CENG 230

Introduction to C Programming

Week 1 – Overview and introduction

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli.

Syllabus

How to study?

- Follow the lectures and the labs
- Read the textbook on a weekly basis
- Get your hands dirty
    - Do the exercises in front of the computer

Appointment

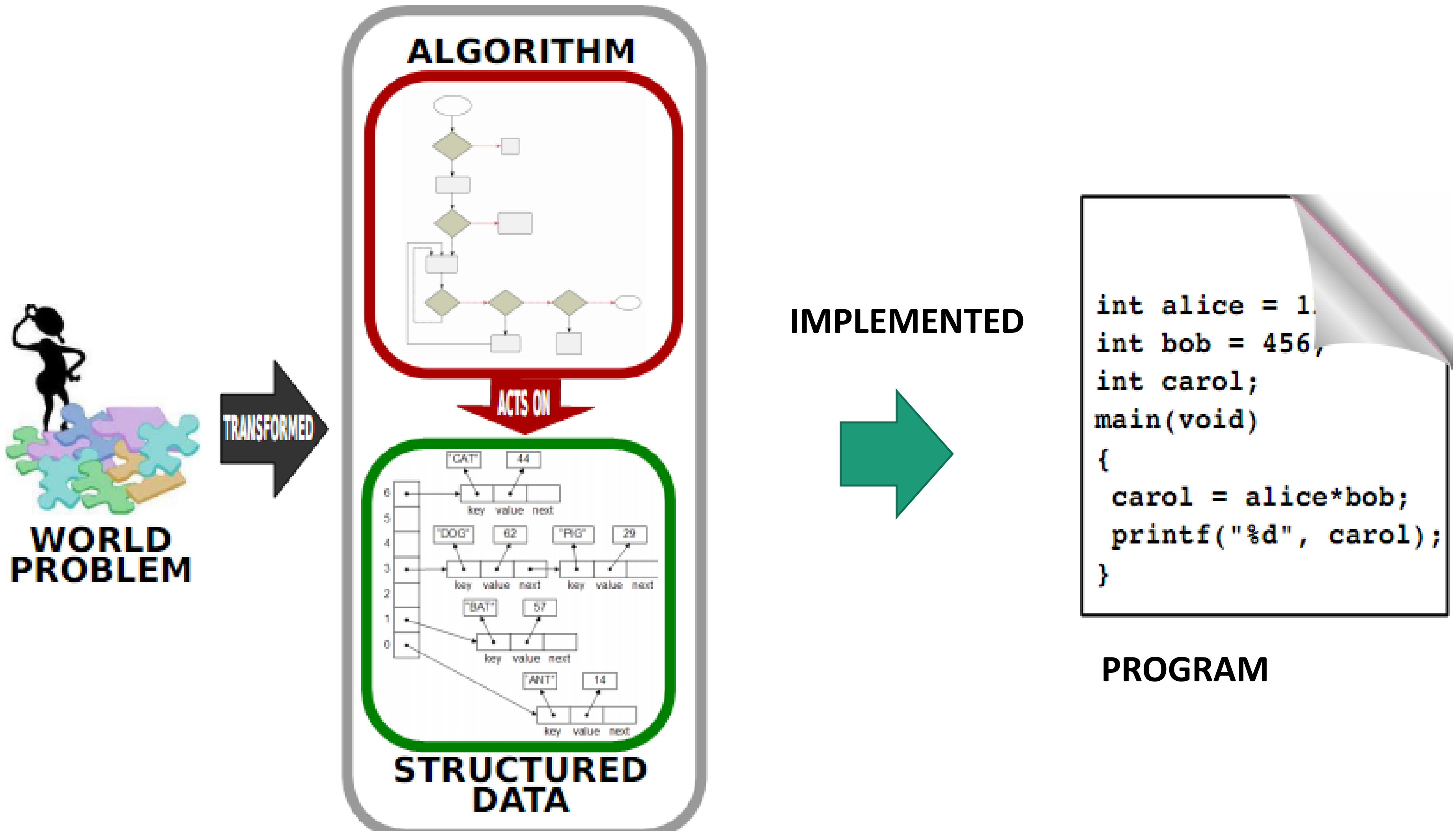
- No office hours. Make an appointment.
- Via email: skalkan@ceng.metu.edu.tr
- Office:

  Room B207,

  Department of Computer Engineering
- WWW:
  - http://kovan.ceng.metu.edu.tr/~sinan/

Programming, computation, algorithm

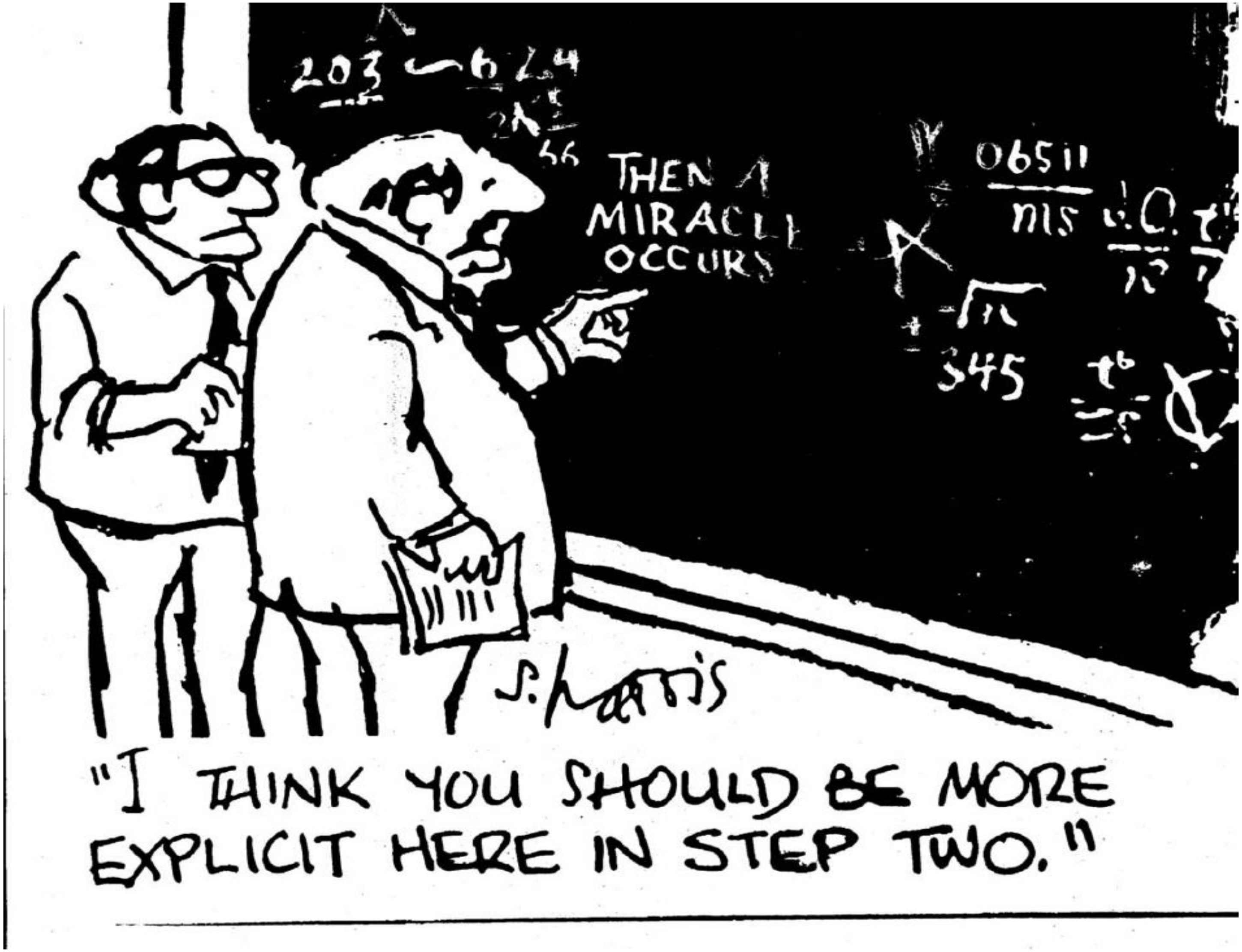
Program, Programming



What is an algorithm?

- An algorithm is a list that looks like
  - STEP 1: Do something
  - STEP 2: Do something
  - STEP 3: Do something
  - . .
  - . .
  - . .
  - STEP N: Stop, you are finished

From “Invitation to Computer Science”



From “Invitation to Computer Science”

A formal definition of algorithm

- “Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, will proceed through a finite number of well-defined successive states, eventually producing "output" and terminating at a final ending state.”

Algorithms

- We use them all the time.
- Can you give examples?
  - Following directions
  - Recording a DVD
  - Adding two numbers
  - Finding Greatest Common Divisor
  - …

From "Invitation to Computer Science"

An example algorithm

Algorithm for Adding Two m-Digit Numbers

Given: $m \geq 1$ and two positive numbers each containing m digits, $a_{m-1}\ a_{m-2,} \ldots a_0$ and $b_{m-1}\ b_{m-2,} \ldots b_0$
Wanted: $c_m c_{m-1}\ c_{m-2} \ldots c_0$, where $c_m c_{m-1}\ c_{m-2} \ldots c_0 = (a_{m-1}\ a_{m-2} \ldots a_0) + (b_{m-1}\ b_{m-2} \ldots b_0)$

Algorithm:

Step 1 Set the value of *carry* to 0.
Step 2 Set the value of *i* to 0.
Step 3 While the value of *i* is less than or equal to $m - 1$, repeat the instructions in steps 4 through 6.
Step 4 Add the two digits a_i and b_i to the current value of *carry* to get c_i.
Step 5 If $c_i \geq 10$, then reset c_i to $(c_i - 10)$ and reset the value of *carry* to 1; otherwise, set the new value of *carry* to 0.
Step 6 Add 1 to *i*, effectively moving one column to the left.
Step 7 Set c_m to the value of *carry*.
Step 8 Print out the final answer, $c_m\ c_{m-1}\ c_{m-2} \ldots c_0$.
Step 9 Stop.

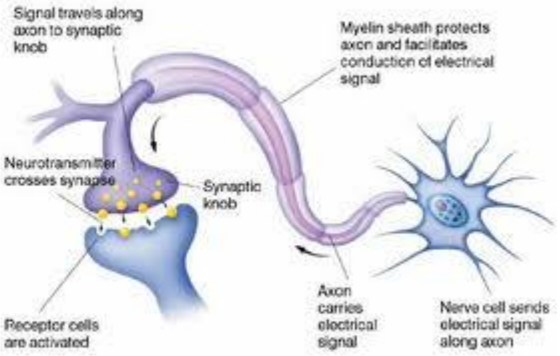
From "Invitation to Computer Science"

“Computation”

- Digital vs. analog computation
- Sequential vs. parallel computation
- Batch vs. interactive computation
- Evolutionary, molecular, quantum computation
- “Physical computation” / “Digital Physics”
  - ‘The whole universe is itself a computation’

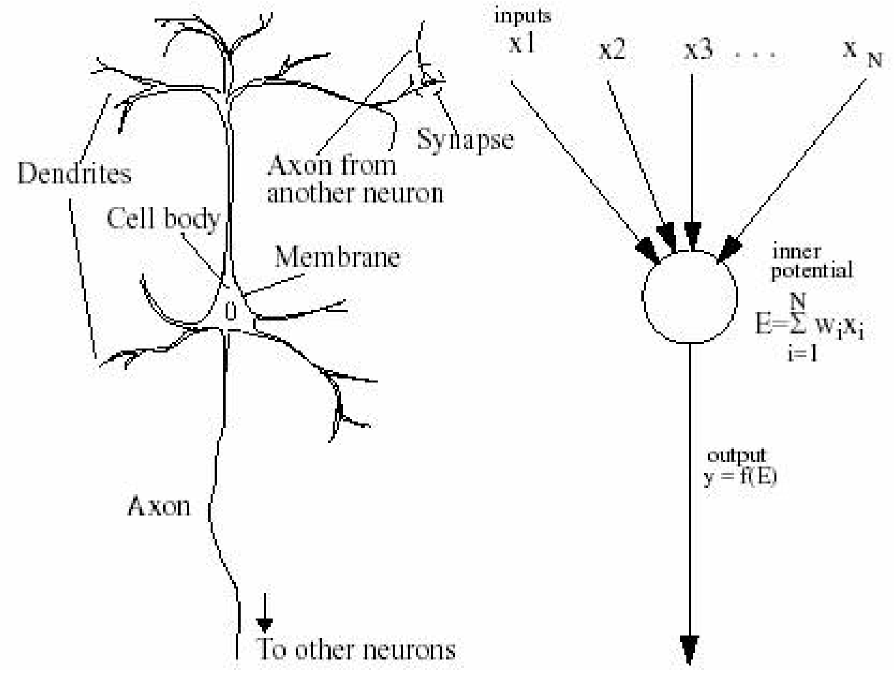
Computation in our brain

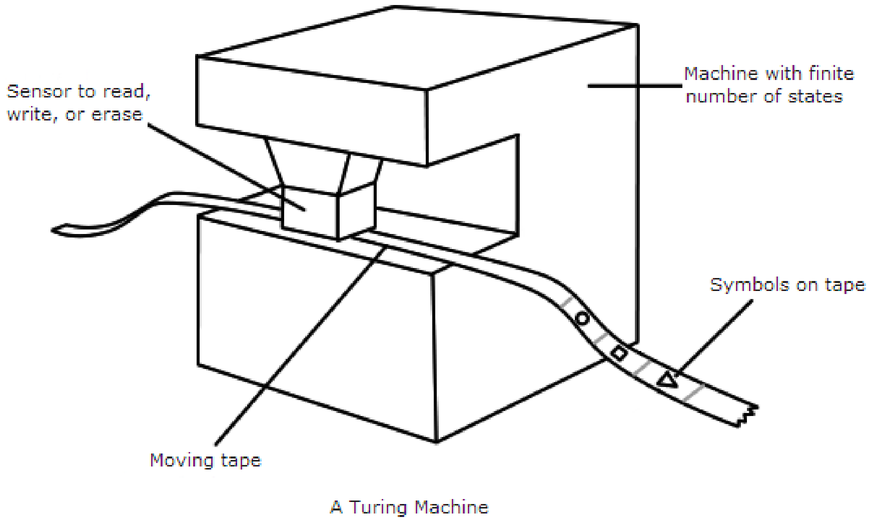
- Highly-connected network of neurons.
- How many neurons?
  - Approx. 10^{11} neurons and 10^{14} synapses.
- How do they transmit information?
  - Using nothing else than charged molecules.



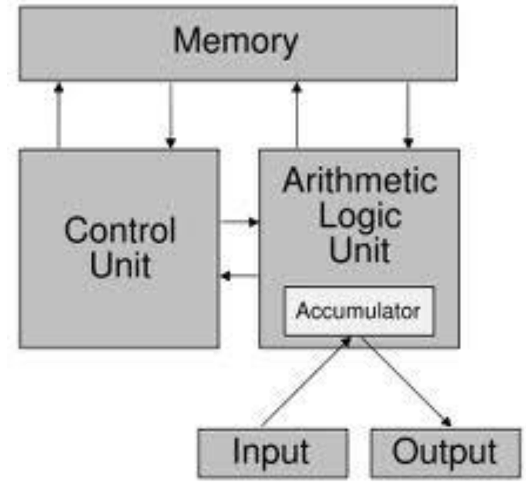
Computation in our brain (cont’d)

- Each neuron gets input and produces an output using an “activation function”





Turing Machine



Von Neumann Architecture

DIGITAL COMPUTATION

But first some historical overview

The Early Period: Up to 1940

- 3,000 years ago: Mathematics, logic, and numerical computation
  - Important contributions made by the Greeks, Egyptians, Babylonians, Indians, Chinese, and Persians
  - Cuneiform
  - Stone “abacus”

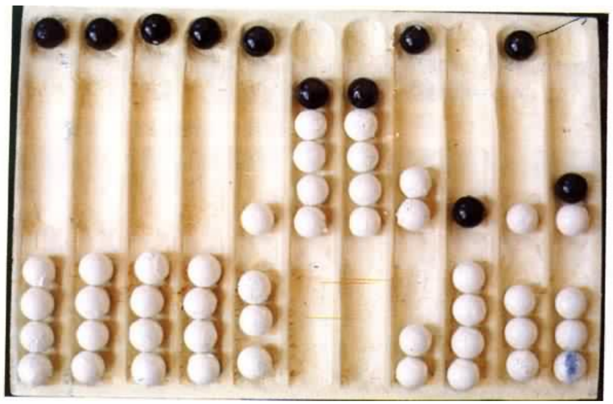
- http://www.thocp.net/slideshow/0469.htm

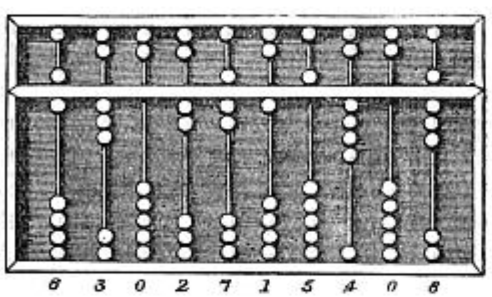
Slide from “Introduction to Computing”

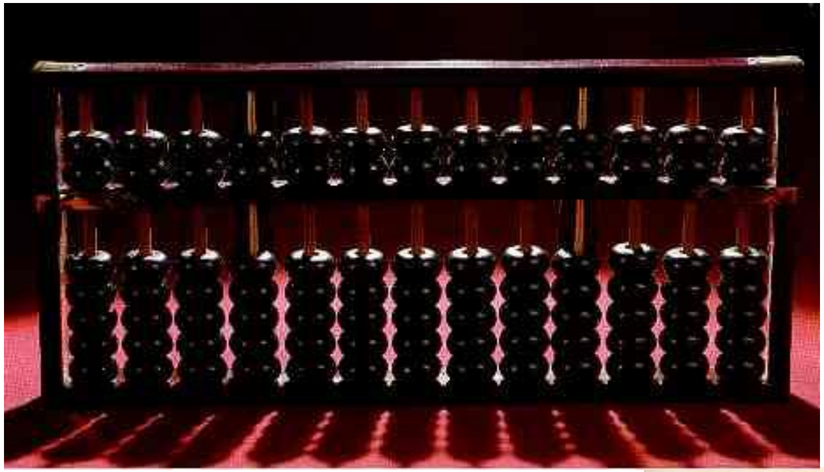
ABACUS

Early calculating devices

ABACUS – 2700 BC (Mesopotamia)





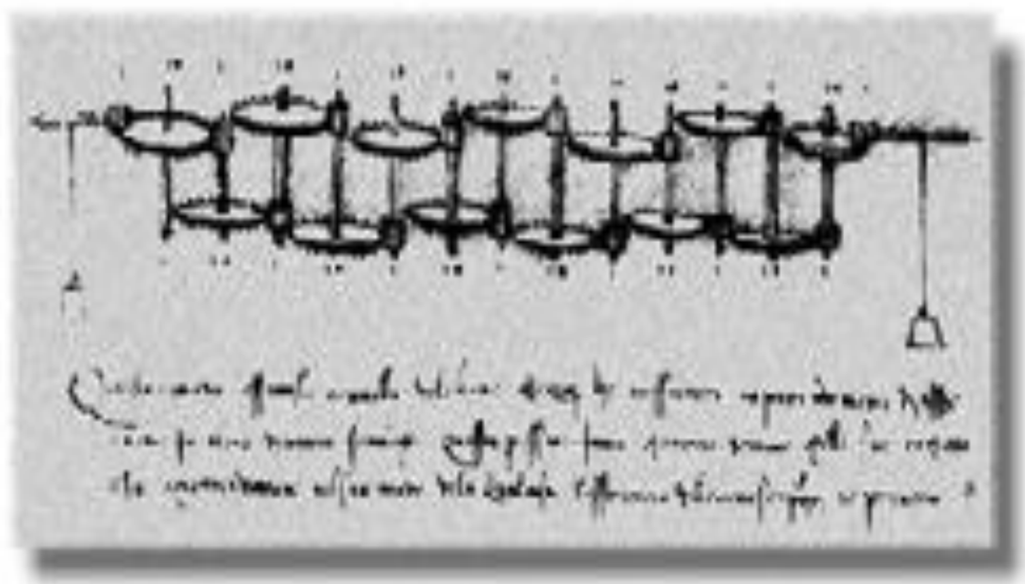


http://www.computersciencelab.com/ComputerHistory/History.htm

Slide from "Introduction to Computing"

DaVinci

- 1452-1519 Leonardo DaVinci sketched gear-driven calculating machines but none were ever built.



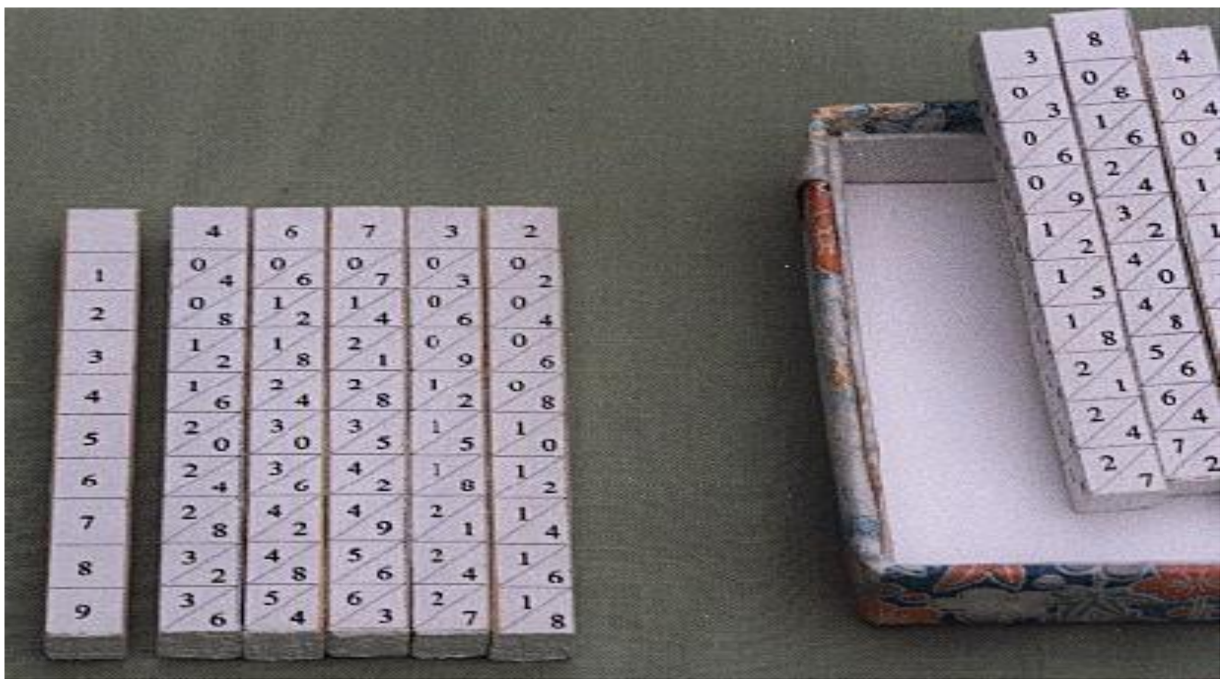
http://www.computersciencelab.com/ComputerHistory/History.htm

Slide from "Introduction to Computing"

Napier’s Bones

- 1614: Logarithms
  - Invented by John Napier to simplify difficult mathematical computations

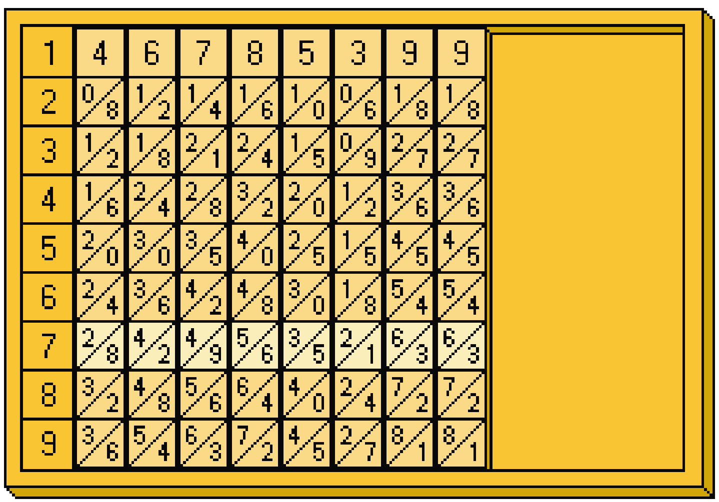
Napier’s Bones:

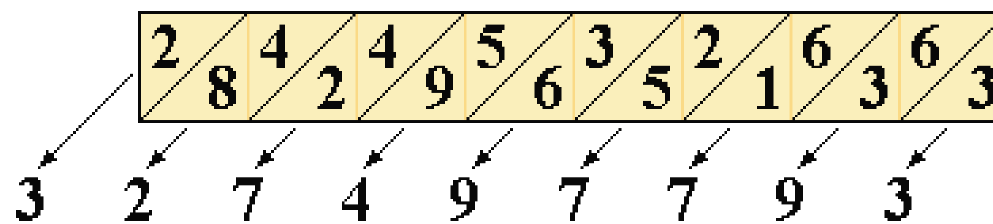


http://www.computersciencelab.com/ComputerHistory/History.htm

Slide from “Introduction to Computing”

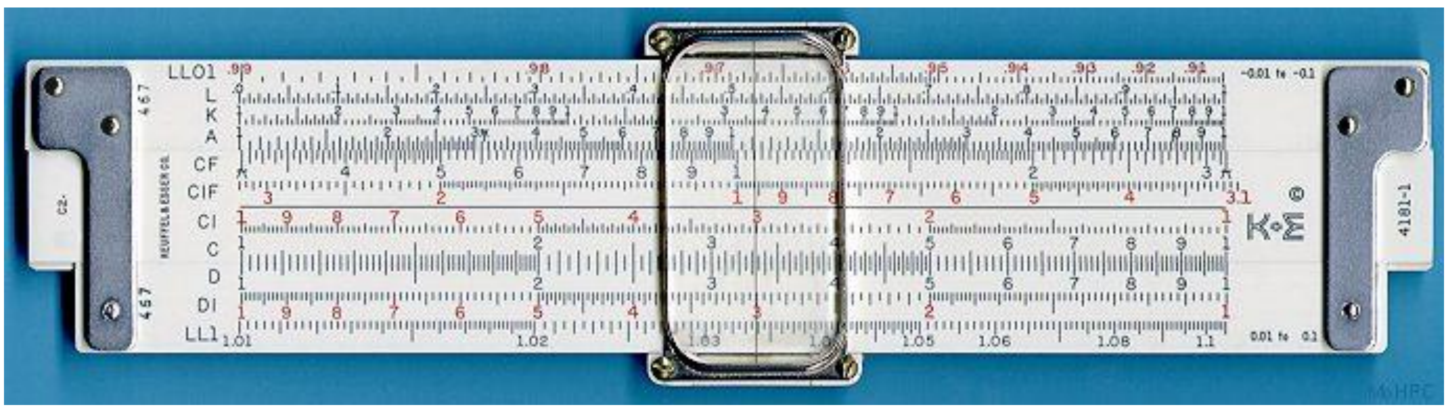
- If you want to multiply 7 by 46785499:





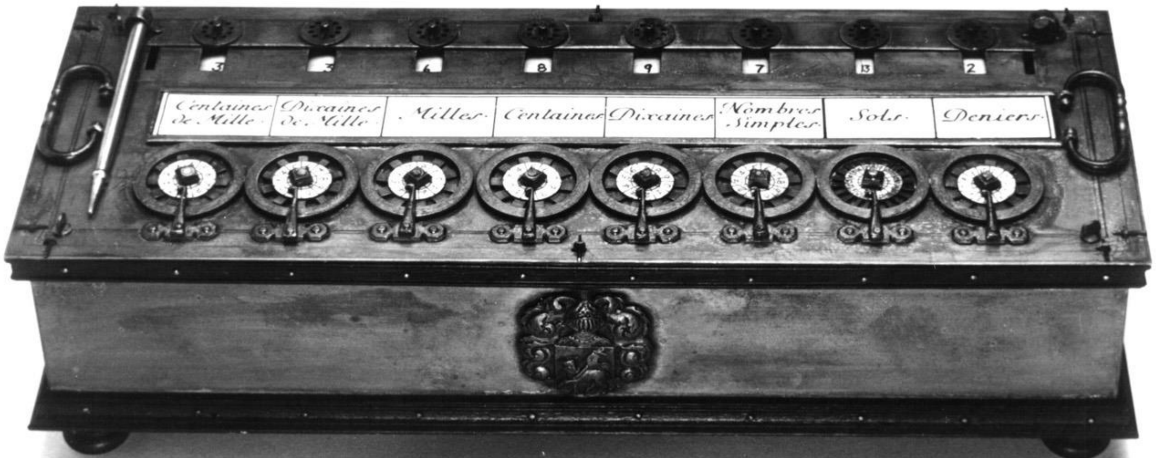
Slide Rule (slipstick)
"a mechanical analog computer"

Around 1622: First slide rule created



http://www.computersciencelab.com/ComputerHistory/History.htm

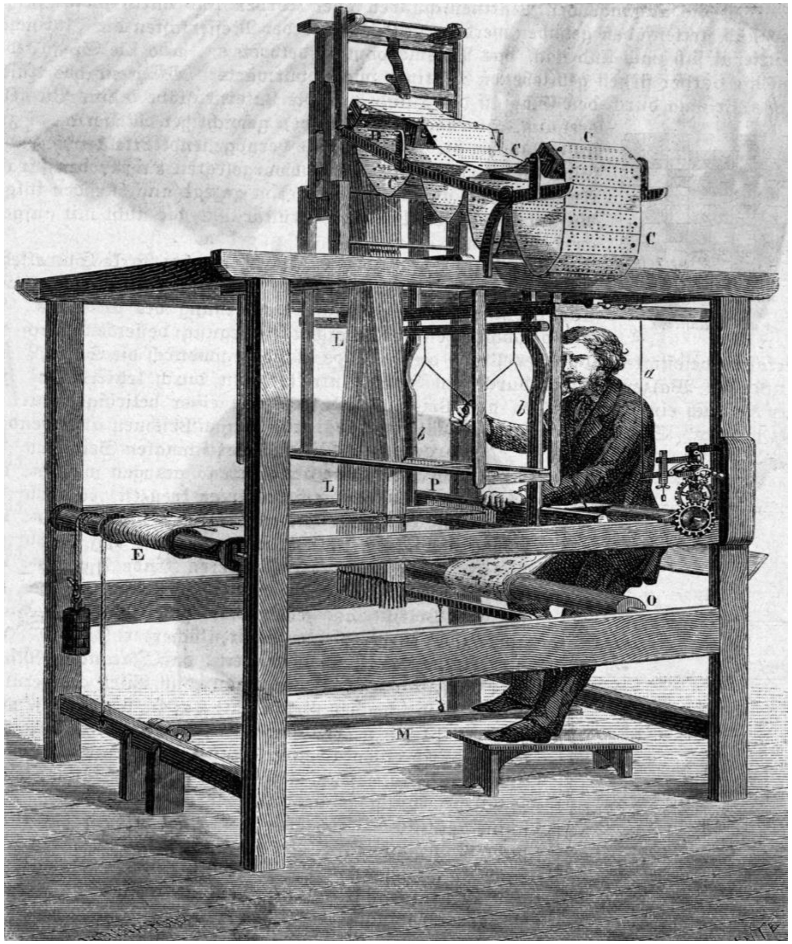
Slide from "Introduction to Computing"



The Pascaline: One of the Earliest Mechanical Calculators

Slide from “Introduction to Computing”

The Early Period: Up to 1940

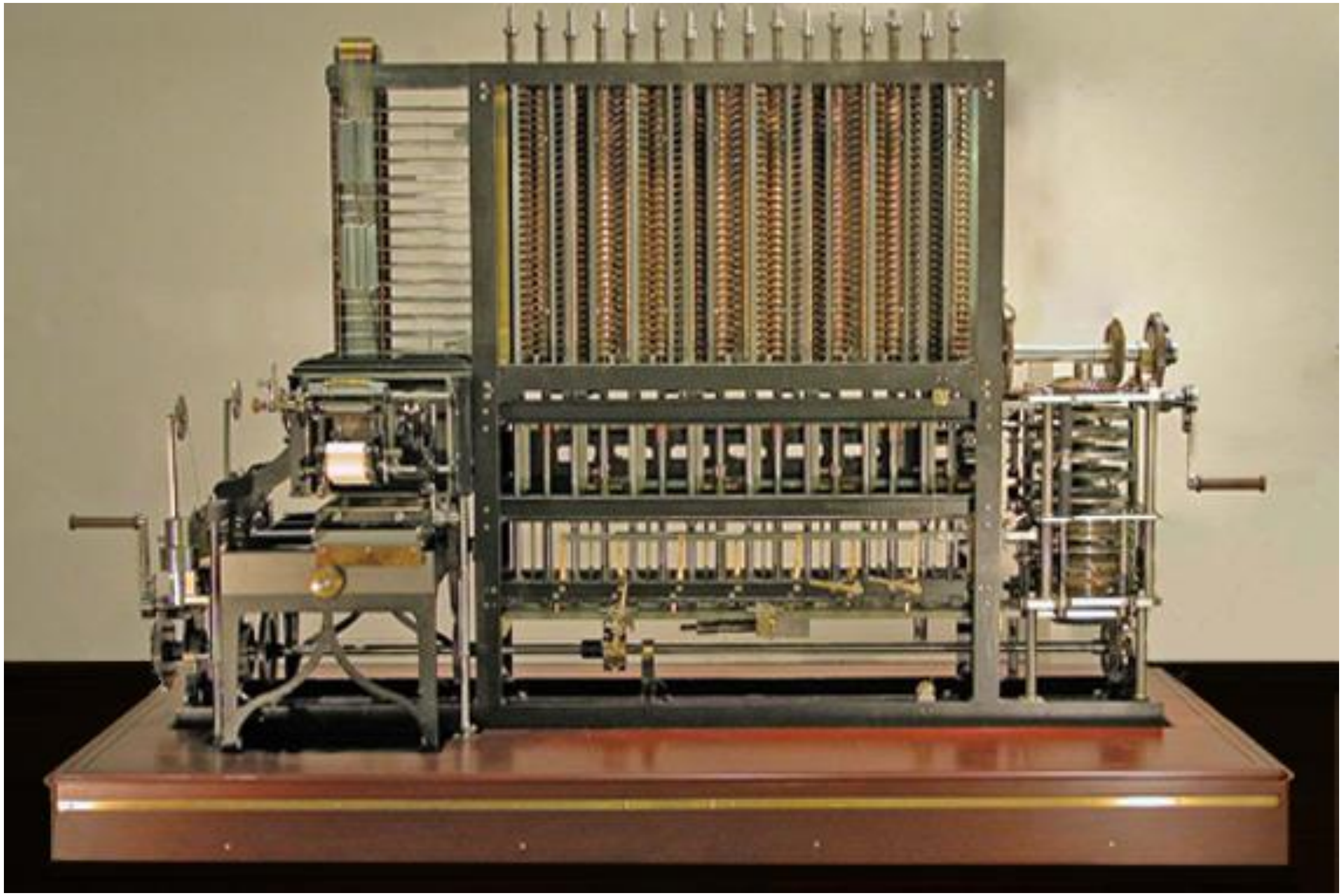


Jacquard's Loom

Also see http://www.computersciencelab.com/ComputerHistory/HistoryPt2.htm

Slide from "Introduction to Computing"

Difference engine



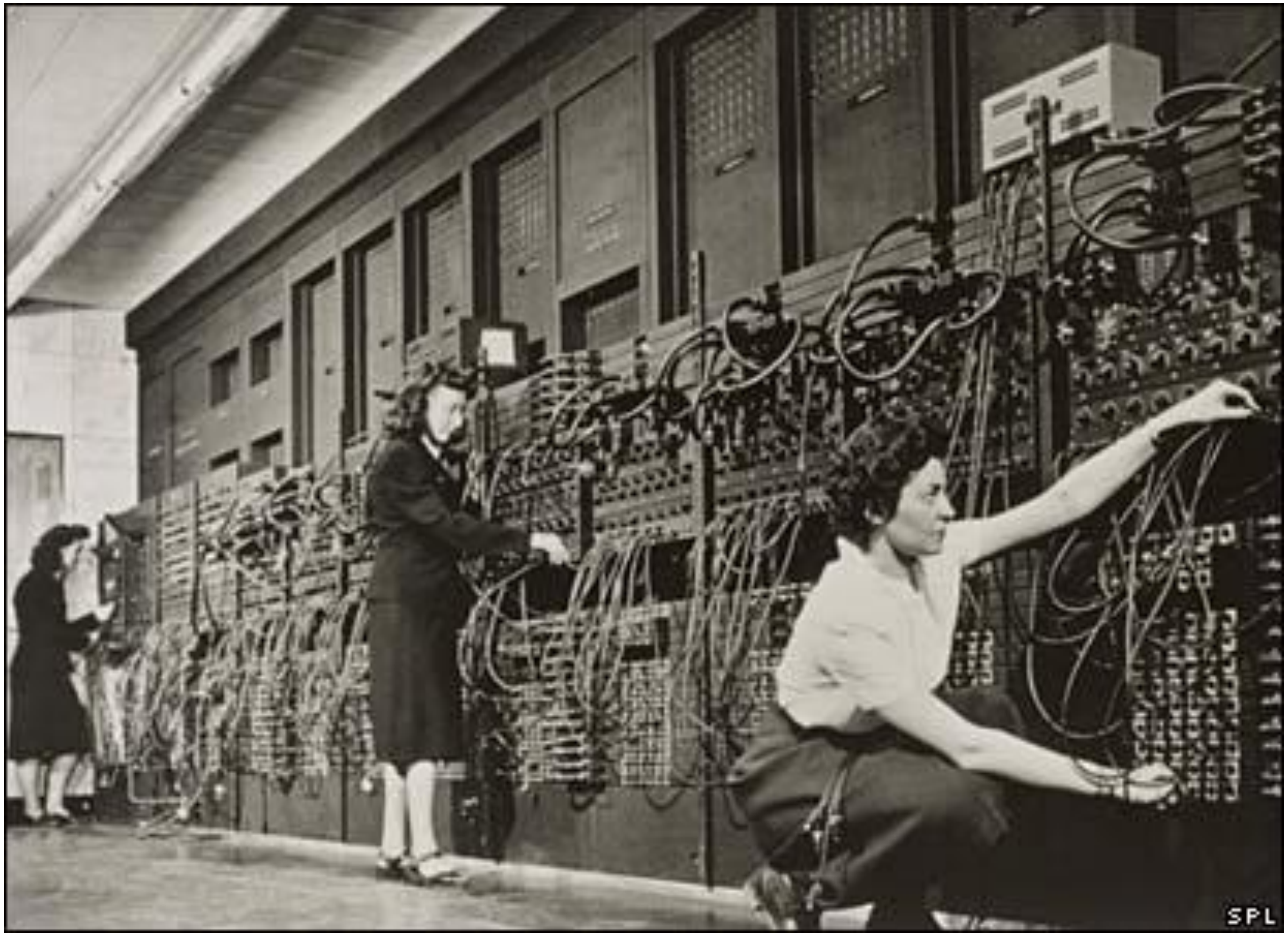
http://www.youtube.com/watch?v=0anIyVGeWOI

Slide from "Introduction to Computing"

The Harvard Mark-I

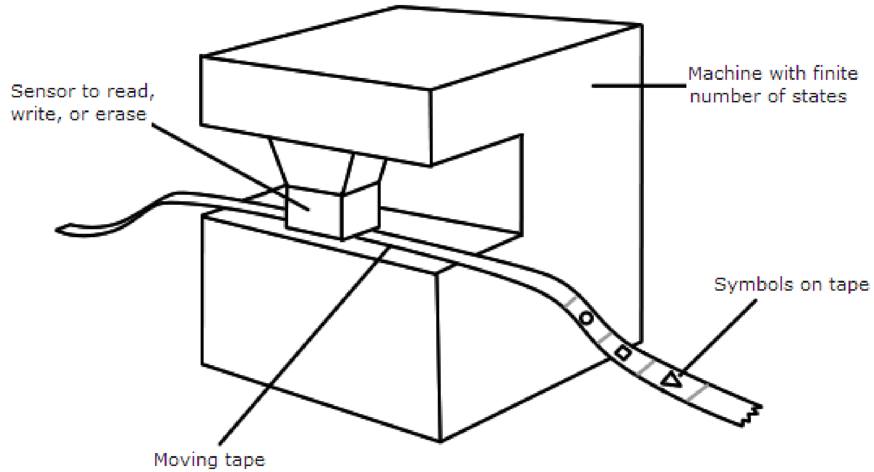
Grace M. Hopper working on the Harvard Mark-I, developed by IBM and Howard Aiken. The Mark-I remained in use at Harvard until 1959, even though other machines had surpassed it in performance, providing vital calculations for the navy in World War II.

Slide from "Introduction to Computing"



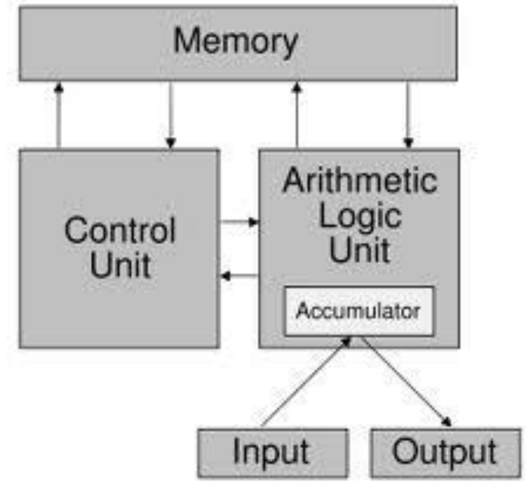
Programming the ENIAC

Slide from "Introduction to Computing"



A Turing Machine

Turing Machine

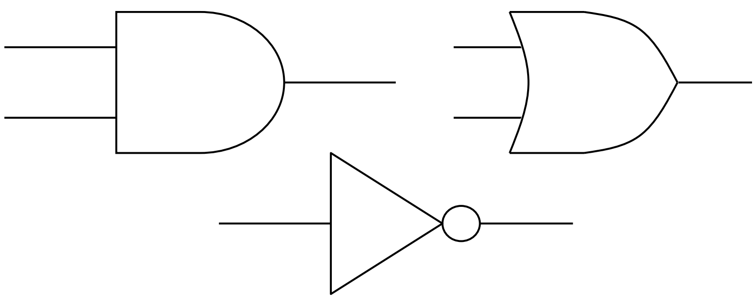


Von Neumann Architecture

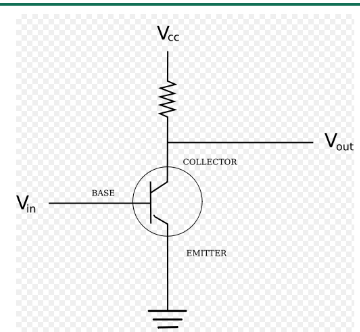
DIGITAL COMPUTATION

A computer

Devices

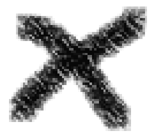
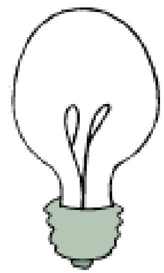
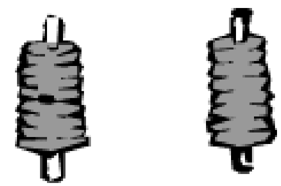
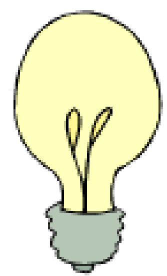
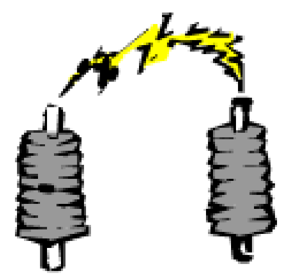


Gates

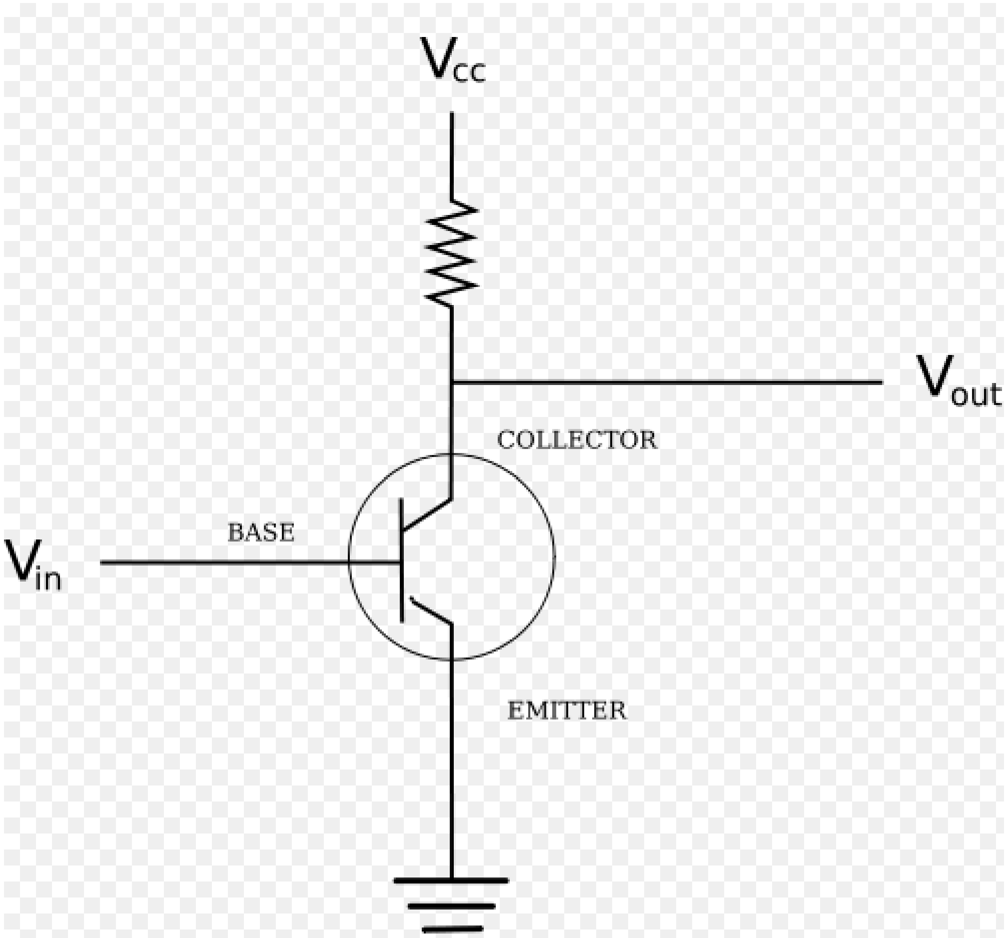


Transistors

Everything in a PC is Binary
… well, almost …

States of a Bit			
0	 2+2=5 FALSE	 OFF	 LOW VOLTAGE
1	 2+2=4 TRUE	 ON	 HIGH VOLTAGE

A transistor



This circuit functions
as a switch. In other words,
based on the *control* voltage,
the circuit either passes Vin
to output or not.

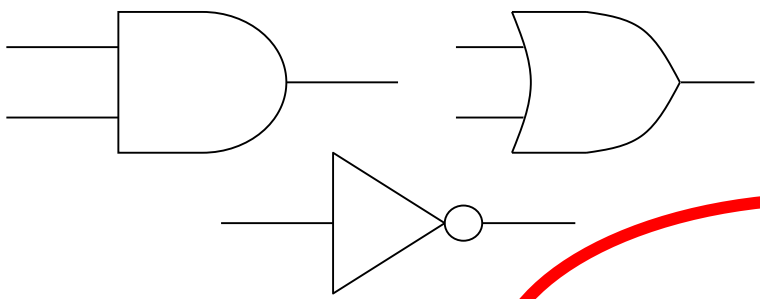
Examples of transistors

Replica of the first transistor

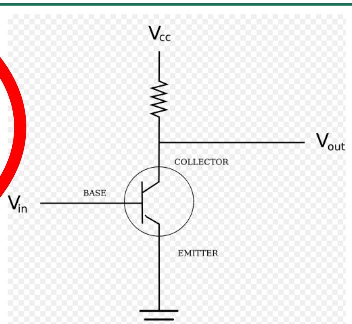
A set of transistors, depicting the fast change in technology.

A computer

Devices



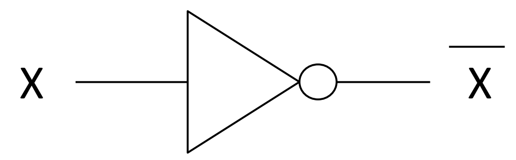
Gates



Transistors

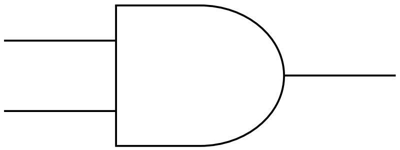
NOT Gate

X	$\overline{X}$
0	1
1	0



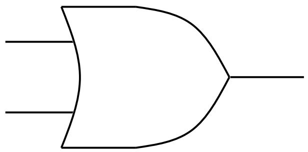
AND gate

X	Y	X·Y
0	0	0
0	1	0
1	0	0
1	1	1



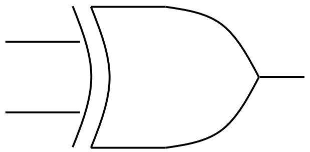
OR Gate

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



XOR Gate

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0



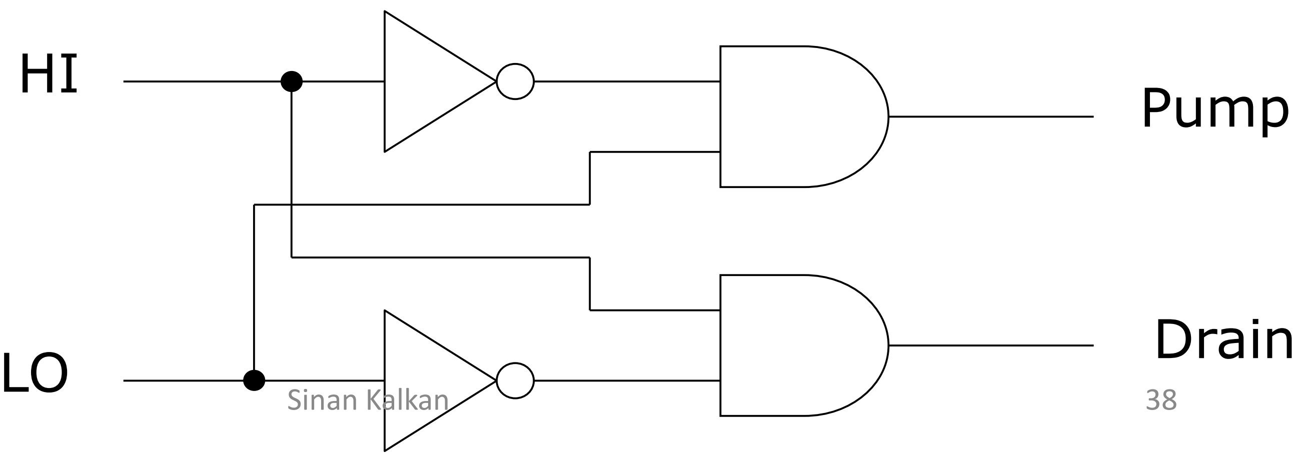
An example problem: Water Tank

HI	LO	Pump	Drain
0	0	0	0
0	1	**1**	0
1	0	0	**1**
1	1	x	x

Truth Table Representation

→ Tank level is OK
→ Low level, pump more in
→ High level, drain some out
→ Inputs cannot occur

Schematic Representation



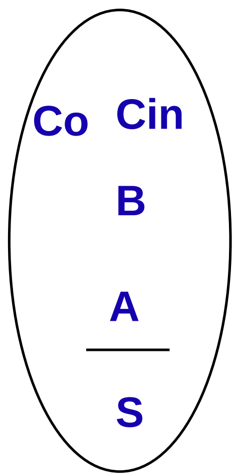
The binary addition

$$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

Question (Binary notation) : 111010 + 11011 = ?

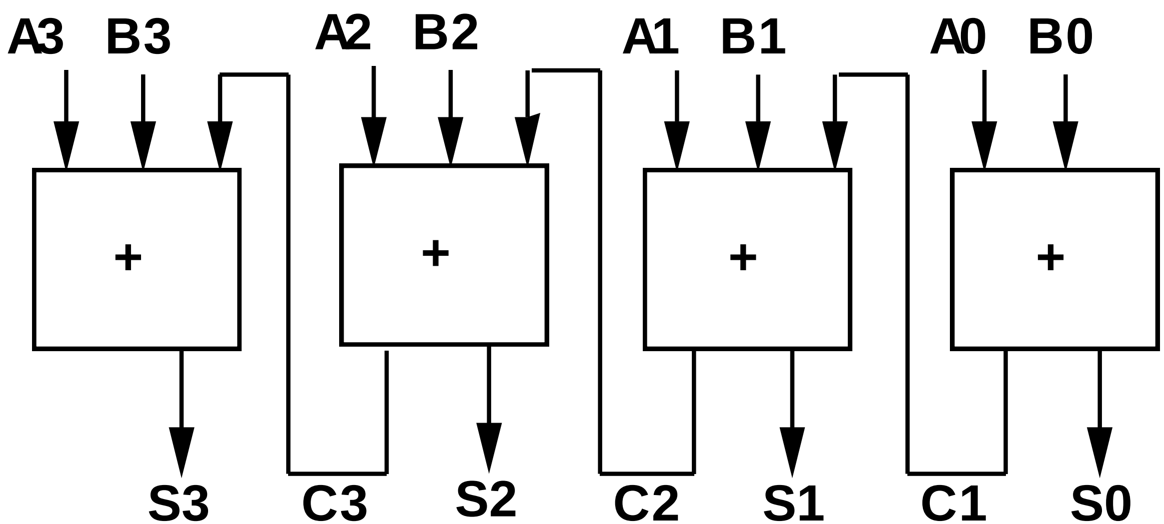
1-bit full-adder

```
      1
    0 0 1 1
+   0 0 1 0
   ---------
    0 1 0 1
```



A	B	CI	S	CO
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

N-bit Adder



Representing data

Data Representation

- Based on 1s and 0s
  - So, everything is represented as a set of binary numbers
- We will now see how we can represent:
  - Integers: 3, 1234435, -12945 etc.
  - Floating point numbers: 4.5, 124.3458, -1334.234 etc.
  - Characters: /, &, +, -, A, a, ^, 1, etc.
  - …

Binary Representation of Numeric Information

- Decimal numbering system
  - Base-10
  - Each position is a power of 10

    $3052 = 3 \times 10^3 + 0 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$

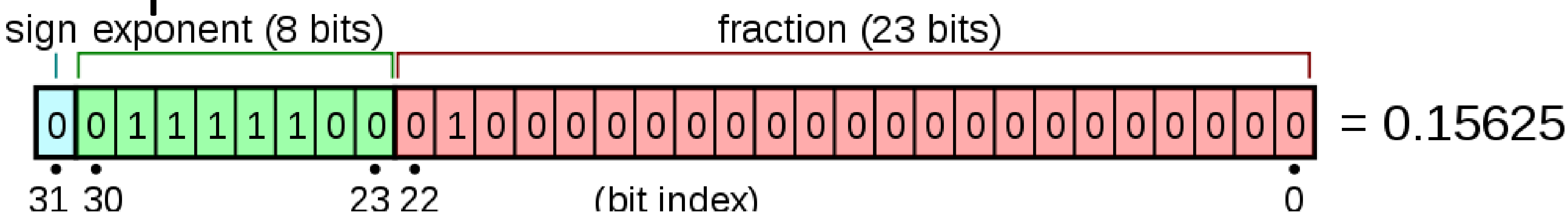
- Binary numbering system
  - Base-2
  - Uses ones and zeros
  - Each position is a power of 2

    $1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

Decimal-to-binary Conversion

- Divide the number until zero:
  - 35 / 2 = 17x2 + 1
  - 17 / 2 = 8x2 + 1
  - 8 / 2 = 4x2 + 0
  - 4 / 2 = 2x2 + 0
  - 2 / 2 = 1x2 + 0
- Therefore, 35 has the binary representation: 100011

IEEE 32bit Floating-Point Number Representation



$$= (-1)^{\text{sign}}(1.b_{-1}b_{-2}...b_{-23})_2 \times 2^{e-127}$$

- M x 2^E
- Exponent (E): 8 bits
  - Add 127 to the exponent value before storing it
  - **E can be 0 to 255 with 127 representing the real zero.**
- Fraction (M - Mantissa): 23 bits
- $2^{128} = 1.70141183 \times 10^{38}$

IEEE 32bit Floating-Point Number Representation

- Example: 12.375
- The digits before the dot:
  - $(12)_{10} \rightarrow (1100)_2$
- The digits after the dot:
  - 1^{st} Way: 0.375 → 0x½ + 1x¼ + 1x1/8 → 011
  - 2^{nd} Way: Multiply by 2 and get the integer part until 0:
    - 0.375 x 2 = 0.750 = 0 + 0.750
    - 0.750 x 2 = 1.50 = 1 + 0.50
    - 0.50 x 2 = 1.0 = 1 + 0.0
- $(12.375)_{10} = (1100.011)_2$
- NORMALIZE: $(1100.011)_2 = (1.100011)_2 x2^3$
- Exponent: 3, adding 127 to it, we get 1000 0010
- Fraction: 100011
- Then our number is: 0 10000010 10001100000000000000000

Binary Representation of Textual Information

- Characters are mapped onto binary numbers
  - ASCII (American Standard Code for Information Interchange) code set
    - Originally: 7 bits per character; 128 character codes
  - Unicode code set
    - 16 bits per character
  - UTF-8 (UCS Transformation Format) code set.
    - Variable number of 8-bits.

Binary Representation of Textual Information (cont'd)

ASCII
7 bits long

Decimal	Binary	Val.
48	00110000	0
49	00110001	1
50	00110010	2
51	00110011	3
52	00110100	4
53	00110101	5
54	00110110	6
55	00110111	7
56	00111000	8
57	00111001	9
58	00111010	:
59	00111011	;
60	00111100	<
61	00111101	=
62	00111110	>
63	00111111	?
64	01000000	@
65	01000001	A
66	01000010	B

Dec.	Unicode	Charac.
0x30	0x0030	0
0x31	0x0031	1
0x32	0x0032	2
0x33	0x0033	3
0x34	0x0034	4
0x35	0x0035	5
0x36	0x0036	6
0x37	0x0037	7
0x38	0x0038	8
0x39	0x0039	9
0x3A	0x003A	:
0x3B	0x003B	;
0x3C	0x003C	<
0x3D	0x003D	=
0x3E	0x003E	>
0x3F	0x003F	?
0x40	0x0040	@
0x41	0x0041	A
0x42	0x0042	B

Unicode
16 bits long

Partial listings only!

Integer Types

Following table gives you details about standard integer types with its storage sizes and value ranges:

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating-Point Types

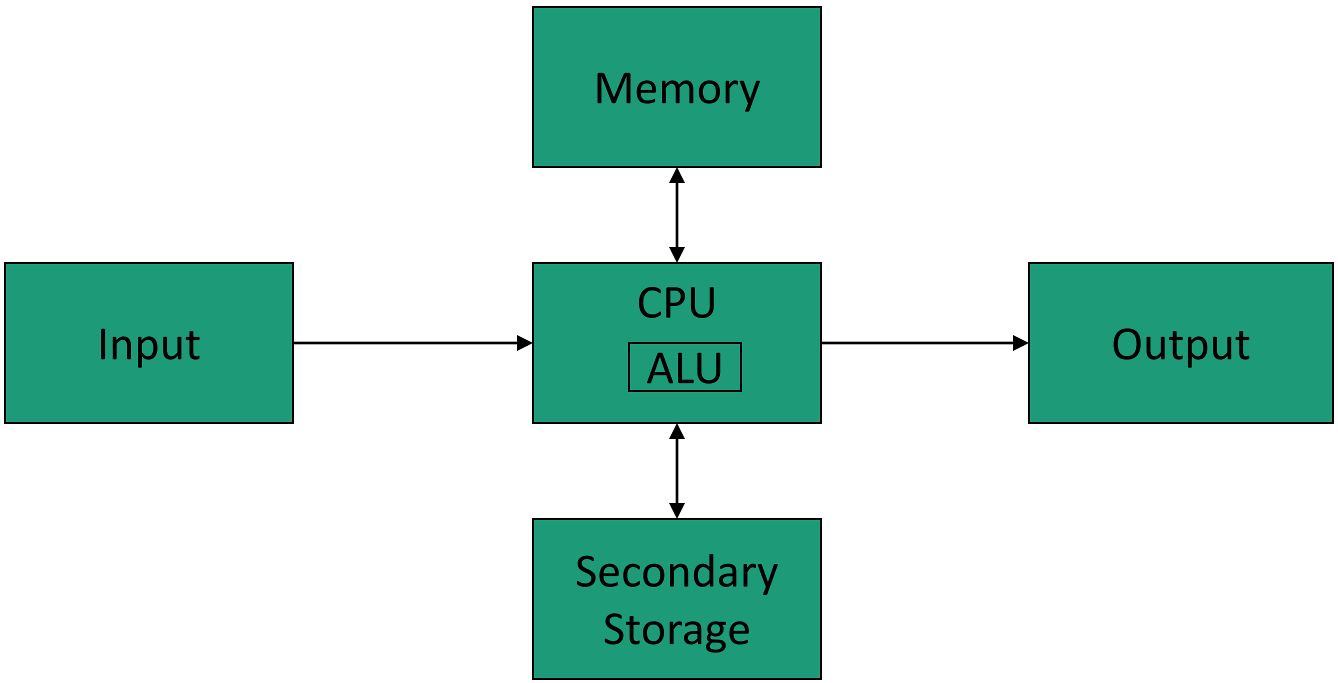
Following table gives you details about standard floating-point types with storage sizes and value ranges and their precision:

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Computer organization

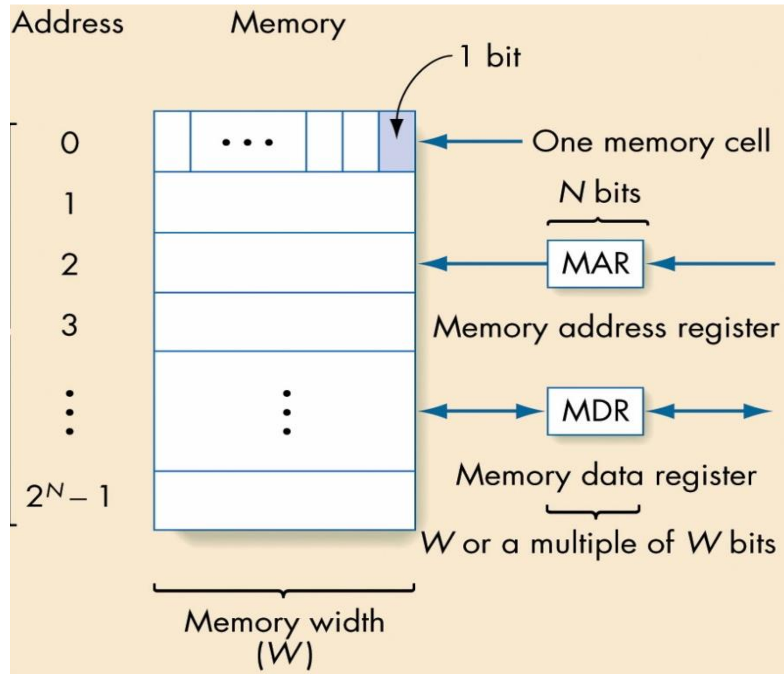
Computer Organization

- Logical organization of computer



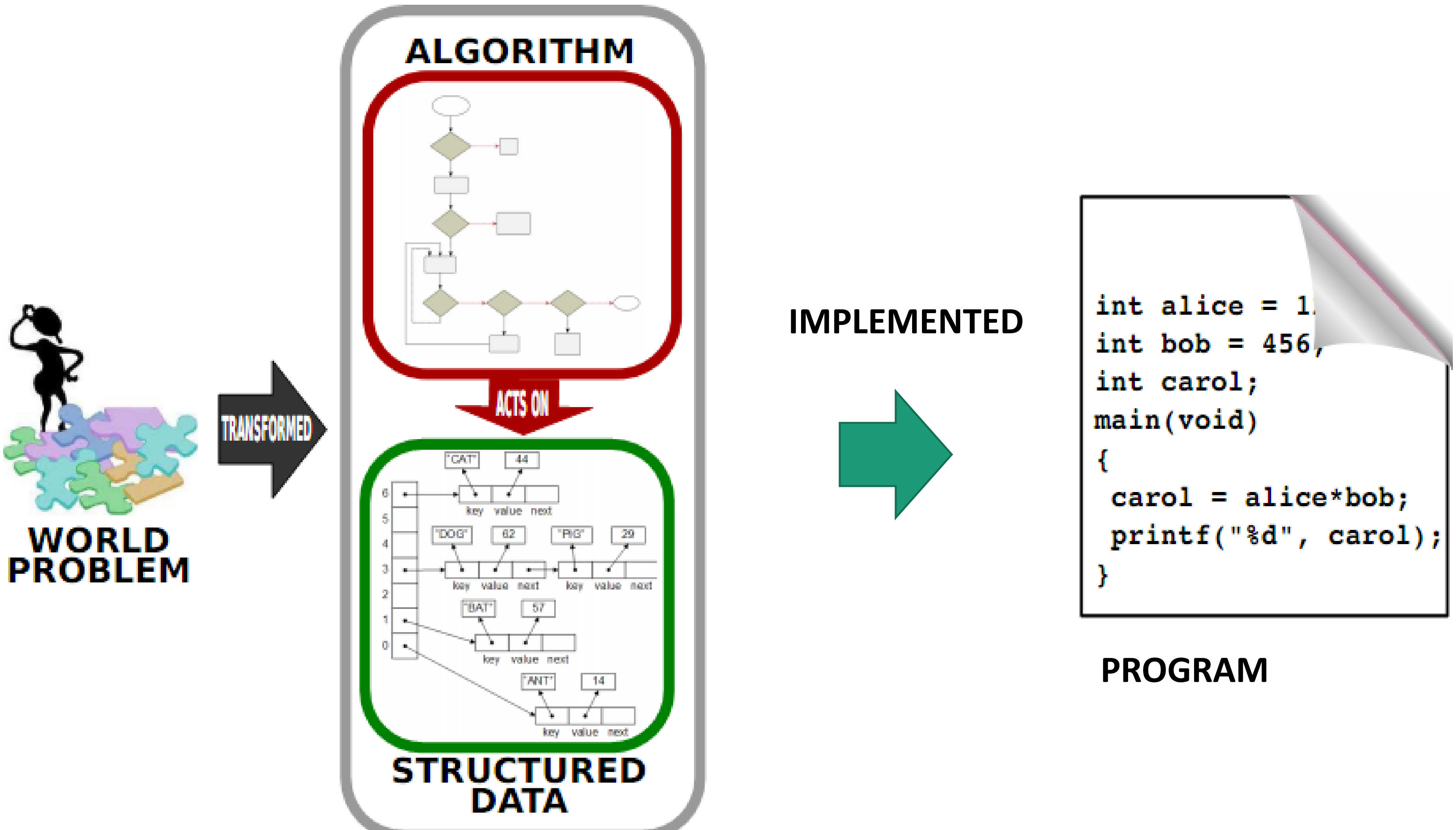
Memory and Cache (continued)

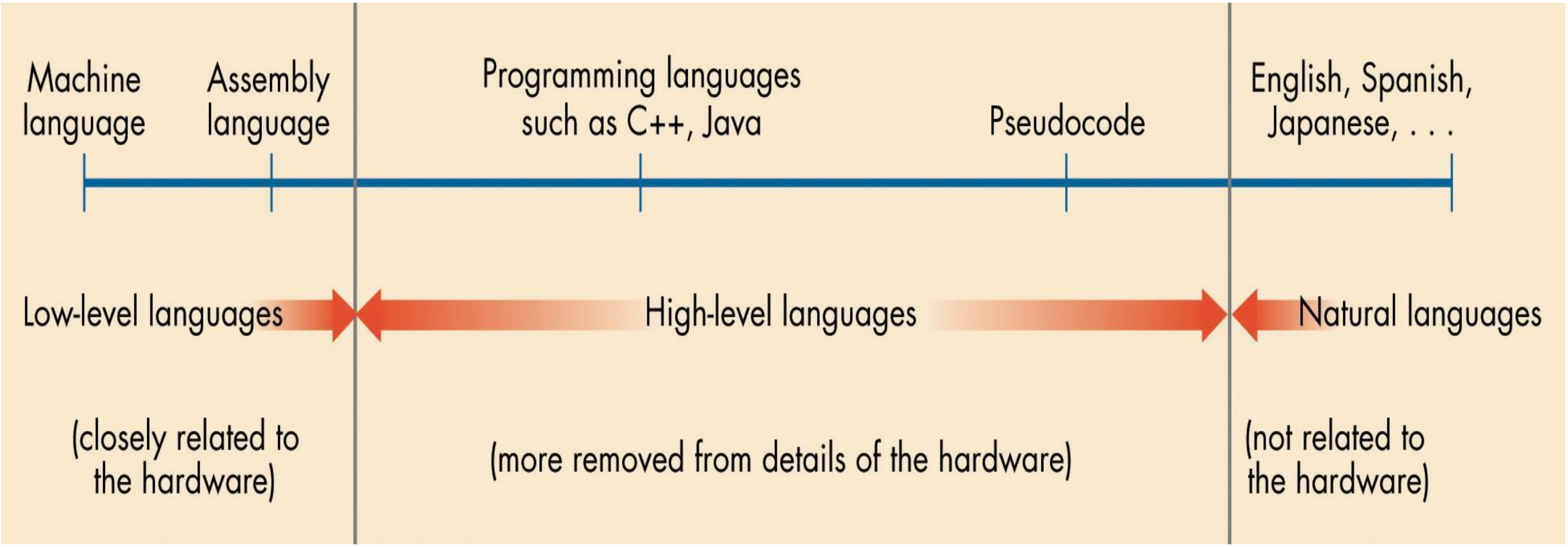
- RAM (Random Access Memory)
  Often called *memory*, *primary memory*
  - Memory made of addressable "cells"
  - Cell size is 8 bits
    - Nowadays, it is 32 or 64 bits.
  - All memory cells accessed in equal time
  - Memory address
    - Unsigned binary number N long
    - Address space is then 2^N cells



Programming

Program, Programming




Machine language
Assembly language
Programming languages such as C++, Java
Pseudocode
English, Spanish, Japanese, . . .
Low-level languages
High-level languages
Natural languages
(closely related to the hardware)
(more removed from details of the hardware)
(not related to the hardware)

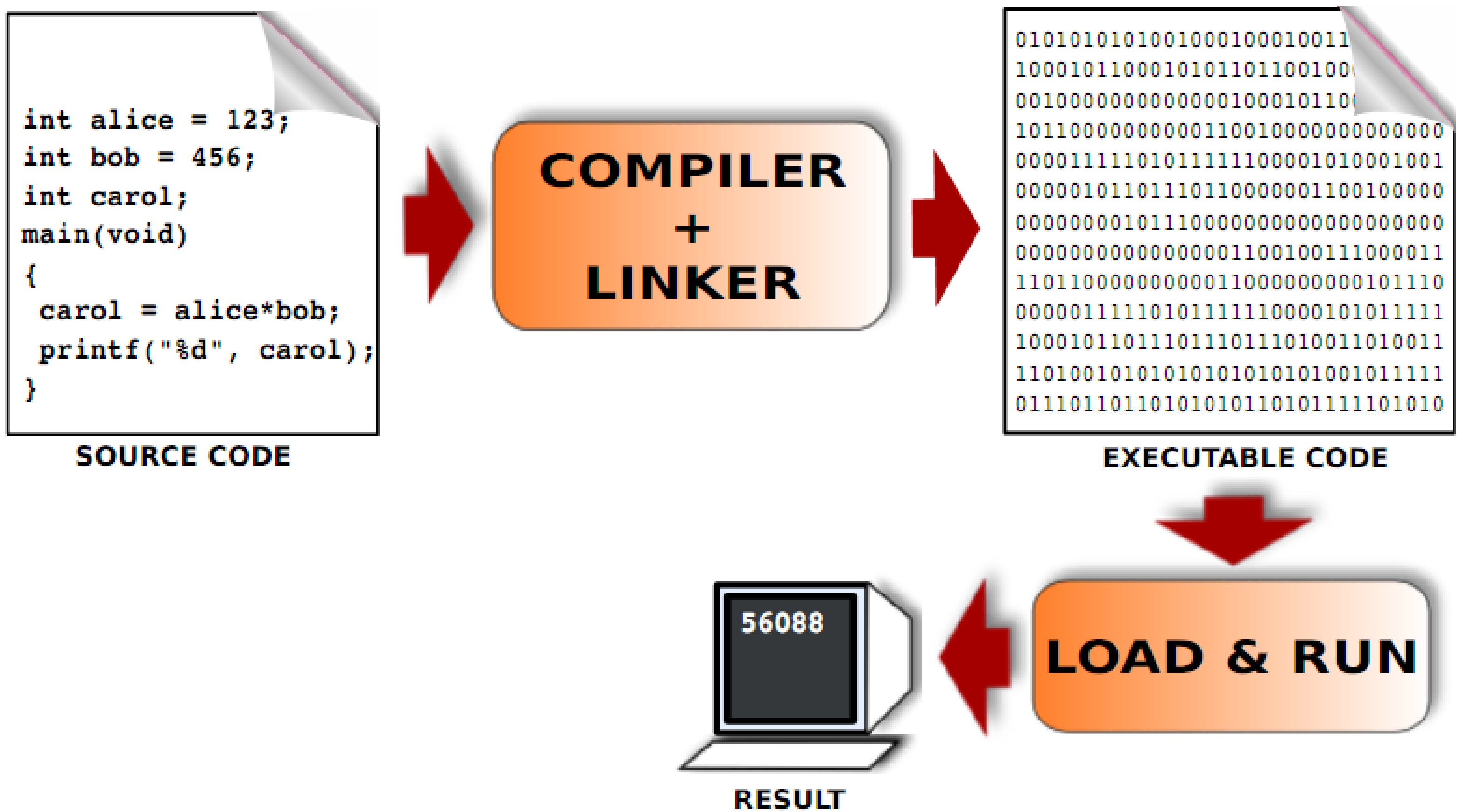
```
01010101 01001000 10001001 11100101 10001011 00010101 10110010 00000011
00100000 00000000 10001011 00000101 10110000 00000011 00100000 00000000
00001111 10101111 11000010 10001001 00000101 10111011 00000011 00100000
00000000 10111000 00000000 00000000 00000000 00000000 11001001 11000011
...
11001000 00000001 00000000 00000000  00000000 00000000
```

```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    alice(%rip), %edx
        movl    bob(%rip), %eax
        imull   %edx, %eax
        movl    %eax, carol(%rip)
        movl    $0, %eax
        leave
        ret
alice:
        .long   123
bob:
        .long   456
```

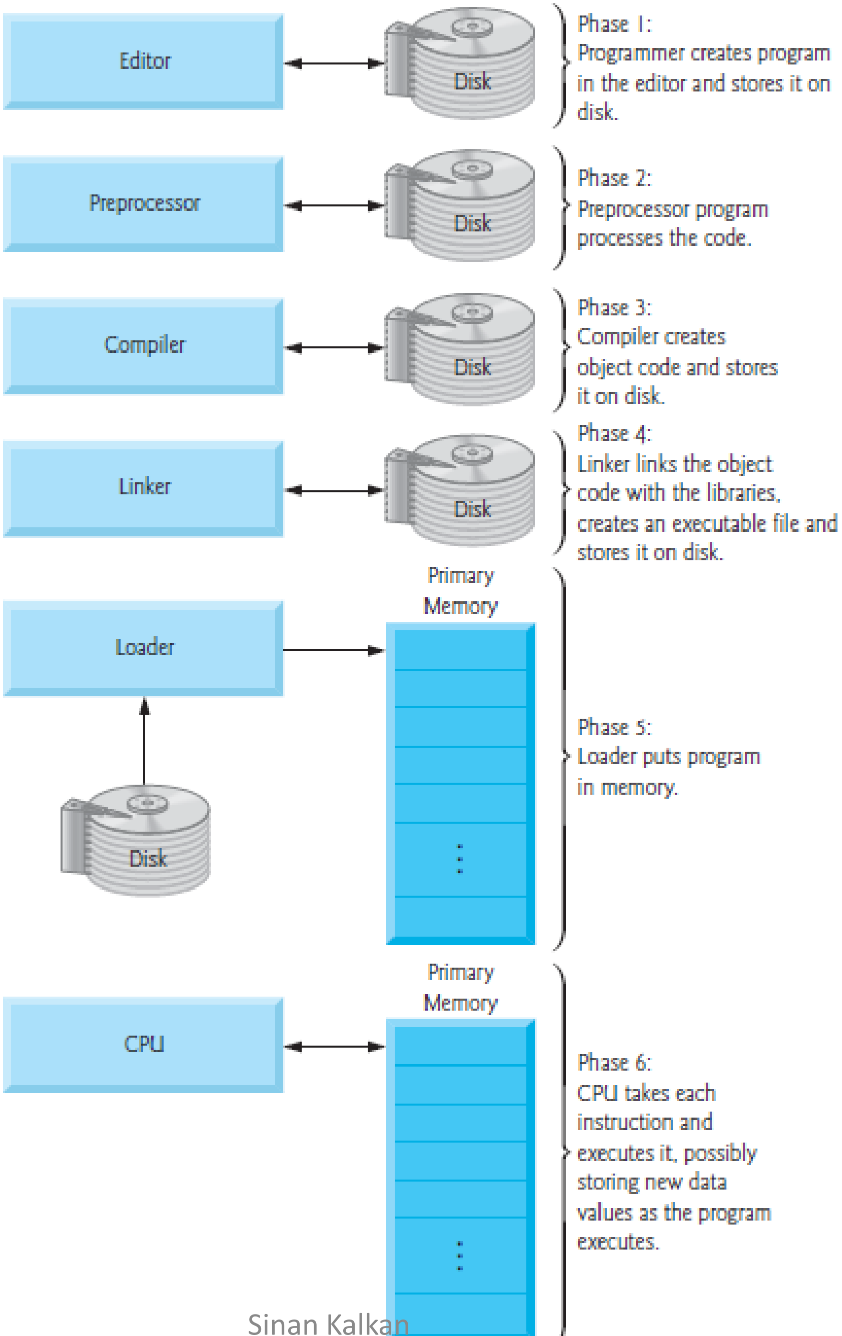
```
int alice = 123;
int bob = 456;
int carol;
main(void)
{
  carol = alice*bob;
}
```

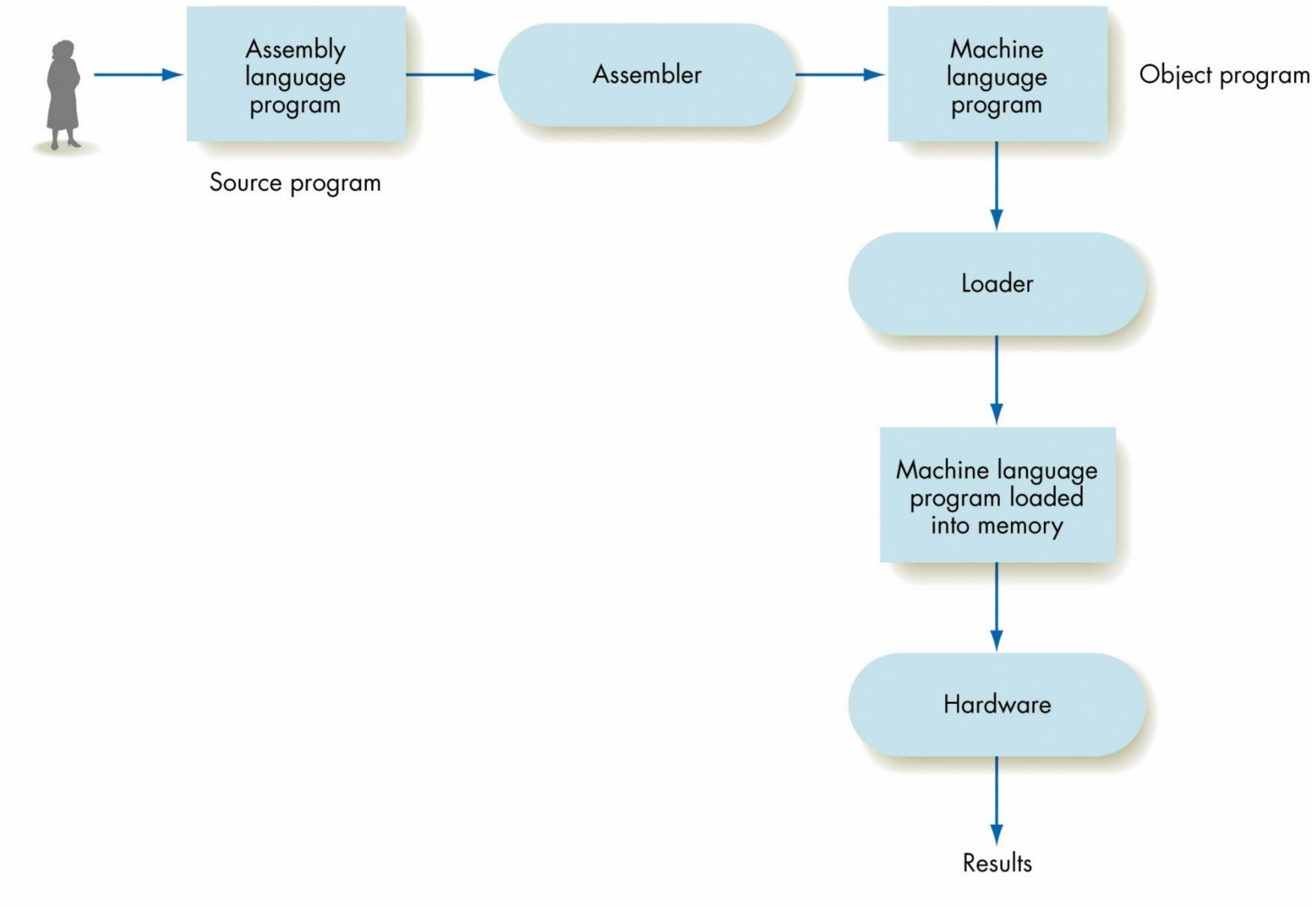
How are languages implemented

COMPILATIVE APPROACH



C language development environment





The Translation/Loading/Execution Process

Address	Contents
100	Value of *a*
101	Value of *b*
102	Value of *c*

Algorithmic notation	Address	Contents	(Commentary)
		⋮	
1. Set *a* to the value *b* + *c*	50	LOAD 101	Put the value of *b* into register R.
	51	ADD 102	Add *c* to register R. It now holds *b* + *c*.
	52	STORE 100	Store the contents of register R into *a*.

Machine Language Instruction Sequences

Algorithmic notation	Address	Contents	(Commentary)
2. If $a > b$ then	50	COMPARE 100, 101	Compare *a* and *b* and set condition codes.
set *c* to the value *a*	51	JUMPGT 54	Go to location 54 if $a > b$.
Else	52	MOVE 101, 102	Get here if $a \leq b$, so move *b* into *c*
set *c* to the value *b*	53	JUMP 55	and skip the next instruction.
	54	MOVE 100, 102	Move *a* into *c*.
	55	• • •	Next statement begins here.

Bugs, Errors

- Syntax Errors

  Area = 3.1415 * R * R

  Area = 3.1415 x R x R

- Run-time Errors

```
>>> def SqrtDelta(a,b,c):
>>>         return sqrt(b*b - 4*a*c)
>>>
>>> print SqrtDelta(1,3,1)
2.23606797749979
>>> print SqrtDelta(1,1,1)
ValueError: math domain error
```

Bugs, Errors

- Logical Errors

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

```
>>> root1 = (- b + sqrt(b*b - 4*a*c)) / 2*a
```

- Design Errors

$$x^3 + ax^2 + bx + c = 0$$

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

C

History of C

- C
  - – Developed by Denis M. Ritchie at AT&T Bell Labs in **1972** as a systems programming language
  - – Used to develop UNIX
  - – Used to write modern operating systems
  - – Hardware independent (portable)

- Standardization
  - – Many slight variations of C existed, and were incompatible
  - – Committee formed to create a "unambiguous, machine independent“ definition
  - – Standard created in 1989, updated in 1999

```c
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Welcome to C!
```

Fig. 2.1 | A first program in C.