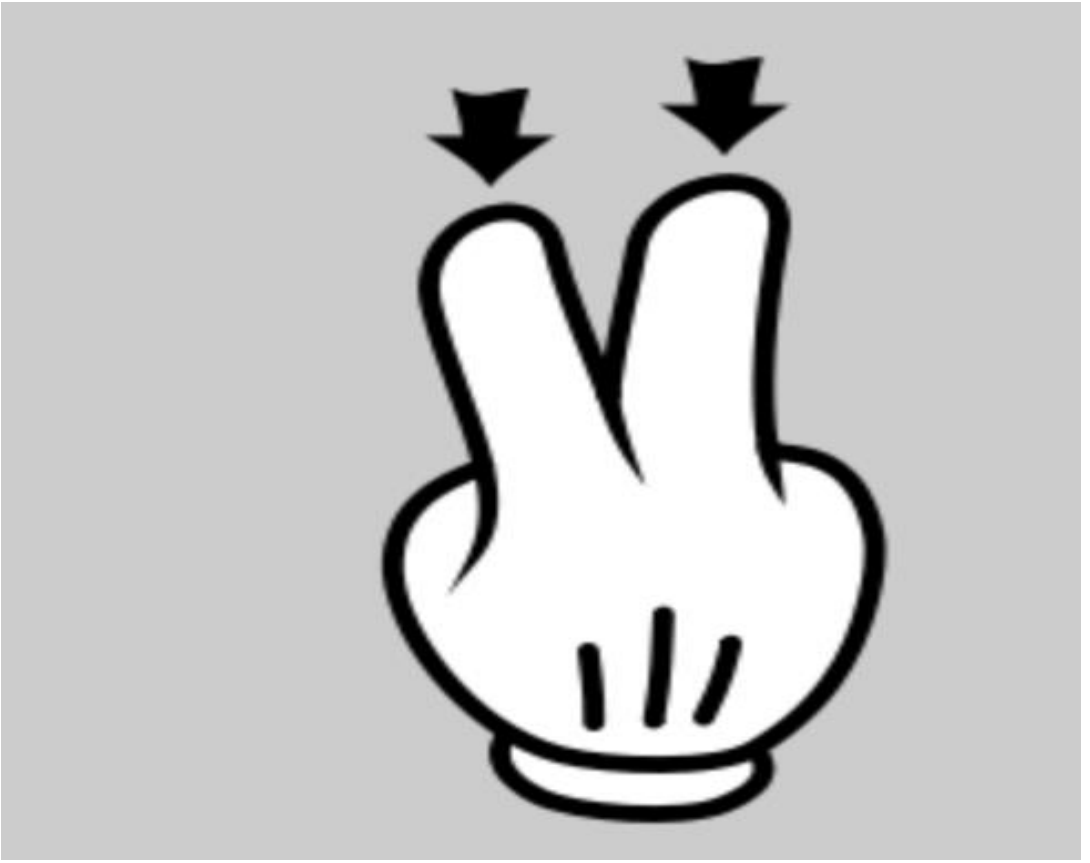
Number Systems in Other Bases

* Why do we use base 10 ?
  * because ...



What if we had a world in which …

* People had only two fingers.



Representing Numbers

$1023_{10} = 1 * 10^3 + 0 * 10^2 + 2 * 10^1 + 3 * 10^0$

Decimal

base

Binary

$17_{10} = 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 10001_2$

Base = 2

Binary Number System

* They would use a number system with base 2.

Number in decimal	Number in binary
5	101
100	1100100
500	111110100
1024	10000000000

MSB and LSB

* MSB (Most Significant Bit) → The leftmost bit of a binary number. E.g., MSB of 1110 is 1
* LSB (Least Significant Bit) → The rightmost bit of a binary number. E.g., LSB of 1110 is 0

Hexadecimal and Octal Numbers

- Hexadecimal numbers
  - Base 16 numbers – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - Start with 0x
- Octal Numbers
  - Base 8 numbers – 0,1,2,3,4,5,6,7
  - Start with 0

Examples

Decimal Numbers:

- **10 Symbols {0,1,2,3,4,5,6,7,8,9}, Base or Radix is 10**
- **$136 = 1 \times 10^2 + 3 \times 10^1 + 6 \times 10^0$**

Binary Numbers:

- **2 Symbols {0,1}, Base or Radix is 2**
- **$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$**

Octal Numbers:

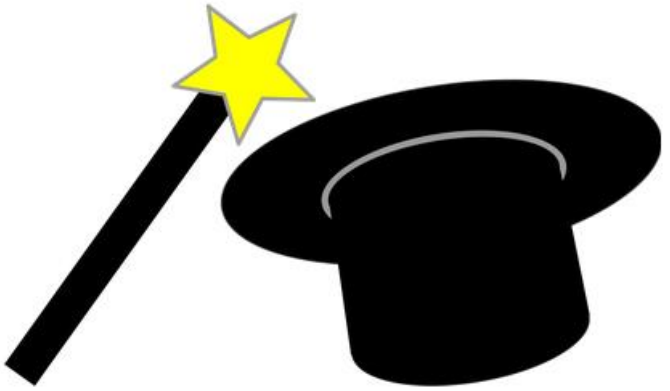
- **8 Symbols {0,1,2,3,4,5,6,7}, Base or Radix is 8**
- **$621 = 6 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$**

Hexadecimal Numbers:

- **16 Symbols {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, Base is 16**
- **$6AF = 6 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$**

Examples

Decimal	Binary	Octal	Hexadecimal
9	1001	0 11	0x 9
12	1100	0 14	0x C
17	10001	0 21	0x 11
28	11100	0 34	0x 1C



Convert 110010111 to the octal format : $\underbrace{110}\,\underbrace{010}\,\underbrace{111}$ = 0612

Convert 111000101111 to the hex format : $\underbrace{1110}\,\underbrace{0010}\,\underbrace{1111}$ = 0xC2F

Binary-to-Decimal Conversion

- Each digit position of a binary number has a weight
  - Some power of 2
- A binary number:

$$B = b_{n-1}\, b_{n-2} \ldots\ldots b_1\, b_0$$

Corresponding value in decimal:

$$D = \sum_{i=-m}^{n-1} b_i\, 2^i$$

Examples

$101011 \rightarrow 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$

$= 43$

$(101011)_2 = (43)_{10}$

$101 \rightarrow 1x2^2 + 0x2^1 + 1x2^0$

$= 5$

$(101)_2 = (5)_{10}$

Decimal to Binary: Integer Part

- Consider the integer and fractional parts separately.
- For the integer part:
  - Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders in reverse order.

Base	Numb	Rem
2	89	
2	44	1
2	22	0
2	11	0
2	5	1
2	2	1
2	1	0
	0	1

$(89)_{10} = (1011001)_2$

Decimal to Binary: Integer Part

- Consider the integer and fractional parts separately.
- For the integer part:
  - Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders in reverse order.

Base	Numb	Rem
2	89	
2	44	1
2	22	0
2	11	0
2	5	1
2	2	1
2	1	0
	0	1

$(89)_{10} = (1011001)_2$

2	66	
2	33	0
2	16	1
2	8	0
2	4	0
2	2	0
2	1	0
	0	1

$(66)_{10} = (1000010)_2$

Decimal to Binary: Integer Part

- Consider the integer and fractional parts separately.
- For the integer part:
  - Repeatedly divide the given number by 2, and go on accumulating the remainders, until the number becomes zero.
  - Arrange the remainders in reverse order.

Base	Numb	Rem
2	89	
2	44	1
2	22	0
2	11	0
2	5	1
2	2	1
2	1	0
	0	1

$(89)_{10} = (1011001)_2$

2	66	
2	33	0
2	16	1
2	8	0
2	4	0
2	2	0
2	1	0
	0	1

$(66)_{10} = (1000010)_2$

2	239	
2	119	1
2	59	1
2	29	1
2	14	1
2	7	0
2	3	1
2	1	1
	0	1

$(239)_{10} = (11101111)_2$

Hexadecimal Number System

- A compact way of representing binary numbers
- 16 different symbols (radix = 16)

0 →	0000	8 →	1000
1 →	0001	9 →	1001
2 →	0010	A →	1010
3 →	0011	B →	1011
4 →	0100	C →	1100
5 →	0101	D →	1101
6 →	0110	E →	1110
7 →	0111	F →	1111

Binary-to-Hexadecimal Conversion

- For the integer part,
  - Scan the binary number from right to left
  - Translate each group of four bits into the corresponding hexadecimal digit
    - Add leading zeros if necessary

Example

1. $(\underline{1011}\ \underline{0100}\ \underline{0011})_2 = (B43)_{16}$
2. $(\underline{10}\ \underline{1010}\ \underline{0001})_2 = (2A1)_{16}$

Hexadecimal-to-Binary Conversion

- Translate every hexadecimal digit into its 4-bit binary equivalent

- Examples:

  $(3A5)_{16} = (0011\ 1010\ 0101)_2$

  $(12)_{16} = (0001\ 0010)_2$

  $(1)_{16} = (0001)_2$

Bits and Bytes

* Computers do not understand natural human languages, nor programming languages
* They only understand the language of bits

Bit	0 or 1
Byte	8 bits
Word	4 bytes
kiloByte	1024 bytes
megaByte	10^6 bytes

77

Fundamentals of C

First C program – print on screen

```
#include <stdio.h>
void main()
{
    printf ("Hello, World! \n") ;
}
```

More printing … (code and see)

```
#include <stdio.h>
void main()
{
    printf ("Hello, World! ") ;
    printf ("Hello \n World! \n") ;
}
```

Some more printing

```
#include <stdio.h>
void main()
{
    printf ("Hello, World! \n") ;
    printf ("Hello \n World! \n") ;
    printf ("Hell\no \t World! \n") ;
}
```

Reading values from keyboard

```
#include <stdio.h>
void main()
{
        int num ;
        scanf ("%d", &num) ;
        printf (“No. of students is %d\n”, num) ;
}
```

Centigrade to Fahrenheit

```
#include <stdio.h>
void main()
{
    float cent, fahr;
    scanf("%f",&cent);
    fahr = cent*(9.0/5.0) + 32;
    printf( "%f C equals %f F\n", cent, fahr);
}
```

Largest of two numbers

```
#include <stdio.h>
void main()
{
        int x, y;
        scanf("%d%d",&x,&y);
        if (x>y) printf("Largest is %d\n",x);
        else printf("Largest is %d\n",y);
}
```

What does this do?

```
#include <stdio.h>
void main()
{
        int x, y;
        scanf("%d%d",&x,&y);
        if (x>y) printf("Largest is %d\n",x);
        printf("Largest is %d\n",y);
}
```

largest-2.c

The C Character Set

- The C language alphabet
  - Uppercase letters ‘A’ to ‘Z’
  - Lowercase letters ‘a’ to ‘z’
  - Digits ‘0’ to ‘9’
  - Certain special characters:

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
|	;	:	‘	“	{	}	,
.	<	>	/	?	blank		

A C program should not contain anything else

Structure of a C program

- A collection of functions (we will see what they are later)
- Exactly one special function named main must be present. Program always starts from there
- Each function has statements (instructions) for declaration, assignment, condition check, looping etc.
- Statements are executed one by one

Variables

- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program

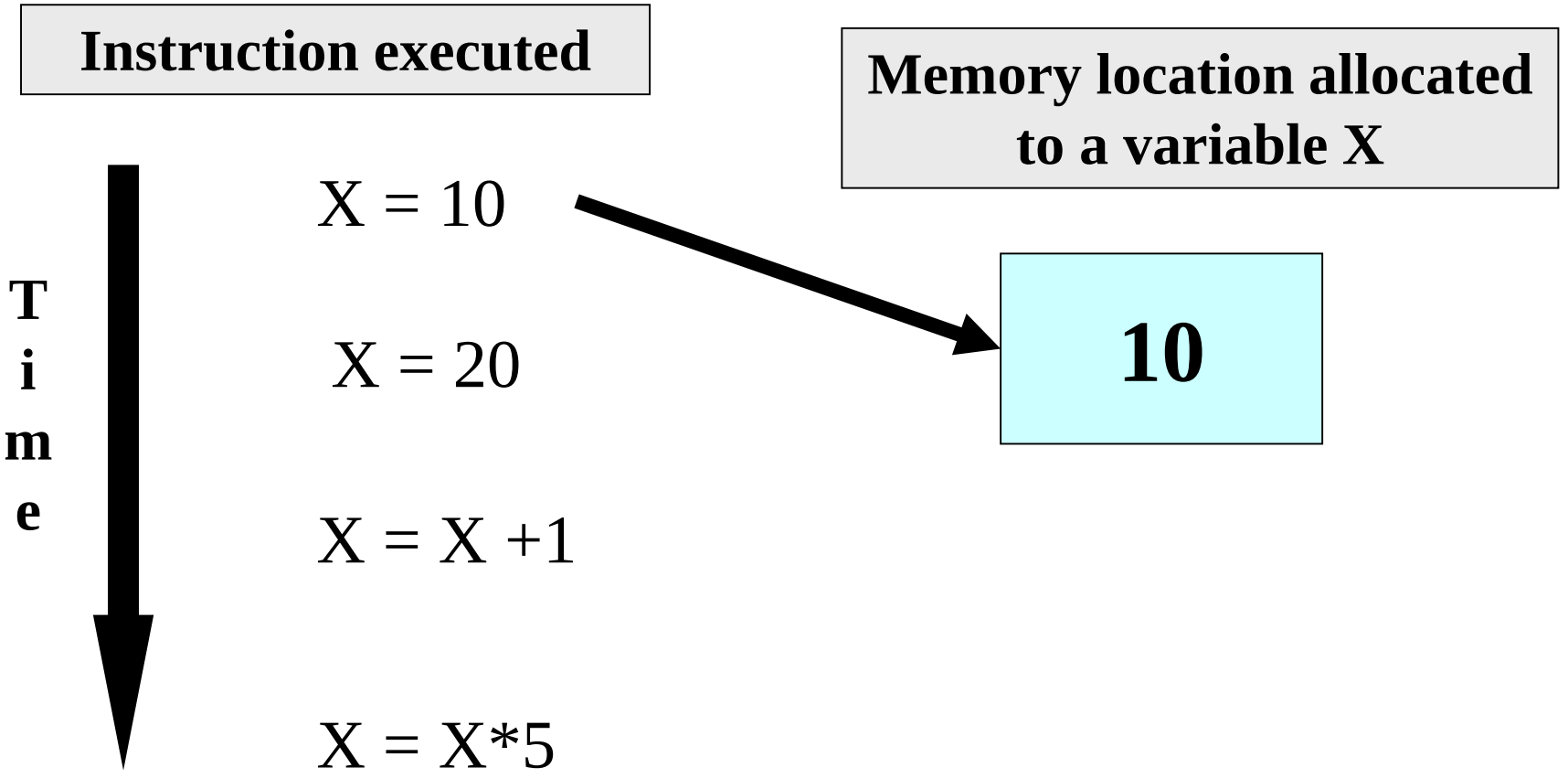
Contd.

- Variables stored in memory
- Remember that memory is a list of storage locations, each having a unique address
- A variable is like a bin
  - The contents of the bin is the value of the variable
  - The variable name is used to refer to the value of the variable
  - A variable is mapped to a location of the memory, called its address

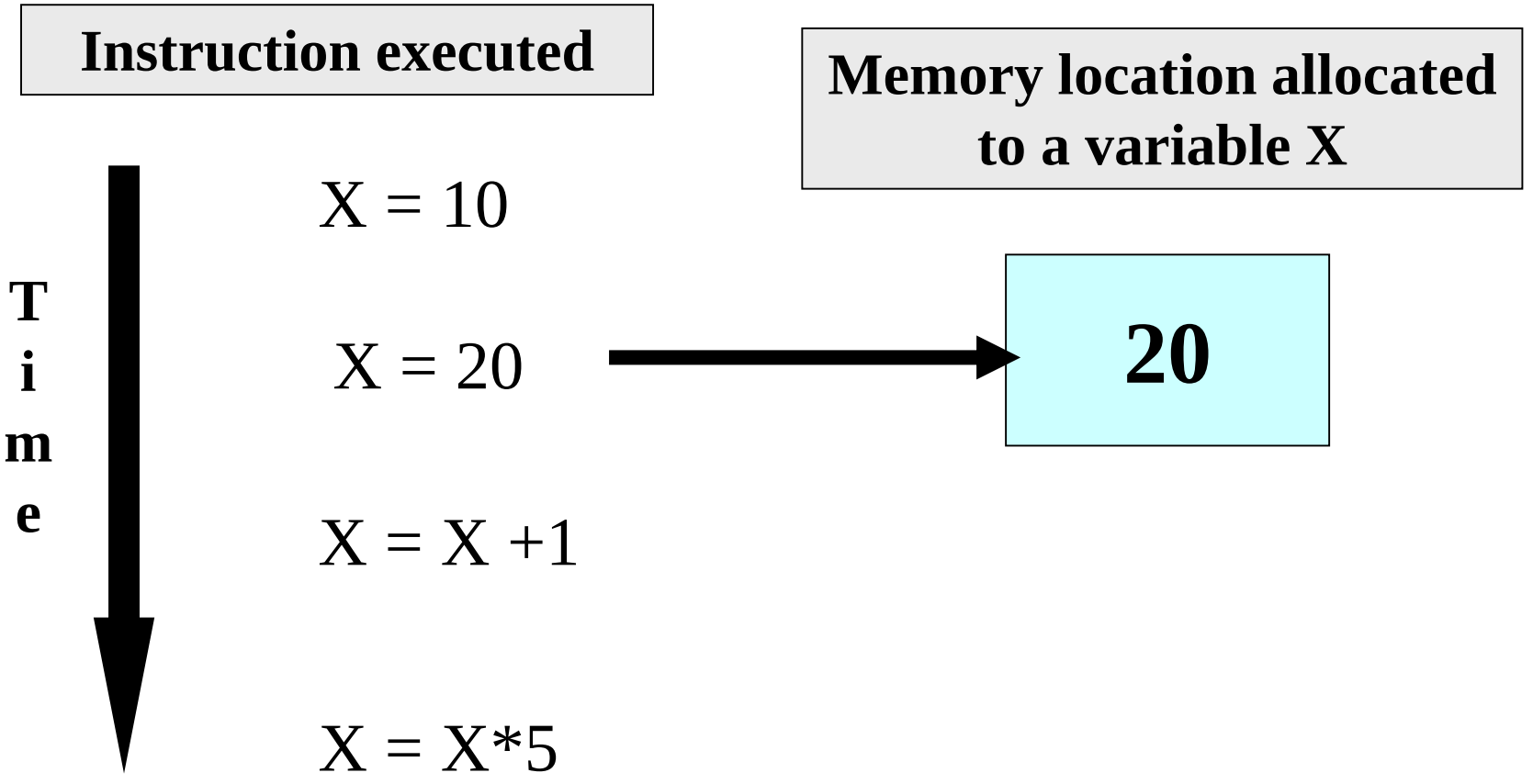
Example

```
#include <stdio.h>
void main( )
{
        int x;
        int y;
        x=1;
        y=3;
        printf("x = %d, y= %d\n", x, y);
}
```

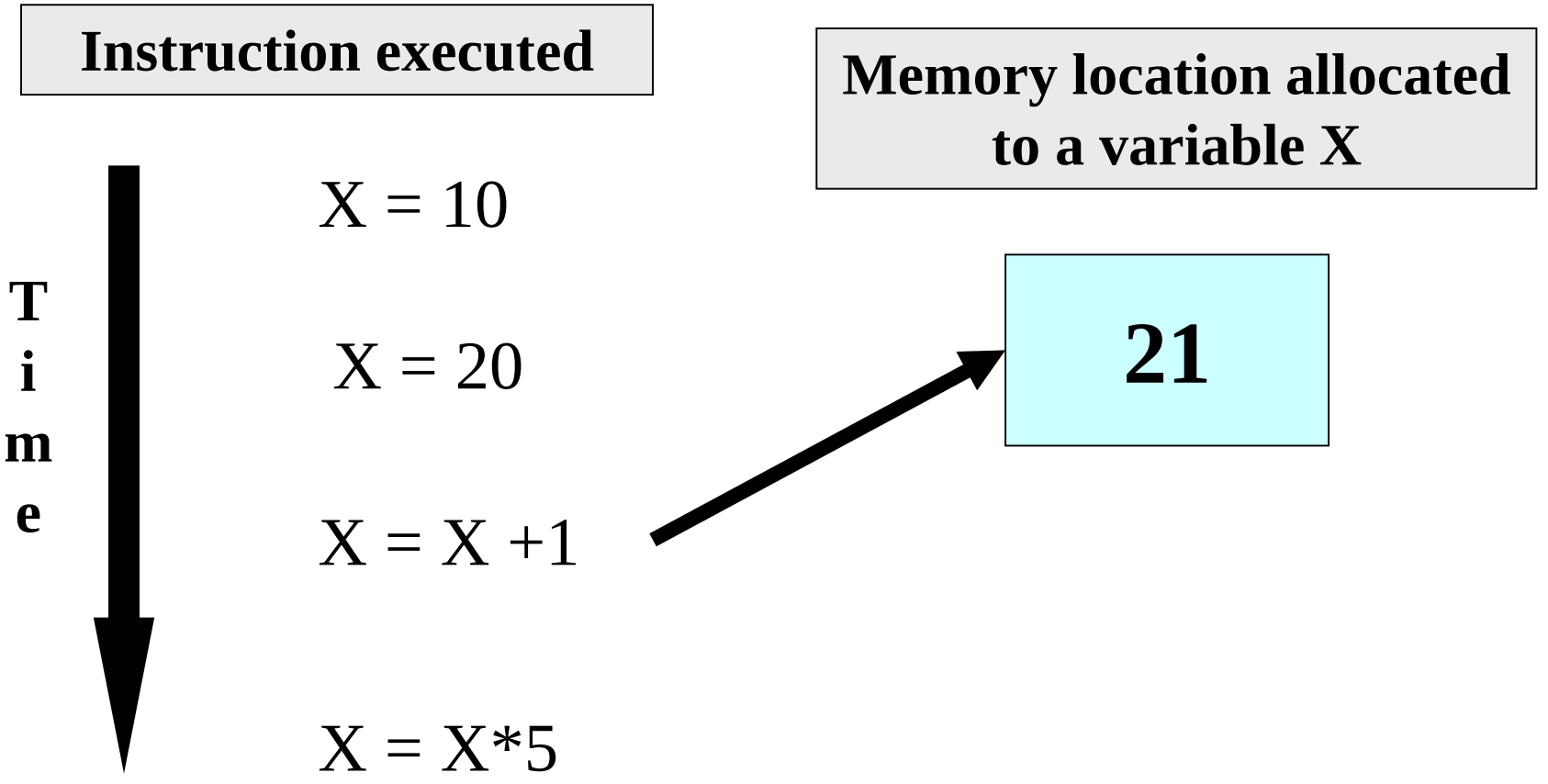
Variables in Memory



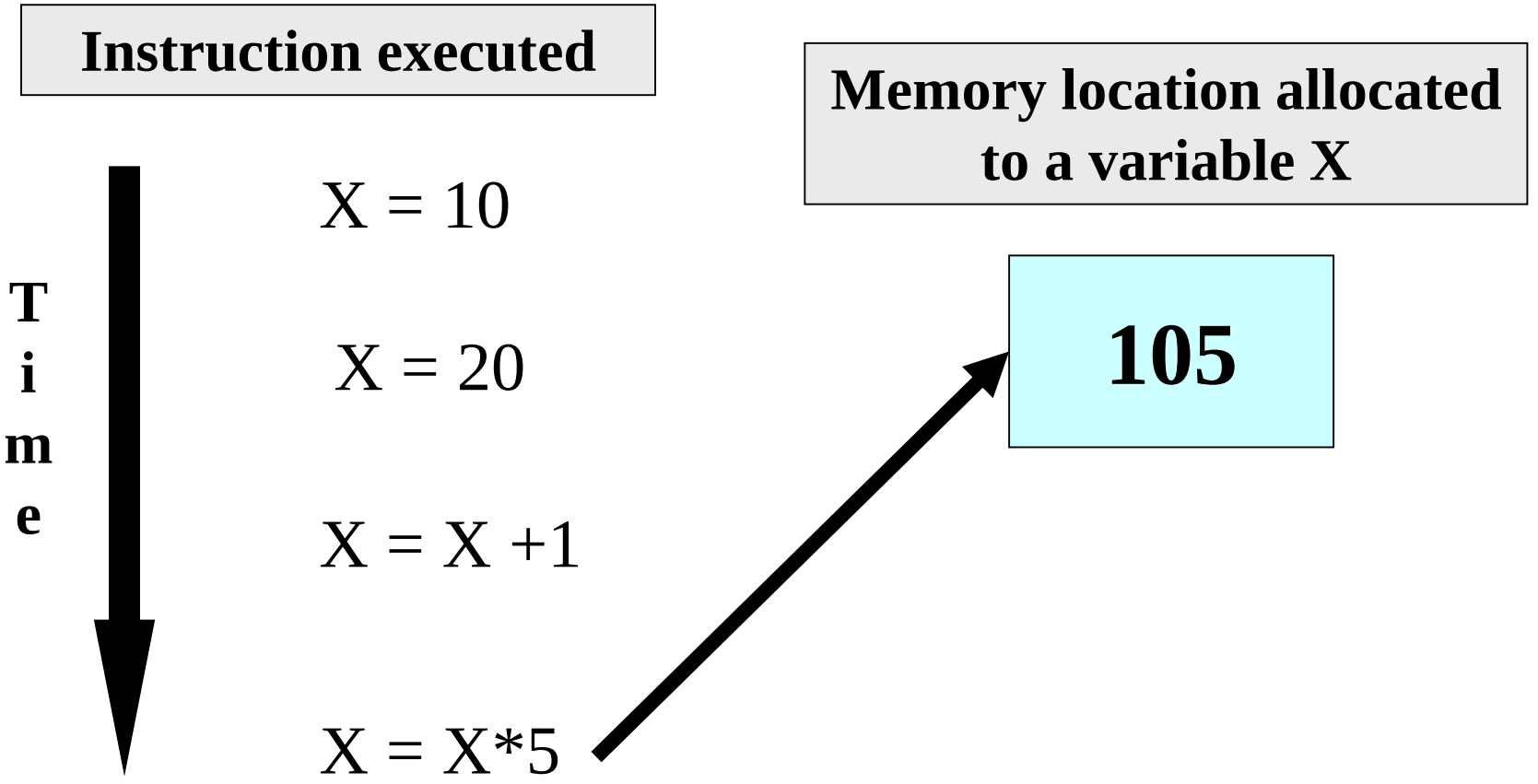
Variables in Memory



Variables in Memory



Variables in Memory



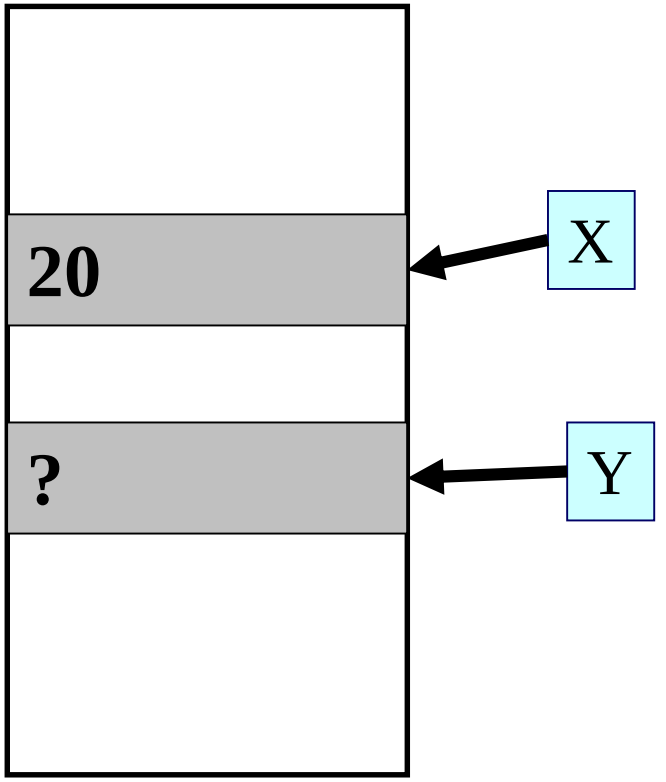
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



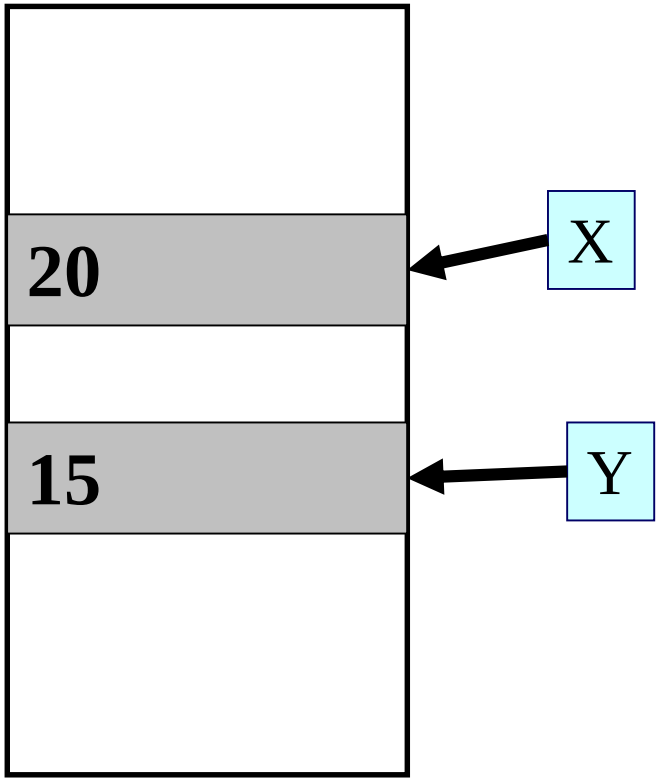
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



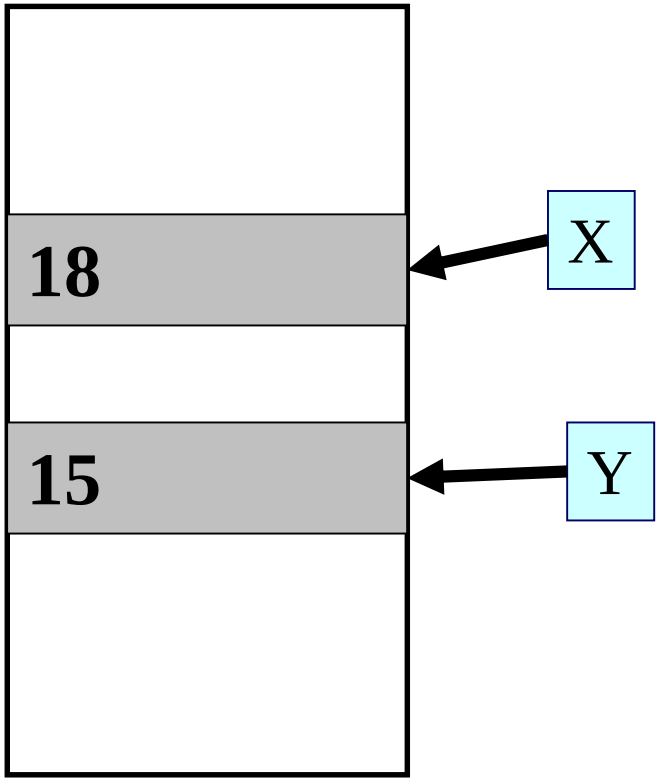
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



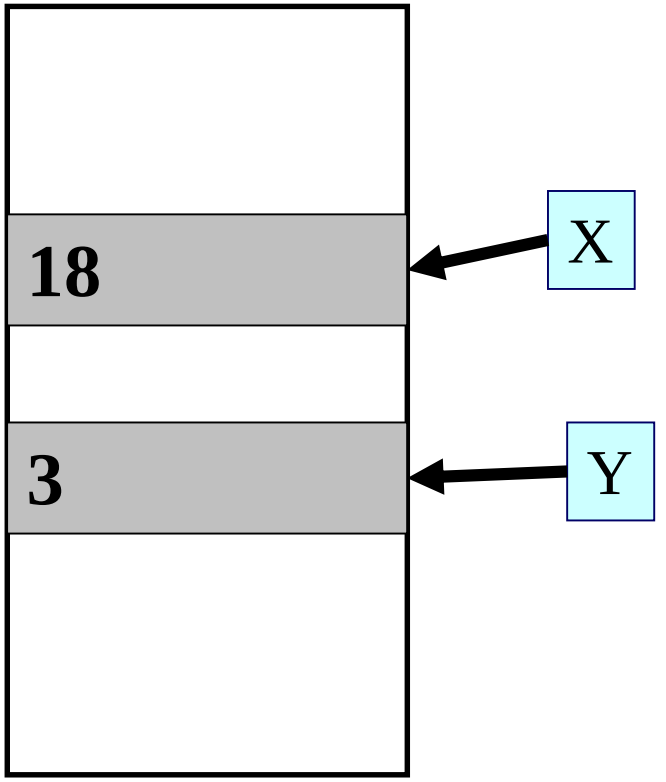
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



Data Types

- Each variable has a type, indicates what type of values the variable can hold
- Four common data types in C
  - int - can store integers (usually 4 bytes)
  - float - can store single-precision floating point numbers (usually 4 bytes)
  - double - can store double-precision floating point numbers (usually 8 bytes)
  - char - can store a character (1 byte)

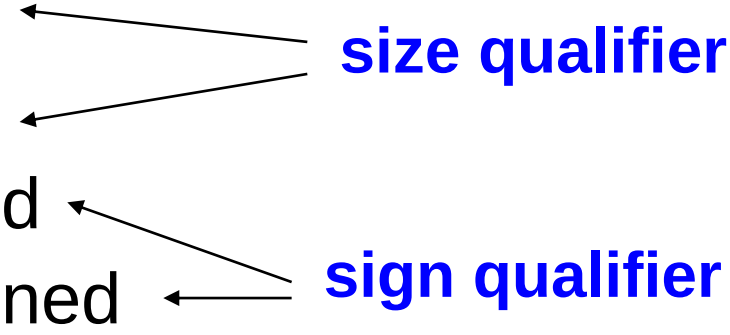
Contd.

- Must declare a variable (specify its type and name) before using it anywhere in your program
- All variable declarations should be at the beginning of the main() or other functions
- A value can also be assigned to a variable at the time the variable is declared.

```
int   speed = 30;
char  flag = ‘y’;
```

More Data Types in C

- Some of the basic data types can be augmented by using certain data type qualifiers:
  - short
  - long
  - signed
  - unsigned



- Typical examples:
  - short int (usually 2 bytes)
  - long int (usually 4 bytes)
  - unsigned int (usually 4 bytes, but no way to store + or -)

Some typical sizes (some of these can vary depending on type of machine)

Integer data type	Bit size	Minimum value	Maximum value
char	8	$-2^7=-128$	$2^7-1=127$
short int	16	$-2^{15}=-32768$	$2^{15}-1=32767$
int	32	$-2^{31}=-2147483648$	$2^{31}-1=2147483647$
long int	32	$-2^{31}=-2147483648$	$2^{31}-1=2147483647$
long long int	64	$-2^{63}=-9223372036854775808$	$2^{63}-1=9223372036854775807$
unsigned char	8	0	$2^8-1=255$
unsigned short int	16	0	$2^{16}-1=65535$
unsigned int	32	0	$2^{32}-1=4294967295$
unsigned long int	32	0	$2^{32}-1=4294967295$
unsigned long long int	64	0	$2^{64}-1=18446744073709551615$

Variable Names

- Sequence of letters and digits
- First character must be a letter or '_'
- No special characters other than '_'
- No blank in between
- Names are case-sensitive (max and Max are two different names)
- Examples of valid names:
  - i rank1 MAX max Min class_rank
- Examples of invalid names:
  - a's fact rec 2sqroot class,rank

More Valid and Invalid Identifiers

- Valid identifiers

  X

  abc

  simple_interest

  a123

  LIST

  stud_name

  Empl_1

  Empl_2

  avg_empl_salary

- Invalid identifiers

  10abc

  my-name

  “hello”

  simple interest

  (area)

  %rate

C Keywords

- Used by the C language, cannot be used as variable names
- Examples:
  - int, float, char, double, main, if else, for, while. do, struct, union, typedef, enum, void, return, signed, unsigned, case, break, sizeof,….
  - There are others, see textbook…

Example 1

```
#include <stdio.h>
void main()
{
    int x, y, sum;
    scanf("%d%d",&x,&y);
    sum = x + y;
    printf( "%d plus %d is %d\n", x, y, sum );
}
```

Three int type variables declared

Values assigned

Example - 2

```
#include <stdio.h>
void main()
{
        float x, y;
        int d1, d2 = 10;
        scanf("%f%f%d",&x, &y, &d1);
        printf( "%f plus %f is %f\n", x, y, x+y);
        printf( "%d minus %d is %d\n", d1, d2, d1-d2);
}
```

Assigns an initial value to d2, can be changed later

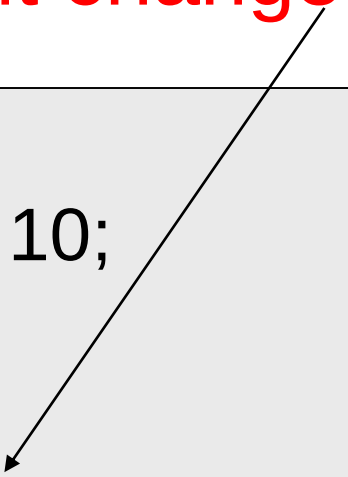
Read-only variables

- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the const keyword in front of the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed

Correct

```
void main() {
   const int LIMIT = 10;
   int n;
   scanf("%d", &n);
   if (n > LIMIT)
      printf("Out of limit");
}
```

Incorrect: Limit changed



```
void main() {
    const int Limit = 10;
    int n;
    scanf("%d", &n);
    Limit = Limit + n;
    printf("New  limit is %d", Limit);
}
```

Constants

- Integer constants
  - Consists of a sequence of digits, with possibly a plus or a minus sign before it
  - Embedded spaces, commas and non-digit characters are not permitted between digits
- Floating point constants
- Two different notations:
  - Decimal notation: 25.0, 0.0034, .84, -2.234
  - Exponential (scientific) notation

    3.45e23, 0.123e-12, 123e2

e means “10 to the power of”

Contd.

- Character constants
  - Contains a single character enclosed within a pair of single quote marks.
  - Examples :: ‘2’, ‘+’, ‘Z’
- Some special backslash characters

‘\n’	new line
‘\t’	horizontal tab
‘\’’	single quote
‘\”’	double quote
‘\\’	backslash
‘\0’	null

Input: scanf function

- Performs input from keyboard
- It requires a format string and a list of variables into which the value received from the keyboard will be stored
- format string = individual groups of characters (usually '%' sign, followed by a conversion character), with one character group for each variable in the list

```
int a, b;
float c;
scanf("%d %d %f", &a, &b, &c);
```

Variable list (note the & before a variable name)

Format string

- Commonly used conversion characters
  - c for char type variable
  - d for int type variable
  - f for float type variable
  - lf for double type variable

- Examples

```
scanf ("%d", &size) ;
scanf ("%c", &nextchar) ;
scanf ("%f", &length) ;
scanf (“%d%d”, &a, &b);
```

Reading a single character

- A single character can be read using scanf with %c
- It can also be read using the getchar() function

```
char c;
c = getchar();
```

- Program waits at the getchar() line until a character is typed, and then reads it and stores it in c

Output: printf function

- Performs output to the standard output device (typically defined to be the screen)
- It requires a format string in which we can specify:
  - The text to be printed out
  - Specifications on how to print the values

```
printf ("The number is %d\n", num);
```

  - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d
  - Output will appear as: The number is 125

Contd.

- General syntax:

  printf (format string, arg1, arg2, …, argn);

  - format string refers to a string containing formatting information and data types of the arguments to be output
  - the arguments arg1, arg2, … represent list of variables/expressions whose values are to be printed
- The conversion characters are the same as in scanf

- Examples:

```
printf  (“Average of %d and %d is %f”, a, b, avg);
printf  (“Hello \nGood \nMorning \n”);
printf (“%3d %3d %5d”, a, b, a*b+2);
printf (“%7.2f  %5.1f”, x, y);
```

- Many more options are available for both printf and scanf
  - Read from the book
  - Practice them in the lab

Acknowledgements

- Slides adapted from several sources:
  - http://www.cs.tulane.edu/~carola/teaching/cmps1500/fall13/slides/
  - http://www.cs.kent.edu/~jbaker/CS10051-Sp09/
  - http://www.computersciencelab.com/ComputerHistory/History.htm
  - https://cse.iitkgp.ac.in/~ppd/