Computer Organization and Architecture : Introduction

Instructor :

Bibhas Ghoshal

Department of Information Technology

IIIT Allahabad

Administrivia

Course website :

profile.iiita.ac.in/bibhas.ghoshal/teaching_coa_2021.html

profile.iiita.ac.in/bibhas.ghoshal/teaching_coa_lab_2021.html

Recommended Books :

1. John L. Hennessy and David A. Patterson, Computer Architecture -- A Quantitative Approach, 5th Edition, Morgan Kaufmann Publications, Elsevier, Inc., 2012.

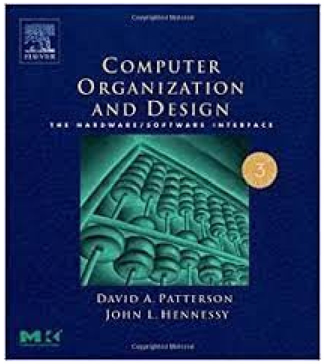
2. Carl Hamachar, Zvonco Vranesic and Safwat Zaky, Computer Organization, McGraw Hill

3. William Stallings, Computer Organization and Architecture: Designing for Performance, Pearson Education

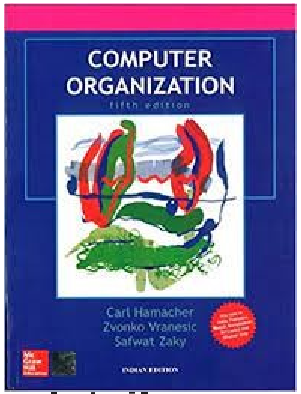
4. John P. Hayes , Computer Architecture and Organization, McGraw Hill

Resources for the course

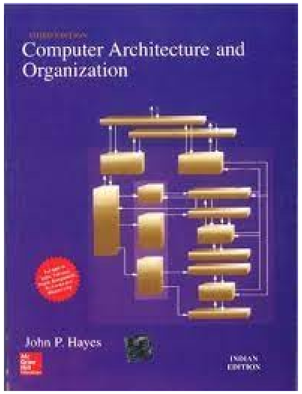
H&P



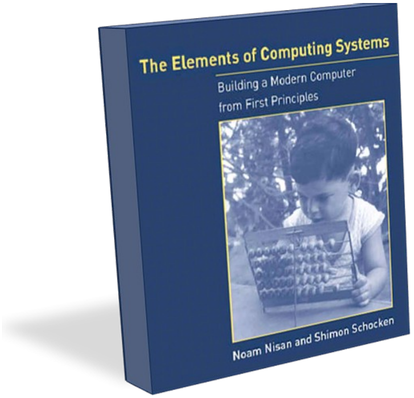
Hamacher et al.



Hayes



- Slides provide key concepts, books provide details
- Use lecture notes supplemented with textbook and internet
- Develop your assembly language programming skills
- For design

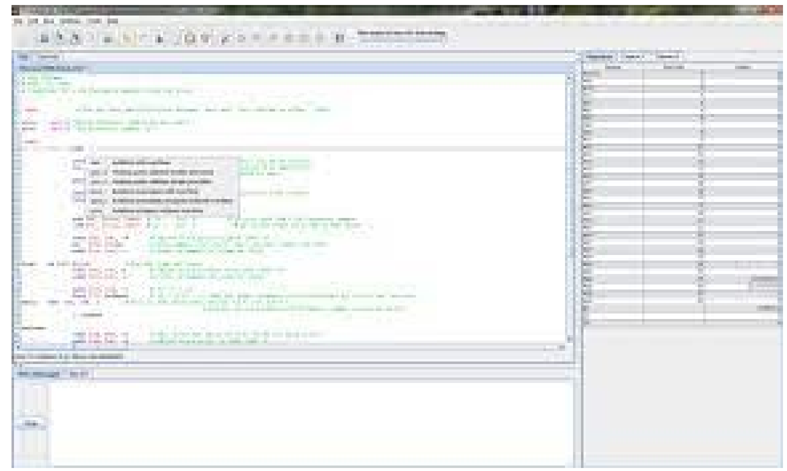


Elements of Computing System : Building a computer from first principles

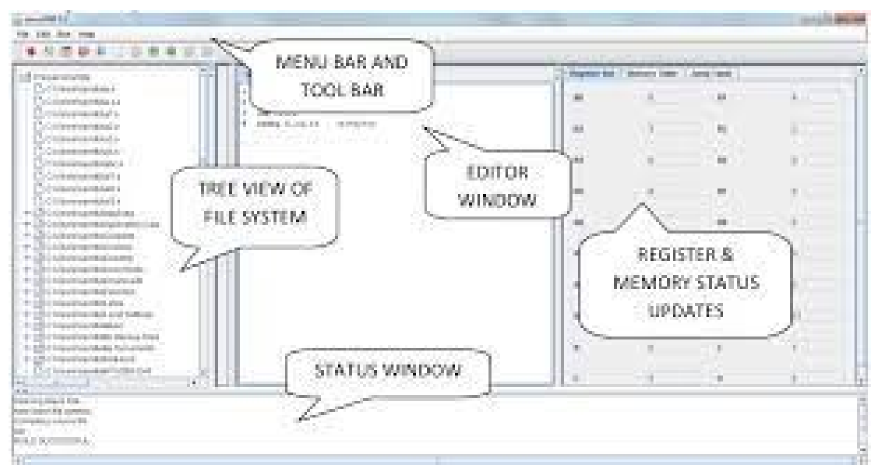
Resources for the Lab

<u>Architectural Simulators:</u>

MARS (MIPS Simulation)



EmuARM

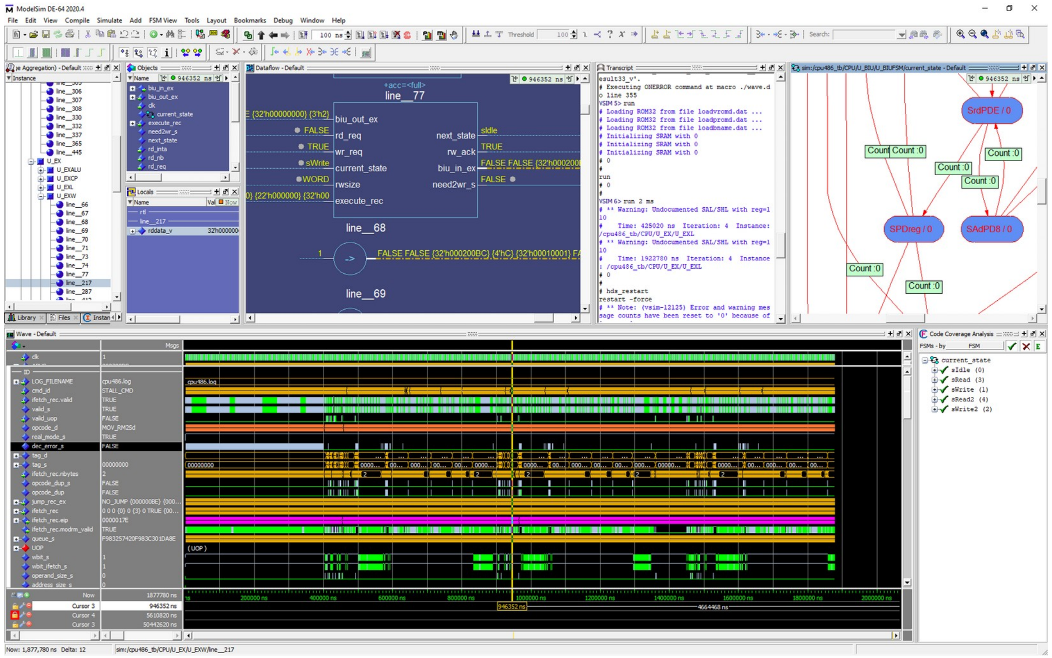


Resources for the Lab

Design :

1. NAND to Tetris - https://www.nand2tetris.org/

2. ModelSim simulator



What is the course all about ???

- What are the components of a computer and how do they work?
- How to program a computer?
- How to store different kinds of data in a computer ?
- How can I run my programs faster ?
- Use of techniques such as caching and pipelining
- How to work with multiple processors?
- What are GPUs and how can they improve performance?
- Can I build my own computer?

.... and more

What is a computer ?

Computer

From Wikipedia, the free encyclopedia

For other uses, see Computer (disambiguation).

A **computer** is a machine that can be programmed to carry out sequences of arithmetic or logical operations automatically. Modern computers can perform generic sets of operations known as programs. These programs enable computers to perform a wide range of tasks. A **computer system** is a "complete" computer that includes the hardware, operating system (main software), and peripheral equipment needed and used for "full" operation. This term may also refer to a group of computers that are linked and function together, such as a computer network or computer cluster.

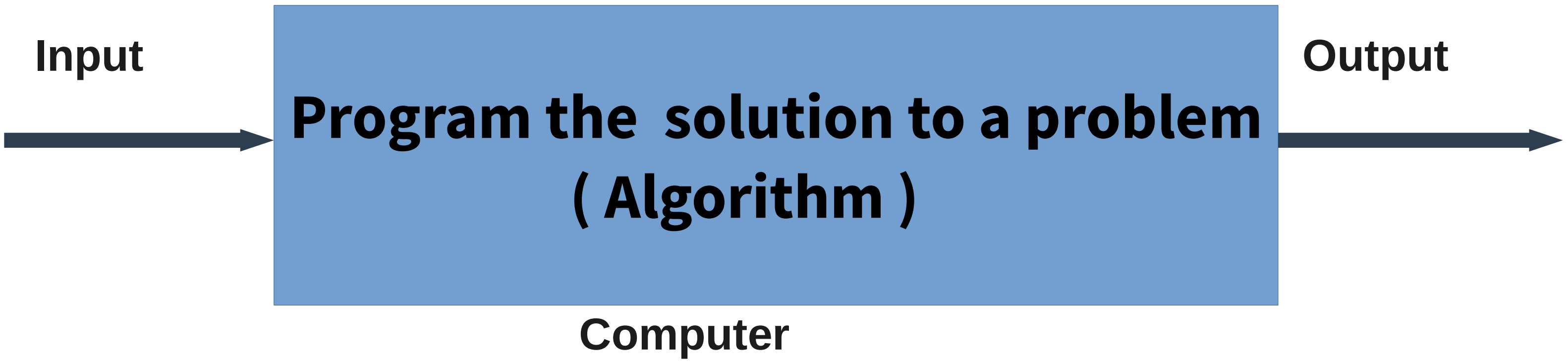
A broad range of industrial and consumer products use computers as control systems. Simple special-purpose devices like microwave ovens and remote controls are included, as are factory devices like industrial robots and computer-aided design, as well as general-purpose devices like personal computers and mobile devices like smartphones. Computers power the Internet, which links hundreds of millions of other computers and users.

Early computers were only meant to be used for calculations. Simple manual instruments like the abacus have aided people in doing calculations since ancient times. Early in the Industrial Revolution, some mechanical devices were built to automate long tedious tasks, such as guiding patterns for looms. More sophisticated electrical machines did specialized analog calculations in the early 20th century. The first digital electronic calculating machines were developed during World War II. The first semiconductor transistors in the late 1940s were followed by the silicon-based MOSFET (MOS transistor) and monolithic integrated circuit (IC) chip technologies in the late 1950s, leading to the microprocessor and the microcomputer revolution in the 1970s. The speed, power and versatility of computers have been increasing dramatically ever since then, with transistor counts increasing at a rapid pace (as predicted by Moore's law), leading to the Digital Revolution during the late 20th to early 21st centuries.

Conventionally, a modern computer consists of at least one processing element, typically a central processing unit (CPU) in the form of a microprocessor, along with some type of computer memory, typically semiconductor memory chips. The processing element carries out arithmetic and logical operations, and a sequencing and control unit can change the order of operations in response to stored information. Peripheral devices include input devices (keyboards, mice, joystick, etc.), output devices (monitor screens, printers, etc.), and input/output devices that perform both functions (e.g., the 2000s-era touchscreen). Peripheral devices allow information to be retrieved from an external source and they enable the result of operations to be saved and retrieved.

Computers and computing devices from different eras – clockwise from top left:
Early vacuum tube computer (ENIAC)
Mainframe computer (IBM System 360)
Desktop computer (IBM ThinkCentre S50 with monitor)
Supercomputer (IBM Summit)
Video game console (Nintendo GameCube)

Computer - A Computing Machine



Q: What can a computer do ?

Ans : Determine if a given integer is a prime number

A Palindrome recognizer

Determine the shortest time journey between two airports

Missile control, finger print recognition, chess player

Speech recognition, language recognition

Computer facilitates the user by validating the devised solution (program / algo) on a number of test cases (input)

Need for a Computer :
Human Efficiency Vs. Machine Performance

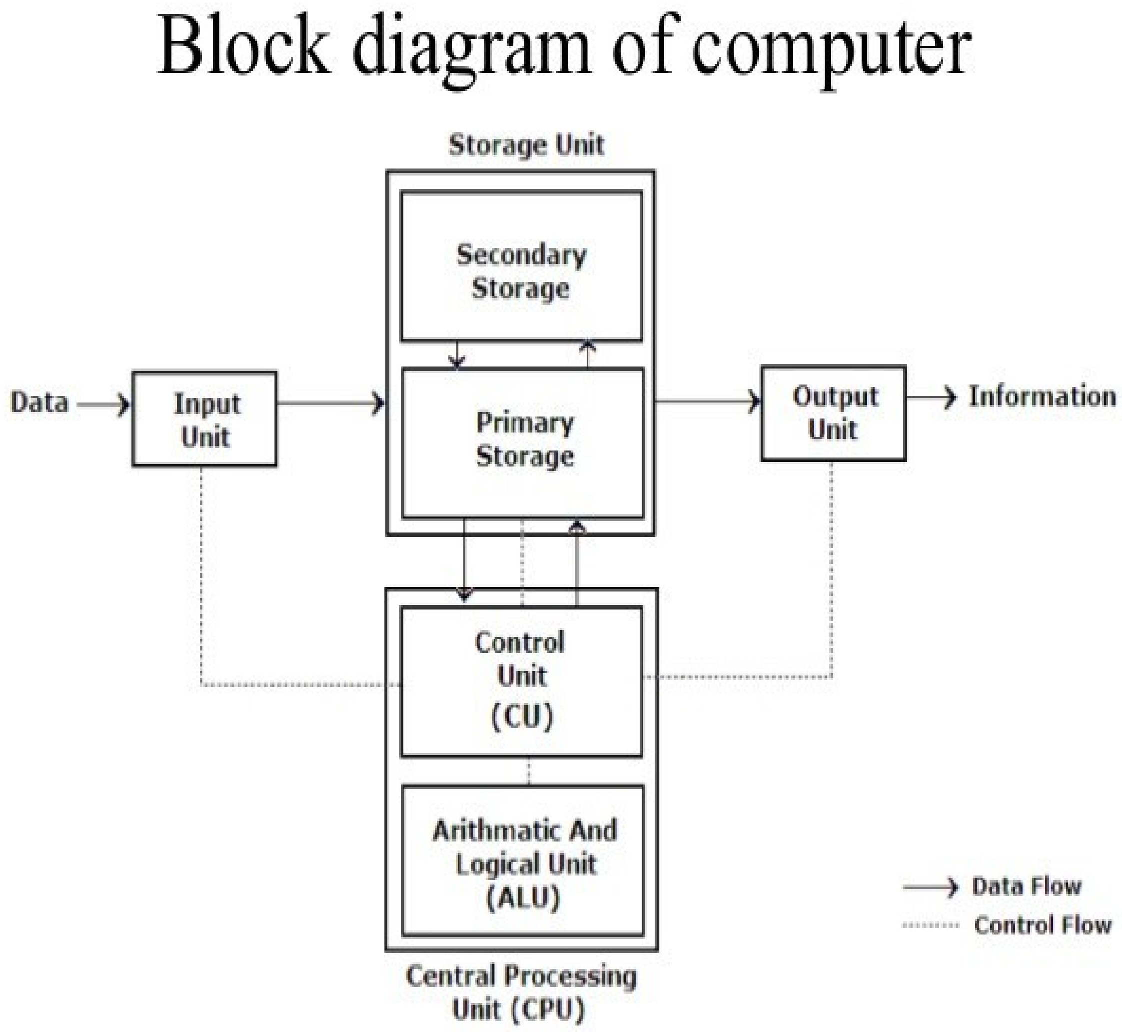
Q: If I am devising the solution (algorithm) to a problem, why do I need a machine?

Ans: The machine performs the task in lesser time and more accurately

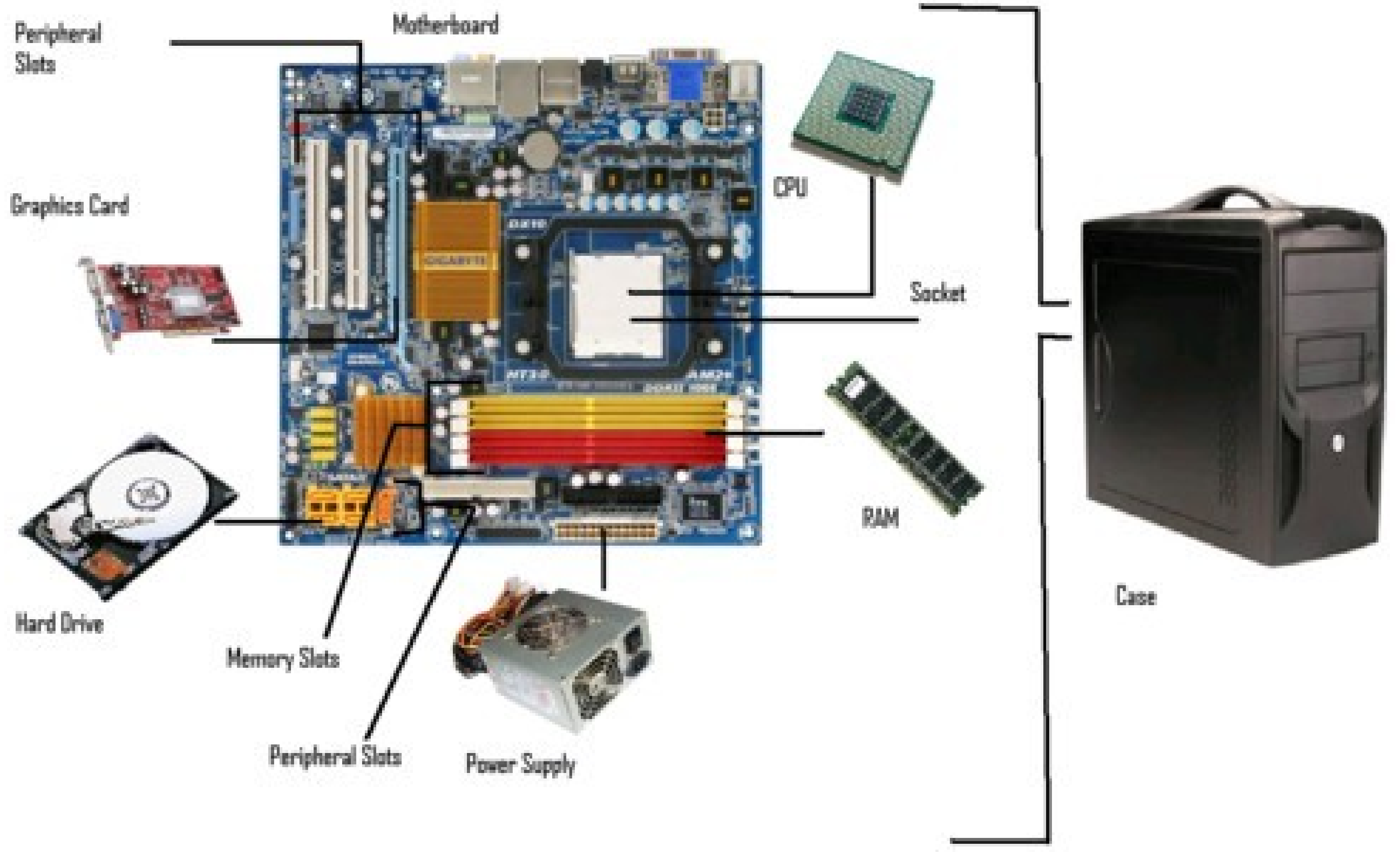
Ex: Compare the task of sorting 1000 numbers when performed by a human and using a computer.

A computer clearly outperforms the human.

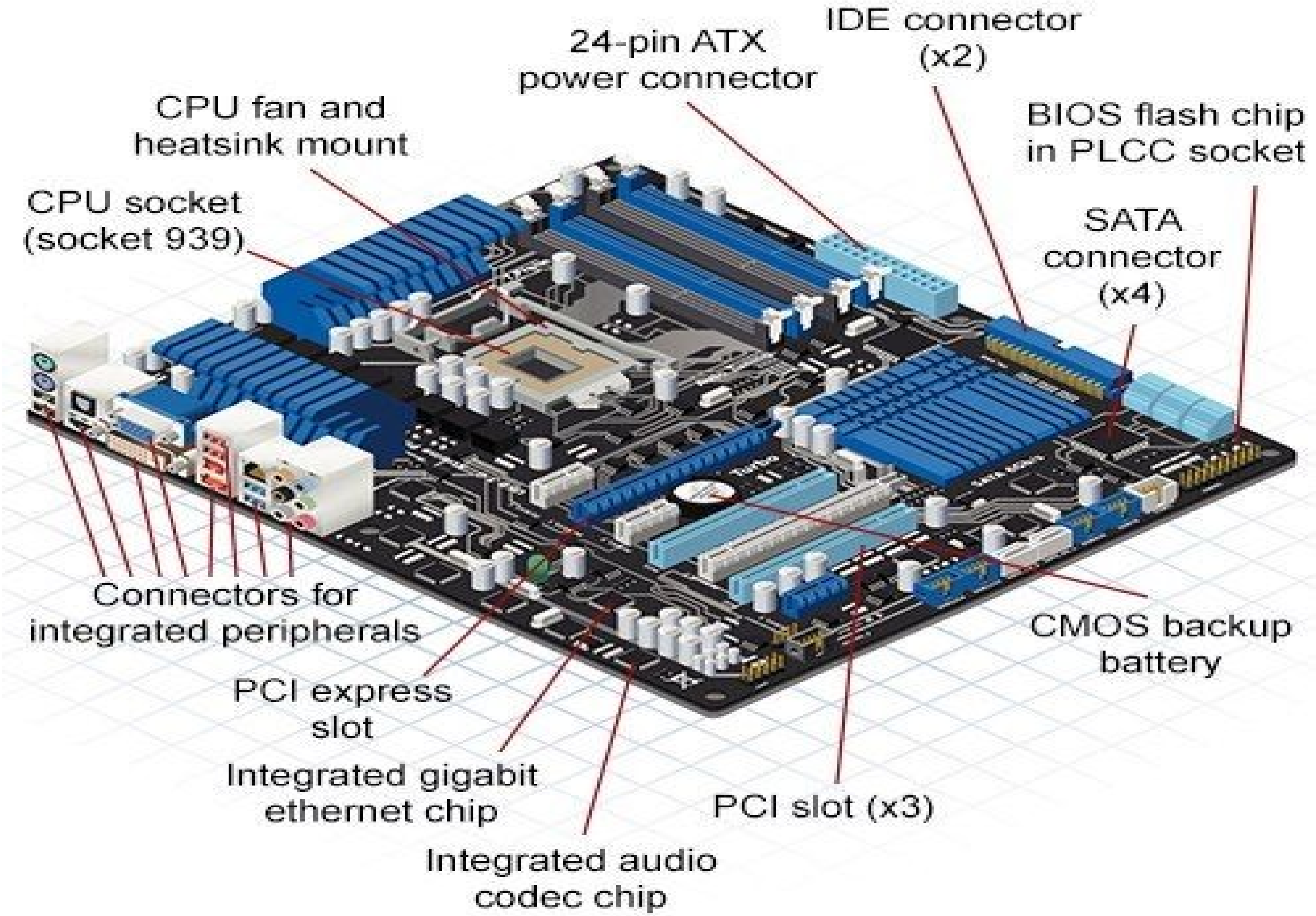
Components of a Computer System



Computer System



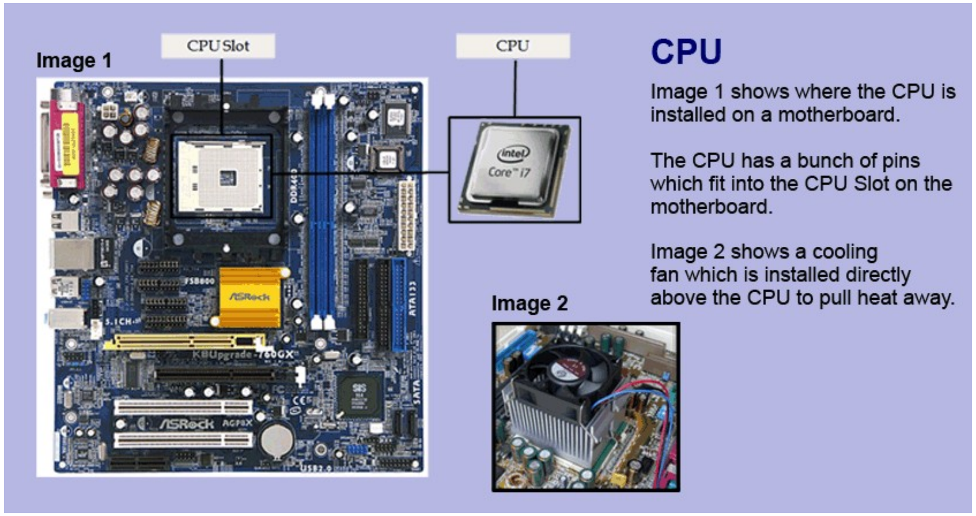
Computer Motherboard

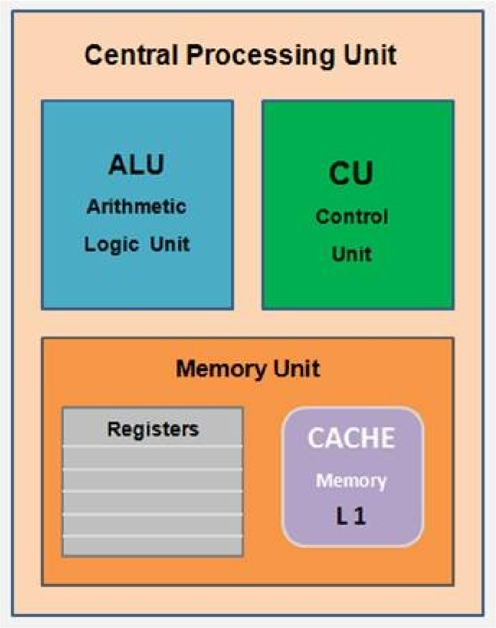


Functions of a Computer

- Data Processing
- Data Control
- Data Movement
- Control

Central Processing Unit (CPU)



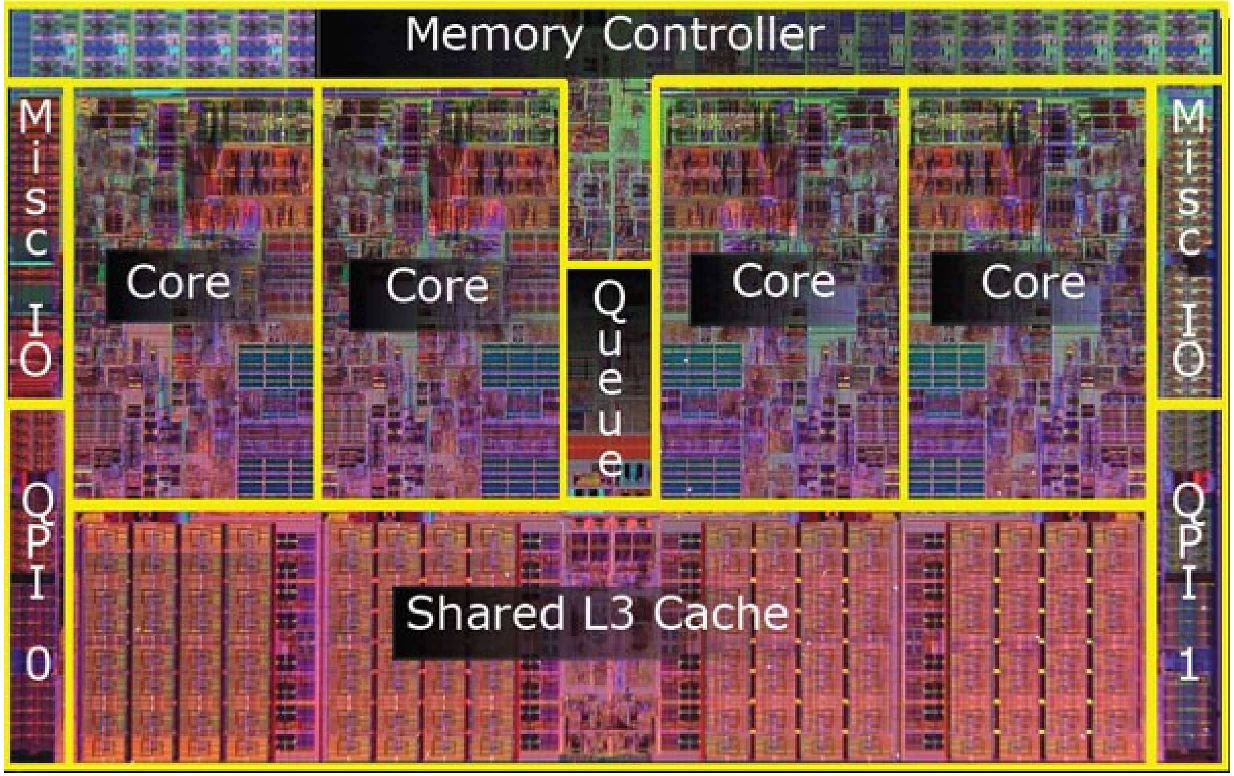


- **The program control unit has a set of registers and control circuit to generate control signals**
- **The execution unit or data processing unit contains a set of registers for storing data and an Arithmetic and Logic Unit (ALU) for execution of arithmetic and logical operations. In addition CPU may have some additional registers for temporary storage**

CPU Components

- **Control Unit (CU):** **Controls the operation of the CPU and hence the computer**
- **Arithmetic and Logic Unit (ALU):** **Performs computer's data processing functions.**
- **Register:** **Provides storage internal to the CPU.**
- **CPU Interconnection: communication among the control unit, ALU, and register.**

Inside the CPU



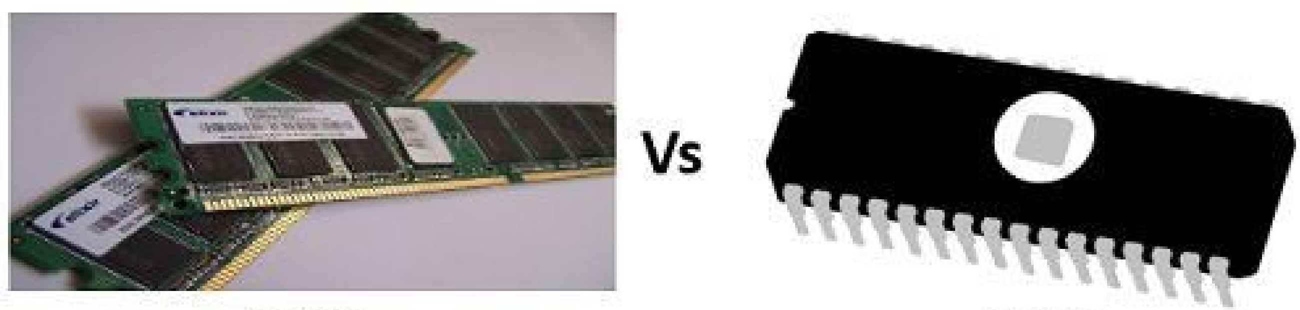
Nehalem : Intel's core i7 microarchitecture

Storage elements – Primary Memory

Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semi conductor memories

Volatile Memory : RAM (Random Access Memory).

Non-Volatile Memory : ROM (Read only Memory), PROM (Programmable ROM) EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

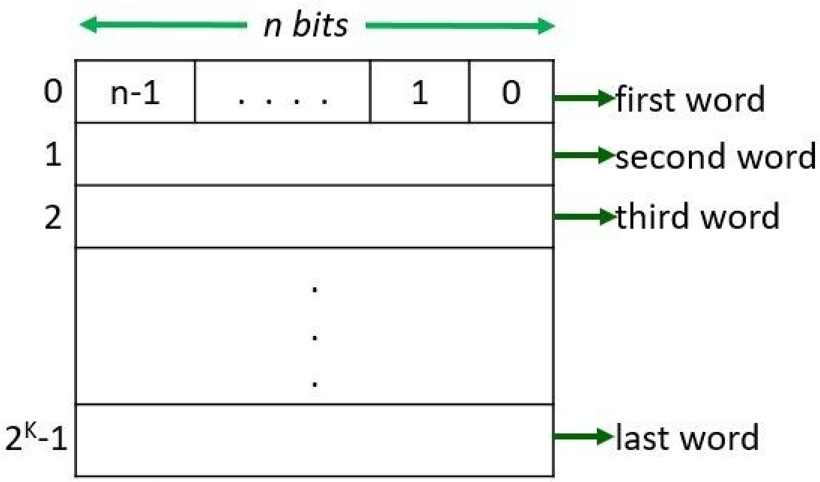


RAM
Random Access Memory

ROM
Read only Memory

Memory Organization

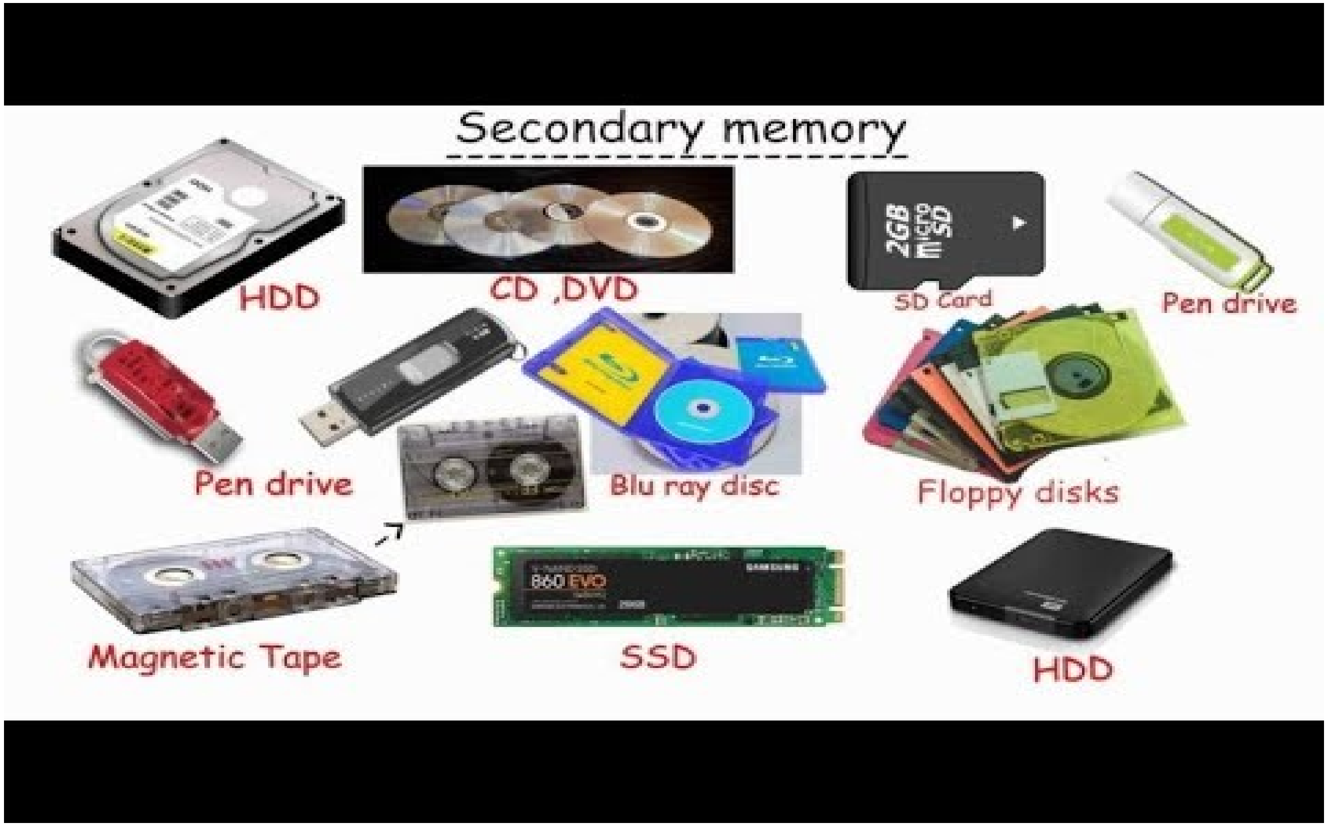
- Memory consists of millions of storage cells each storing a bit (0/1) of information
- Bits are held in groups of fixed size - requires basic operation
- Each *n-bit* group is called a *Word (n* describes the word length *)*
- Memory is a collection of words
- Each location of memory has an unique address



Memory Words

Storage Elements – Secondary Memory

Secondary memories are non volatile memory and it is used for permanent storage of data and program. Example of secondary memories: Hard Disk, USB Drives, CD-ROM



Input Unit

Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction

Output Unit

Used to Provide results to the user.

Data from main storage in transferred to the output units under control of CPU output instructions

How does the computer work ?

- Computer needs to be programmed to do such tasks
- Programming is the process of writing instructions in a language that can be understood by the computer so that a desired task can be performed by it
- Program: sequence of instructions to do a task, computer processes the instructions sequentially one after the other
- Software: programs for doing tasks on computers
- CPU understands machine language
  - Different strings of 0’s and 1’s (Hard to remember)
  - Mnemonic names of the strings - Instruction set
- Alternate way to instruct CPU – use high level language

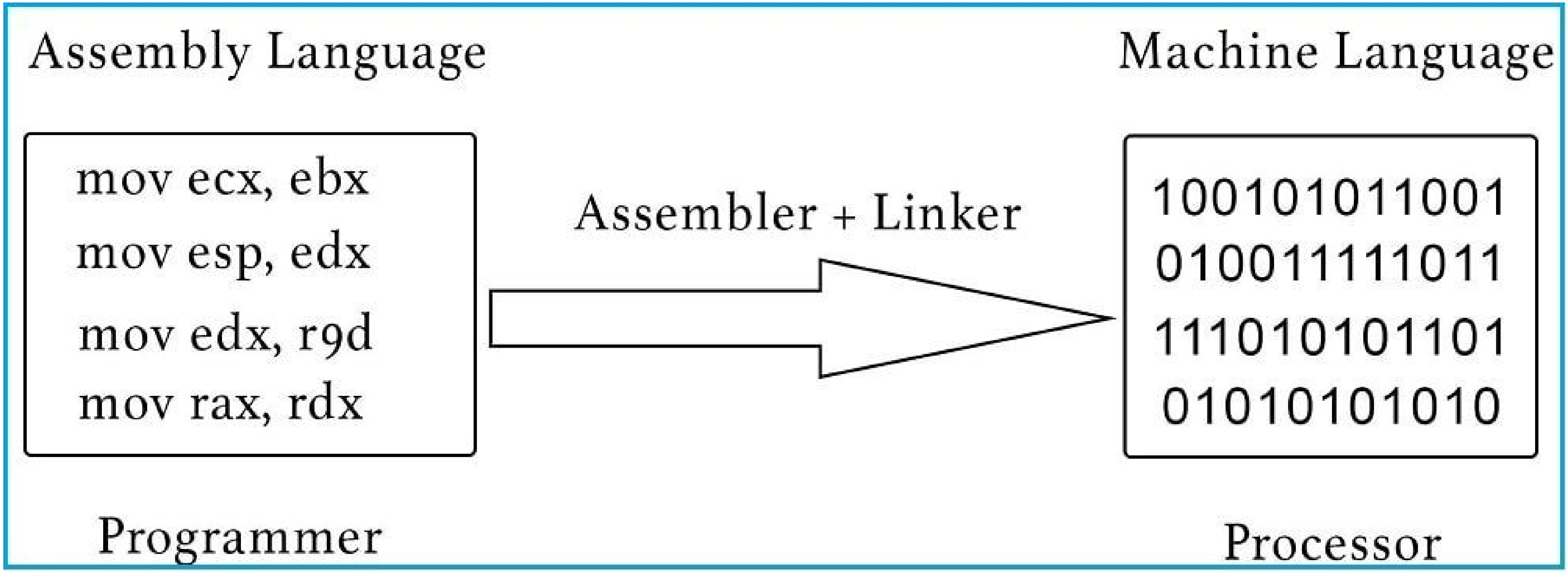
Example of machine-language

Here's what a program-fragment looks like:

10100001 10111100 10010011 00000100
00001000 00000011 00000101 11000000
10010011 00000100 00001000 10100011
11000000 10010100 00000100 00001000

It means: $z = x + y;$

Assembly Language



Programming a Computer in the language of the Computer : Assembly Language

Instruction set :

Start

Read M

Write M

Load Data, M

Copy M1,M2

Add M1, M2, M3 Sub

M1, M2, M3, Compare

M1, M2, M3, Jump L

J_Zero M,L

Halt

Program

0: Start

1: Read 10

2: Read 11

3: Add 10, 11, 12

4: Write 12

5: Halt

Programming a Computer using High Level Language

- **Instruction set of different CPUs are different**
  - Need to write different programs for computers with different types of CPUs even to do the same thing
- **Solution – High Level Language (C, C++, Java, …….)**
  - CPU neutral – One program for many
  - Compiler – to convert from high level program to low level program that the computer understands

High Level Program

Variables x, y;

Begin

Read (x);

Read (y);

If (x >y) then Write (x)

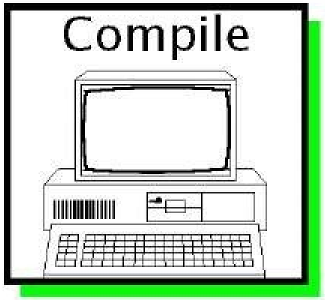
else Write (y); End.

Low Level Program

0: Start
1: Read 20
2: Read 21
3: Compare 20, 21, 22
4: J_Zero 22, 7
5: Write 20
6: Jump 8
7: Write 21
8: Halt

HLL – Assembly – Machine Code

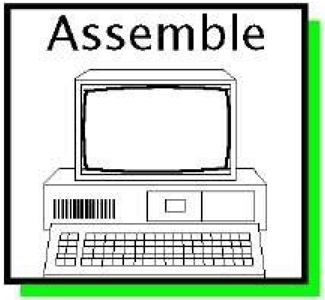
```
while(n>0)
{
sum = sum + n;
--n;
}
```




```
L28   movf    _n,f
      btfsc   STATUS,Z
       goto   L41
      movf    _n,f
      addwf   _sum,f
      btfsc   STATUS,C
       incf   _sum+1,f
      decf    _n,f
      goto    L28
L41
```

(a) First, compile to assembly-level code.

```
L28   movf    _n,f
      btfsc   STATUS,Z
       goto   L41
      movf    _n,f
      addwf   _sum,f
      btfsc   STATUS,C
       incf   _sum+1,f
      decf    _n,f
      goto    L28
L41
```




```
0000100010010011
0001100100000011
0010100000001111
0000100000010011
0000100000010011
0000011110010100
0001100000000011
0000101010010101
0111100000000111
```

(b) Second, assemble-link to machine code.

Levels of Representation

High Level Language Program

Compiler

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Assembly Language Program

Assembler

```
lw   $15,   0($2)
lw   $16,   4($2)
sw   $16,   0($2)
sw   $15,   4($2)
```

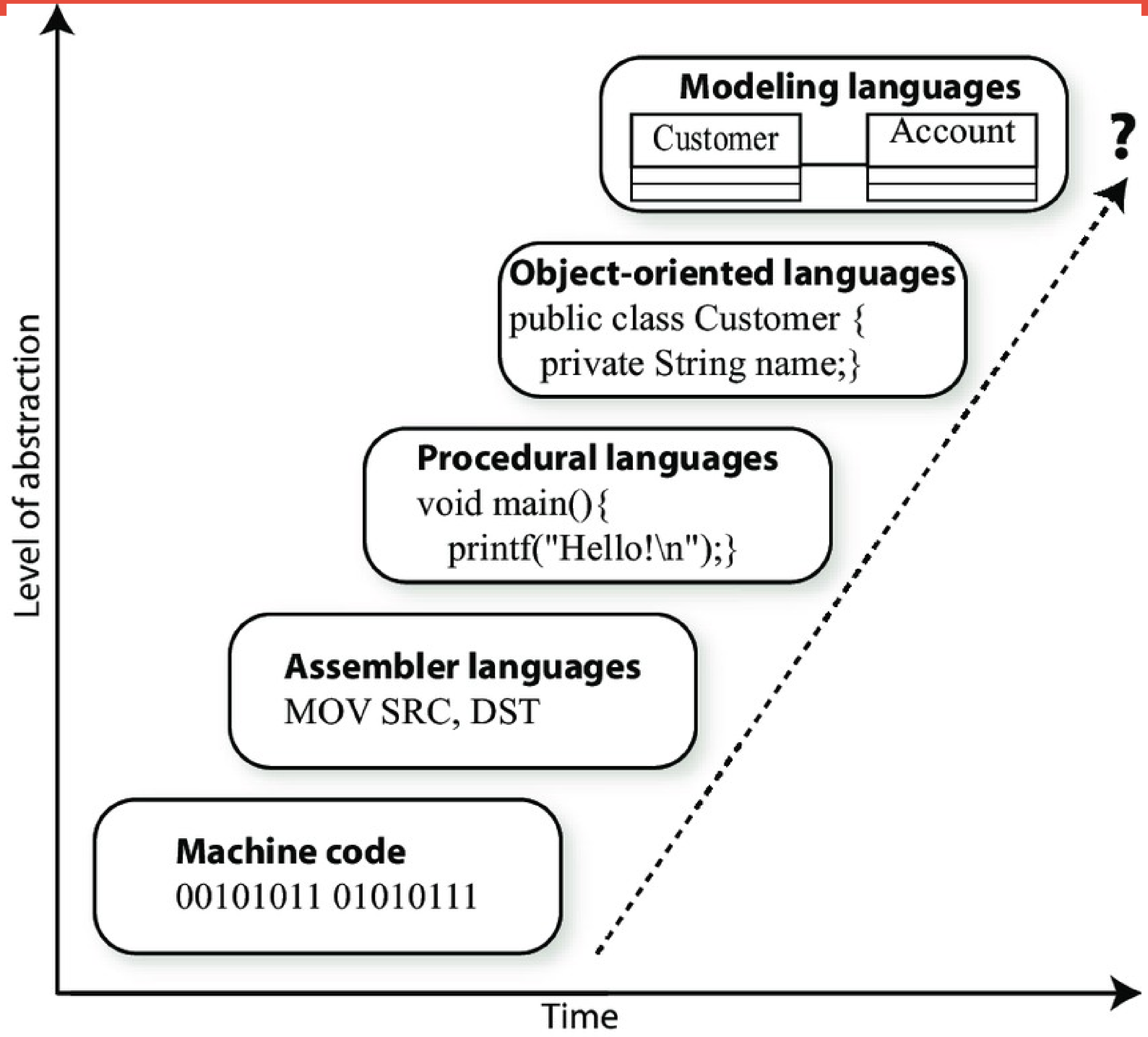
Machine Language Program

Machine Interpretation

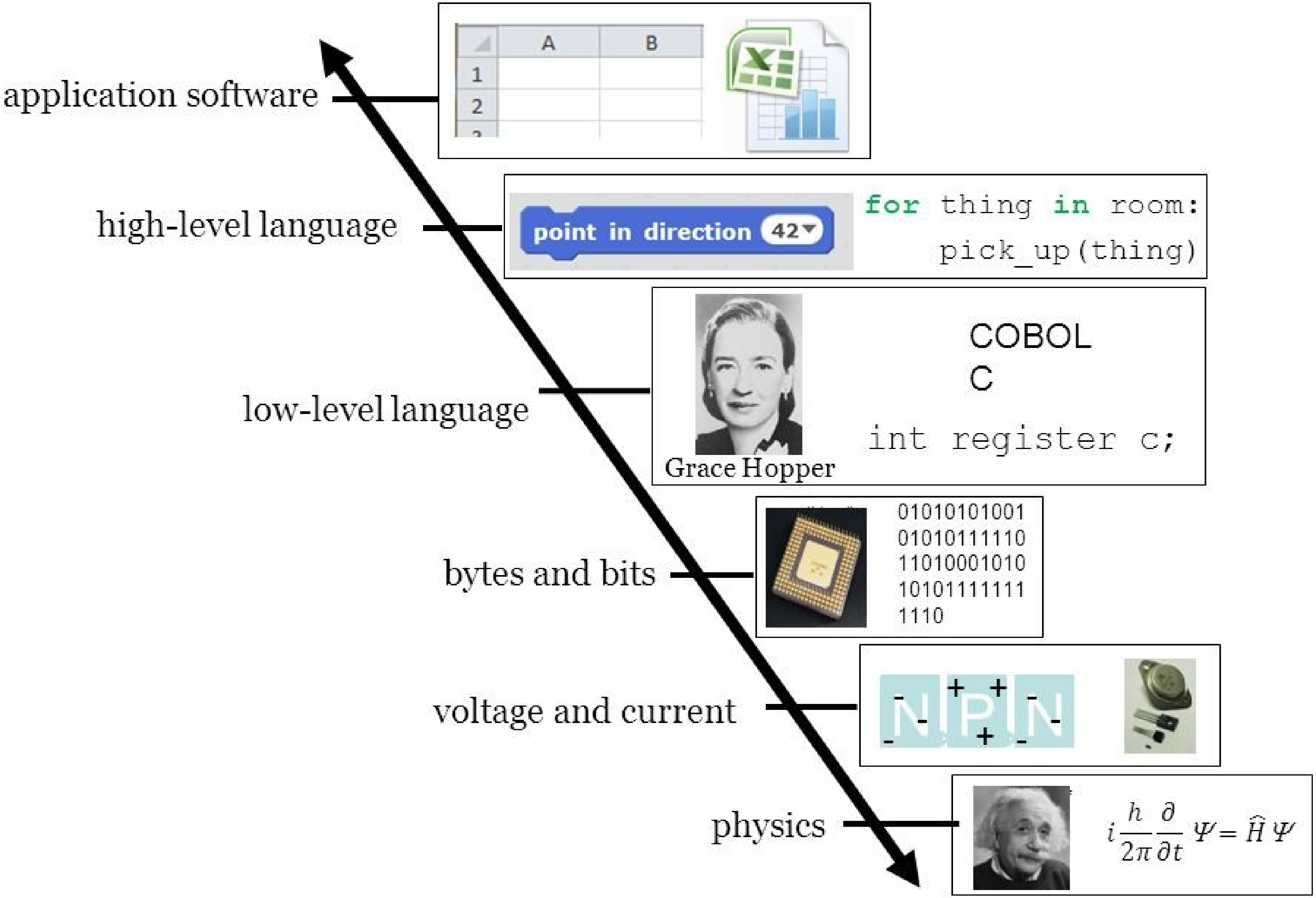
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Control Signal Spec

Computer is all about Abstraction



The Ladder of Abstraction



How Do We Solve a Problem by a Computer ?

Ans : Through a systematic sequence of transformation between layers of abstraction

Software design: **Problem** → **Algorithm**

Language: **Algorithm** → **Program**

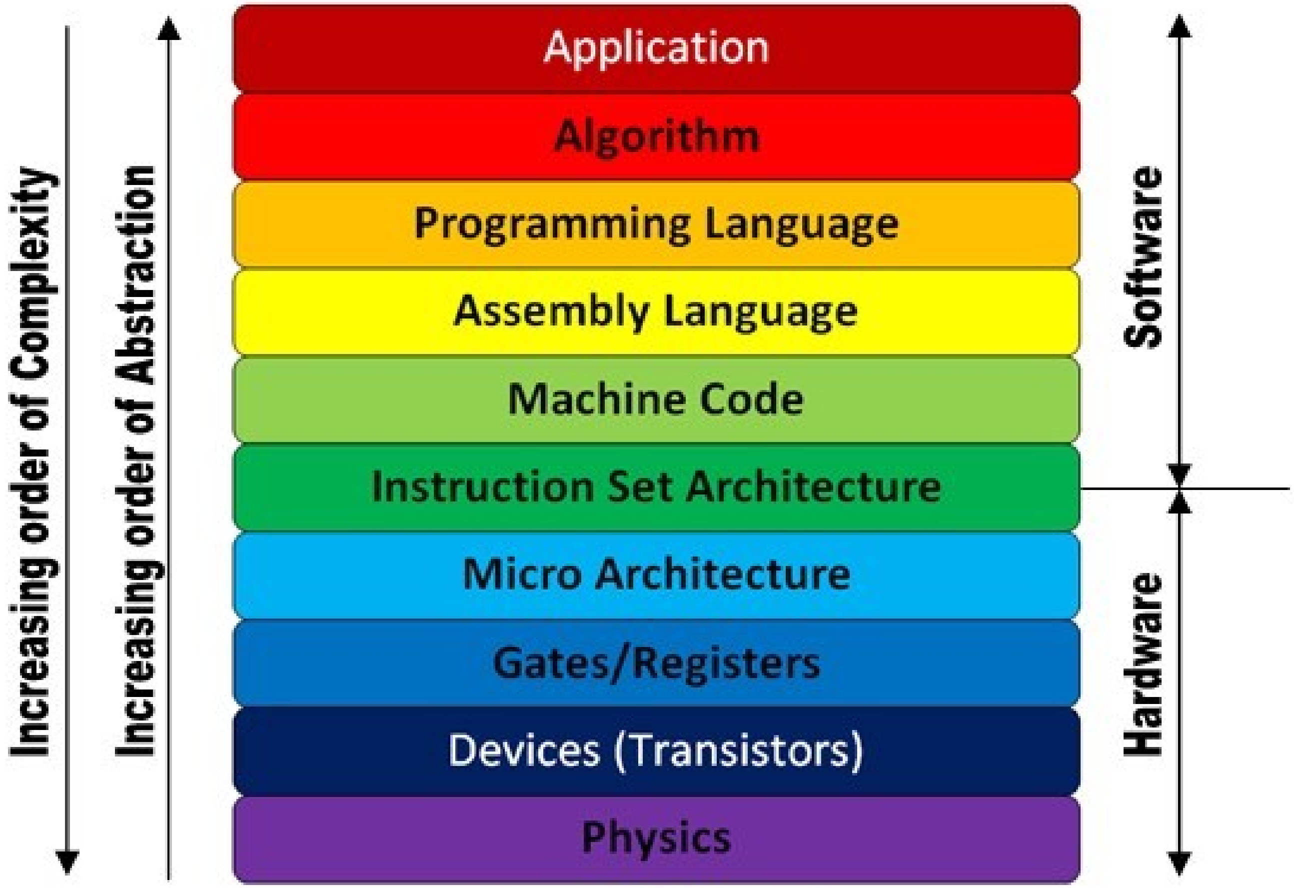
Language to m/c instructions: **Program** → **Instruction Set Architecture(ISA)**

Structures to implement ISA: **ISA** → **Micro architecture**

Gates: **Micro architecture** → **Circuits**

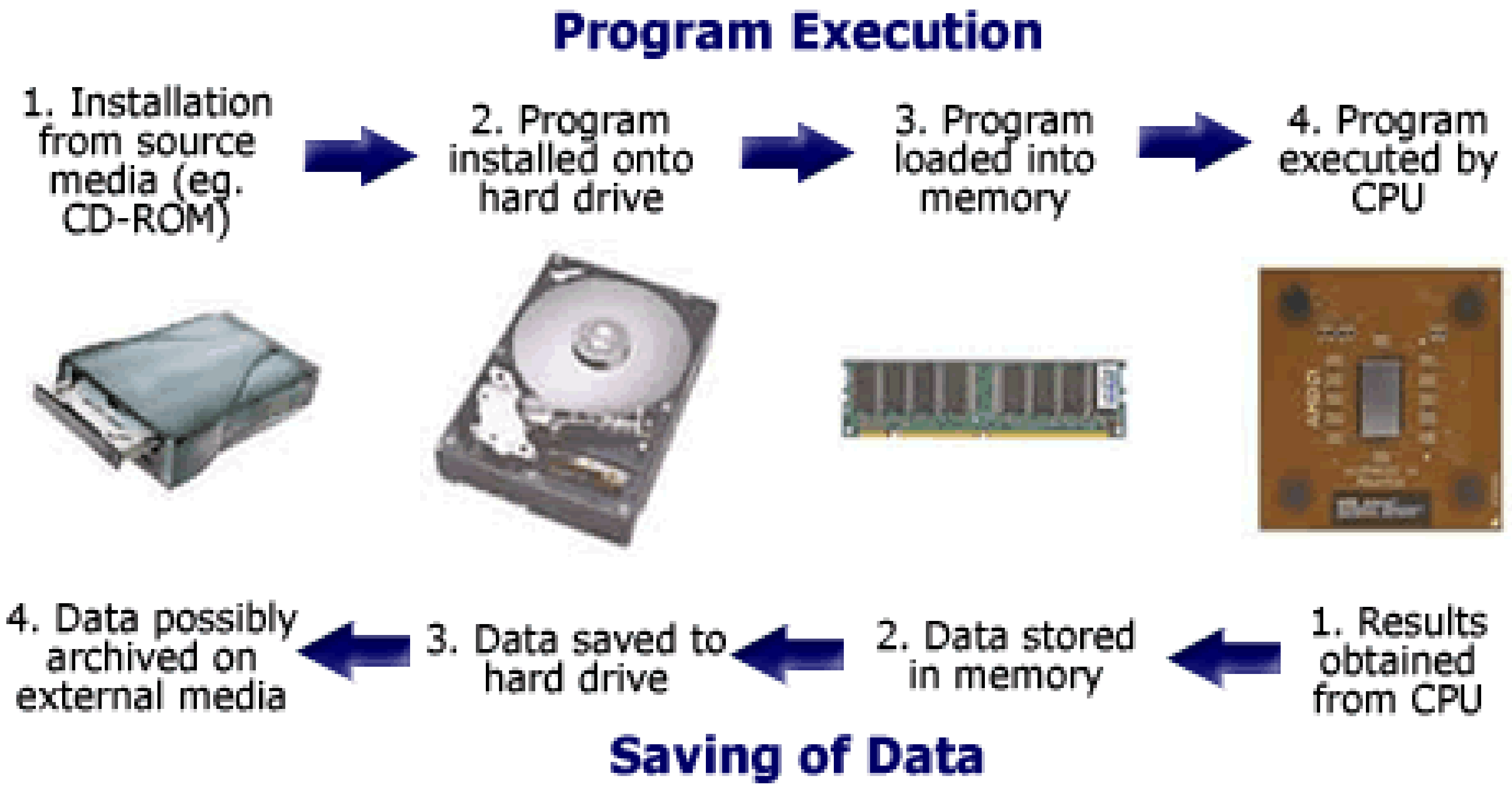
Circuits → **Devices (** develop and manufacture components **)**

Layers of Absttraction

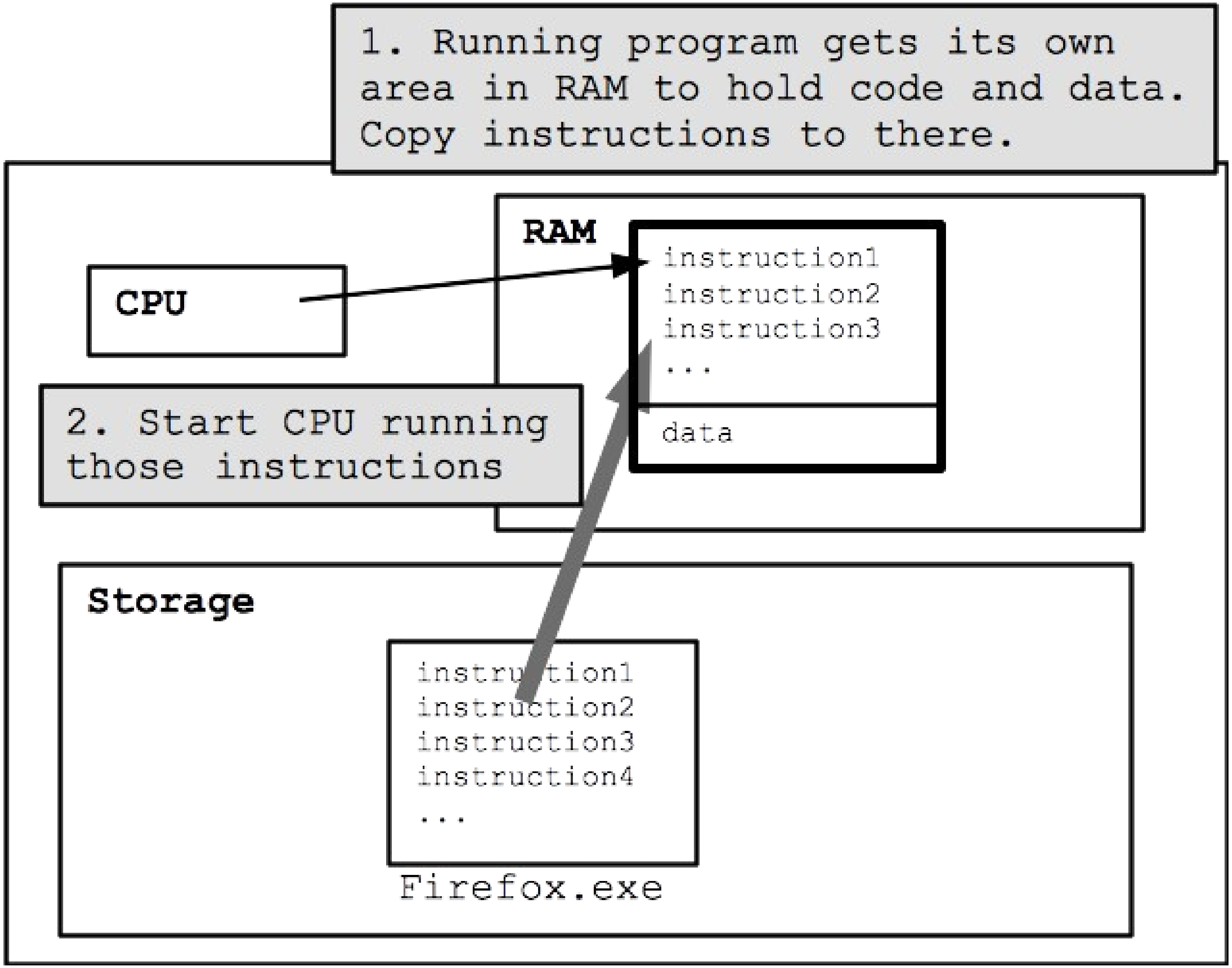


Layers of Abstraction

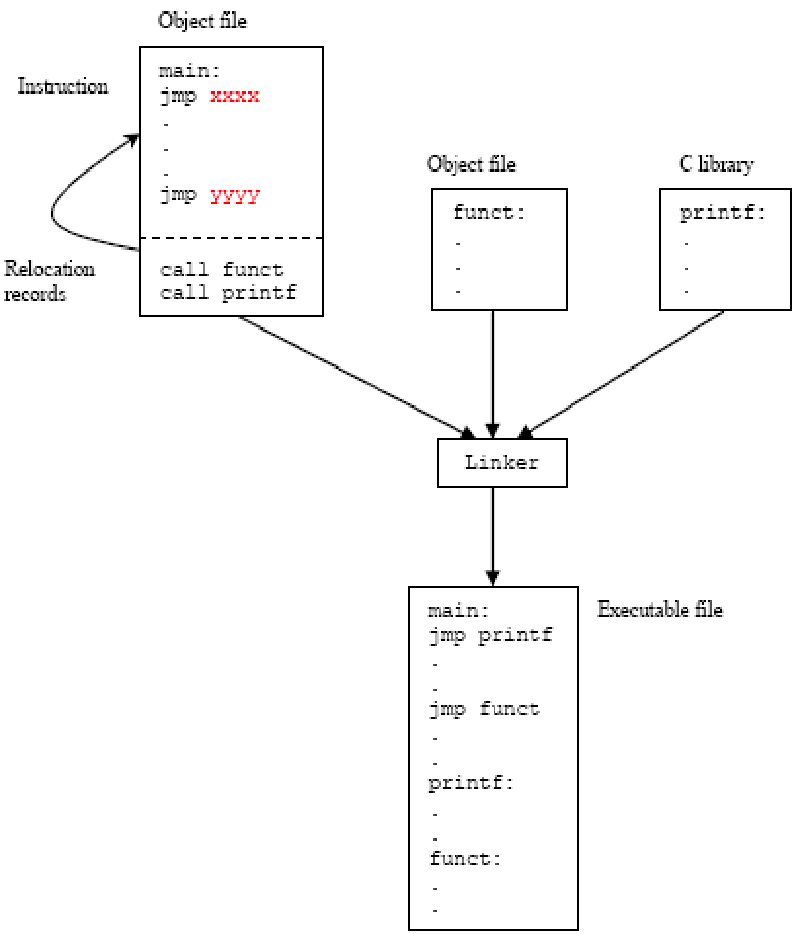
How are programs executed by the computer?



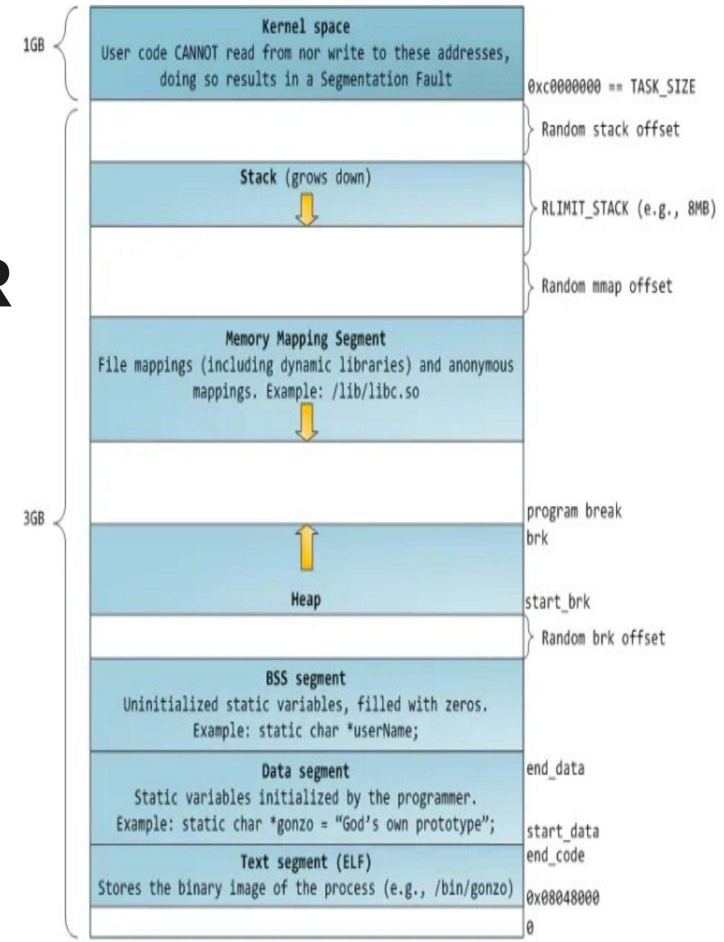
What happens after the machine code a program is generated?



Loading a Program in Memory



LOADER

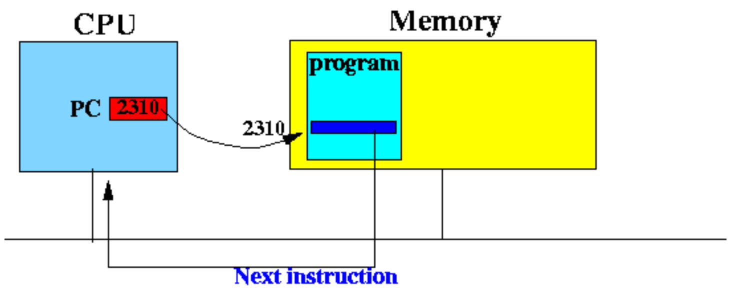


Memory Layout of a C code

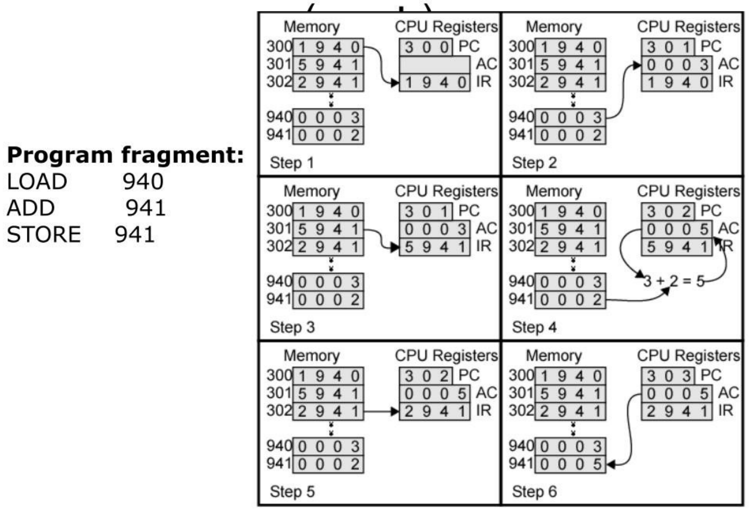
```c
#include <stdio.h>                                                          //     HIGHER   ADDRESS
#include <malloc.h>                                                         // +----------------------+
                                                                            // |       Unmapped       |
                                                                            // |----------------------|-----+
                                                                            // |        main()        |     |
char str[] = "Hi!"; // Initialized read-write area of DATA segment          // |         ptr          |     |
const int x = 1;    // Uninitialized DATA segment                           // |----------------------|     |--Stack segment
int i;                                                                      // |        func()        |     |
                                                                            // |          a           |     |
                                                                            // |----------------------|     |
                                                                            // |          |           |
void func()                                                                 // |          |           |
{                                                                           // |          V           |
    static int var = 0; // Initialized DATA segment                         // |                      |     |
    int a;              // stack fram segment                               // |          ^           |     |
}                                                                           // |          |           |     |--Heap segment
                                                                            // |          |           |     |
                                                                            // |         *ptr         |     |
                                                                            // |----------------------|-----+
int main()                                                                  // |          i           |  Uninitialized DATA segment
{                                                                           // |----------------------|
    char *ptr = (char *)malloc(sizeof(char)); // Heap segment               // |        var=0         |
    func();                                   // stack frame                // |         x=1          |  Initialized DATA segment
    return 0;                                                               // |    str[] = "Hi!"     |
}                                                                           // |----------------------|
                                                                            // |                      |
                                                                            // |      Executable      |  Text segment
                                                                            // |     Instructions     |
                                                                            // |                      |
                                                                            // +----------------------+
                                                                            //      LOWER   ADDRESS
```

What Happens After Loading?

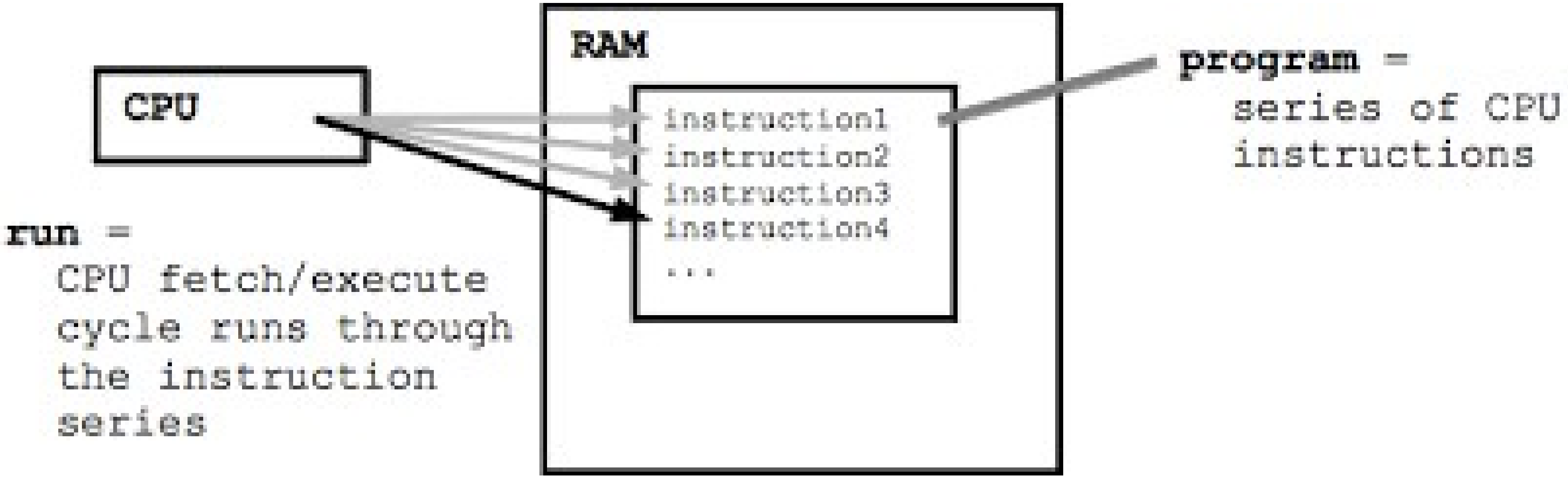
- CPU fetches the compiled code sequentially from memory to its registers
- Executes the code using Arithmetic and Logic Unit on instructions from the Control Unit



Example of Program Execution

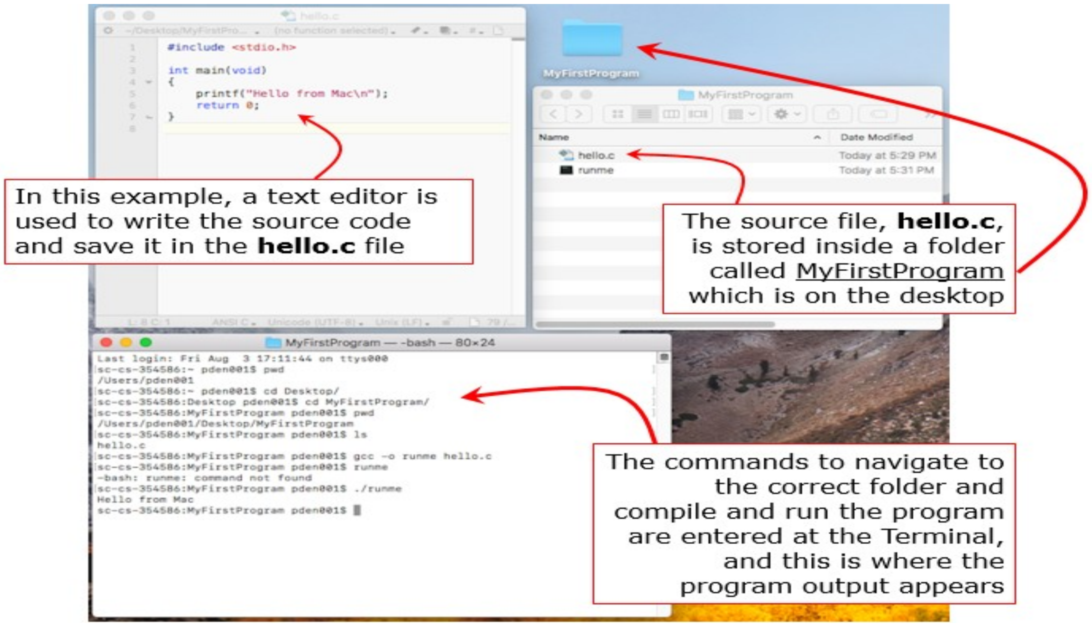


CPU Execution

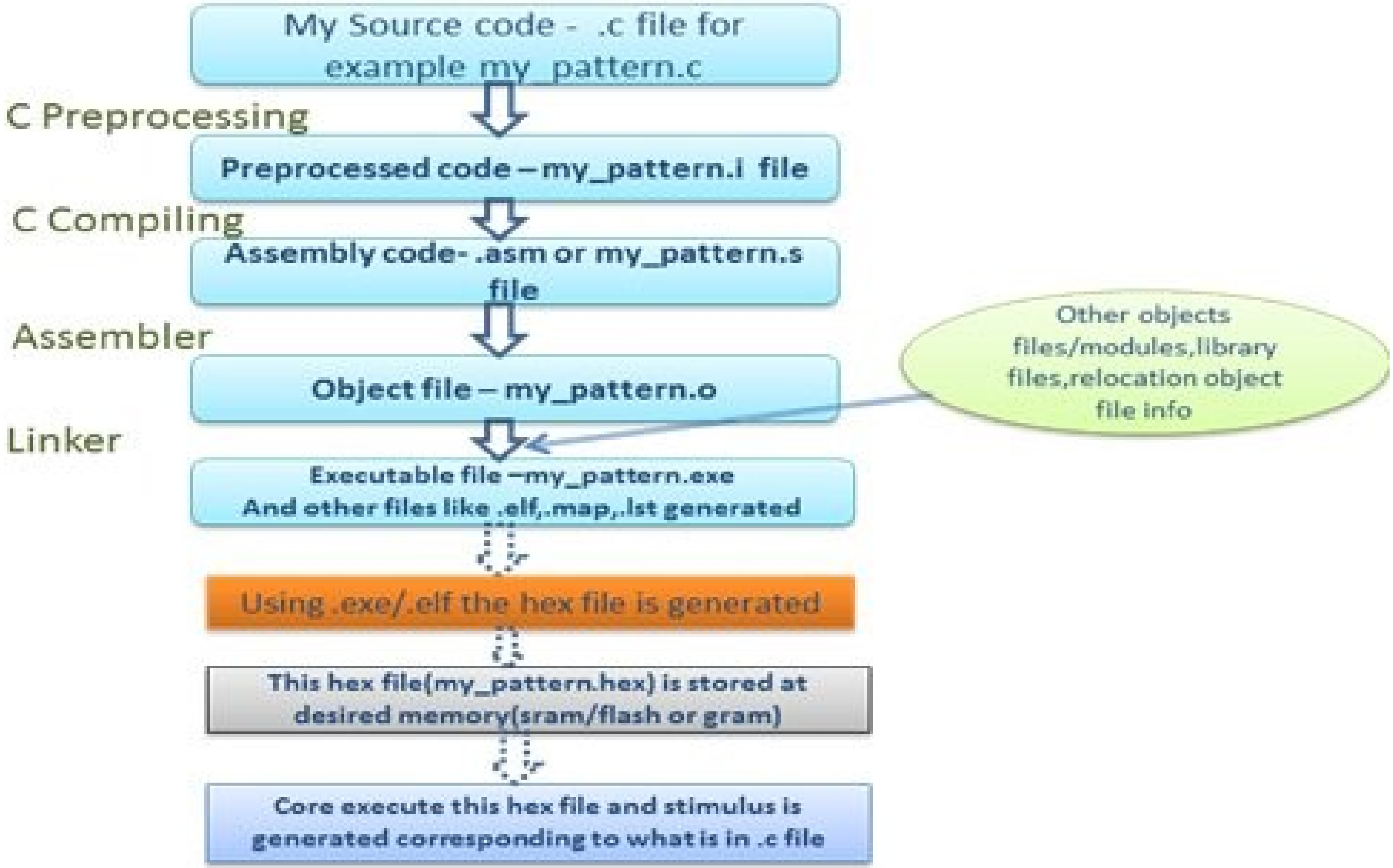


Program execution Steps

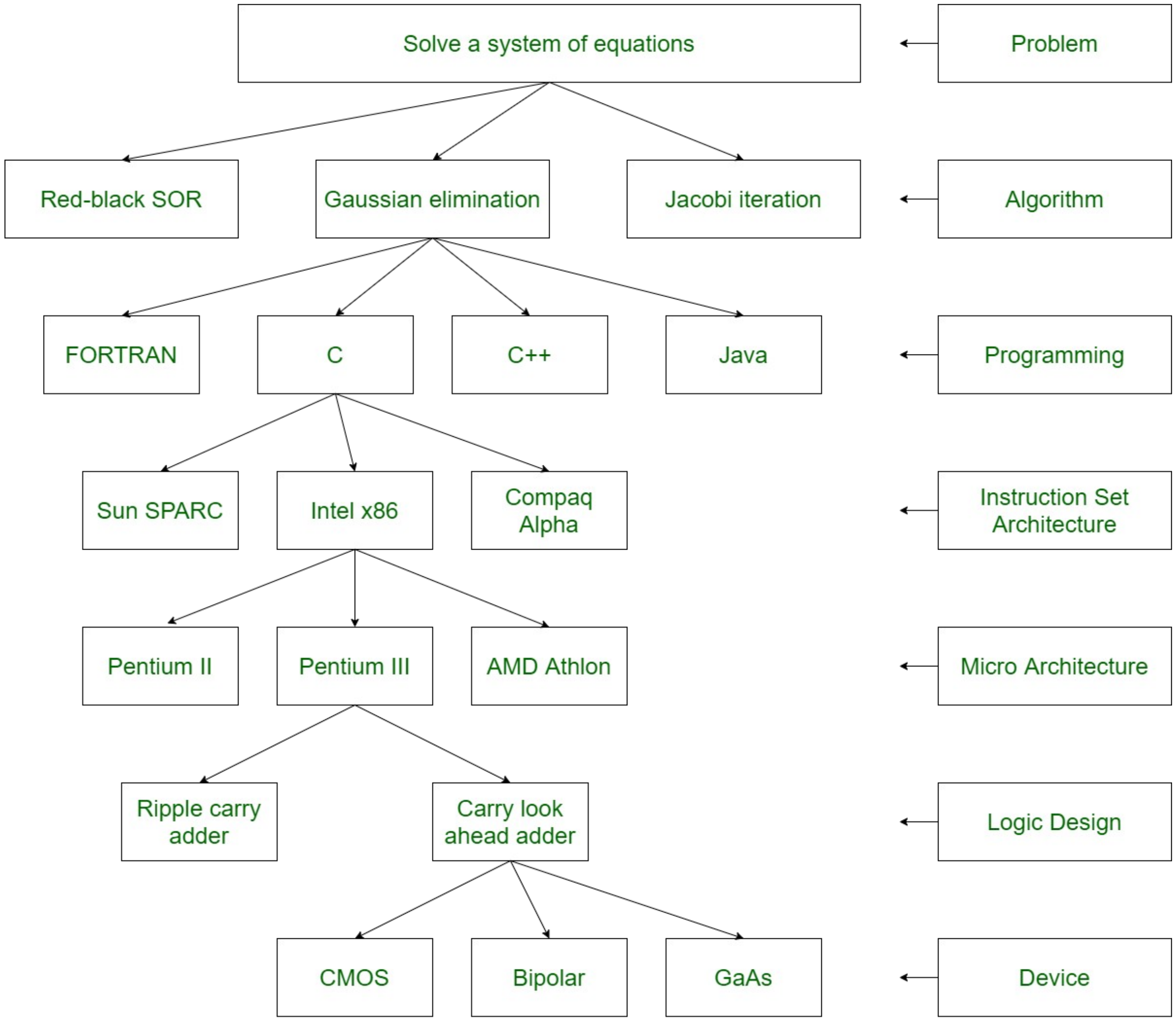
- **Write a program in high level language (Say in C)**
  - *Use a text editor to write the source code (save the file with **.c** extension*
- **Compile the program using a C compiler (gcc compiler)**
  - On the terminal : *gcc **<file.c>** -o **<output file name>***
- **Run the program (Ask the computer to execute it)**
  - **On the terminal : *./ <output file name>***



Compilation Steps



Choices of implementation at different layers



The Hierarchical nature of computer system

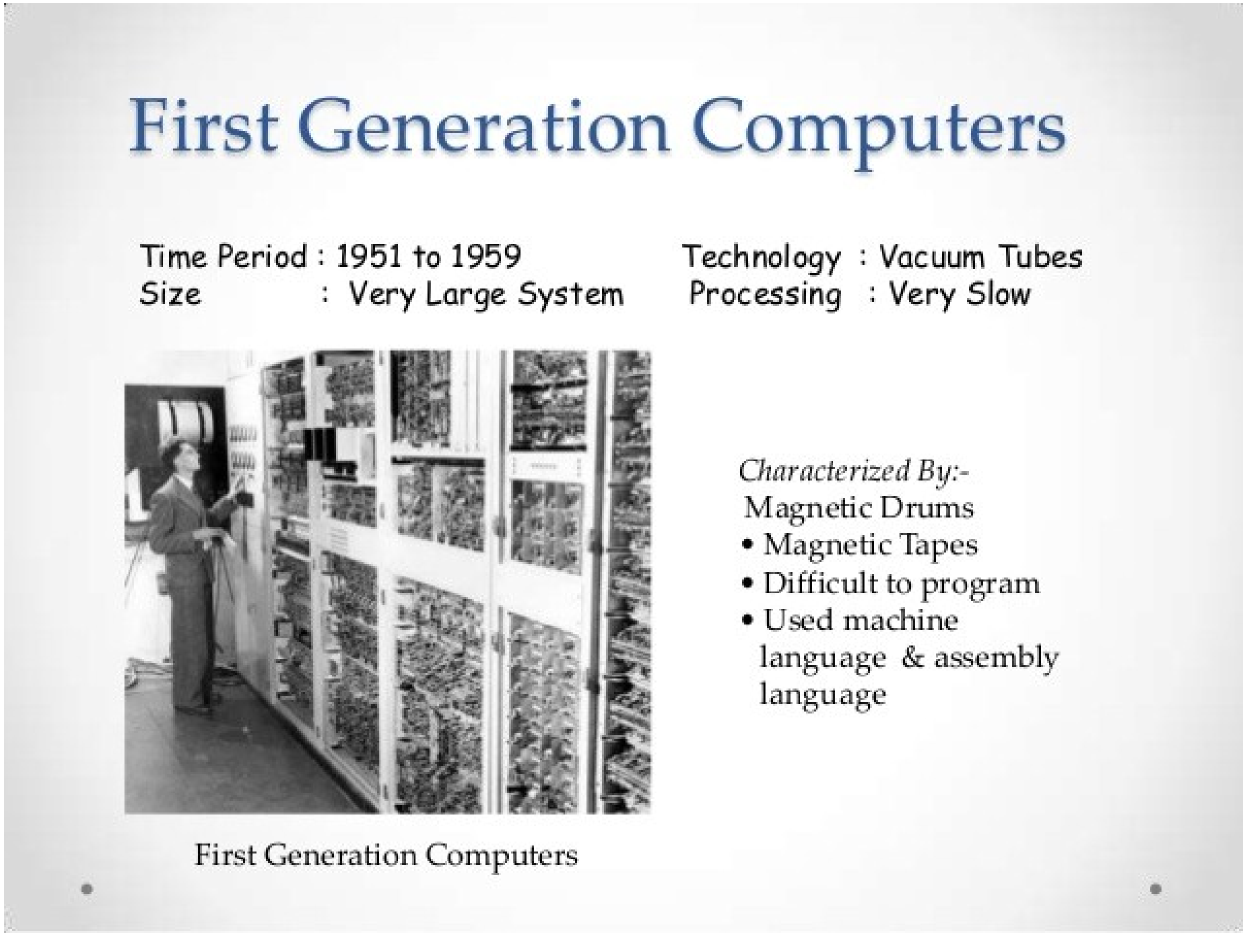
- **A hierarchical system is a set of interrelated subsystems, each in turn, hierarchical in structure; until at the lowest level we have elementary subsystems**
- **A computer is a complex hierarchical system**
- **The hierarchical nature of complex systems is essential to both their design and their description. The designer need only deal with a particular level of the system at a time**
- **At each level, the system consists of a set off *components and their interrelationships*.**
- **The behaviour at each level depends only on a simplified, abstracted characterization of the system at the next lower level.At each level,, the designer is concerned with structure and function:**
- **Structure: The way in which the components are interrelated.**
- **Function : The operation of each individual component**

What is Architecture and Organization?

Computer Architecture vs Computer Organization

- Computer organization
  - Encompasses all physical aspects of computer systems.
  - E.g., circuit design, control signals, memory types.
  - *How does a computer work?*
- Computer architecture
  - Logical aspects of system implementation as seen by the programmer.
  - E.g., instruction sets, instruction formats, data types, addressing modes.
  - *How do I design a computer?*

History of Computer – Past to Present



Second generation of computers

Second Generation Computers

Time Period : 1959 to 1963
Size : Smaller

Technology : Transistors
Processing : Faster


Second Generation Computers

Characterized By:-

- Magnetic Cores
- Magnetic Disk
- Used high level language
- Easier to program

Third generation computers

Third Generation Computers

Time Period	: 1963 to 1975
Technology	: ICs (Integrated Circuits) Incorporated many transistors & electronic circuits on a single chip
Size	: Small as compared to 2nd generation computers
Processing	: Faster then 2nd generation computers

IC (Integrated Circuit)

Characterized by:-

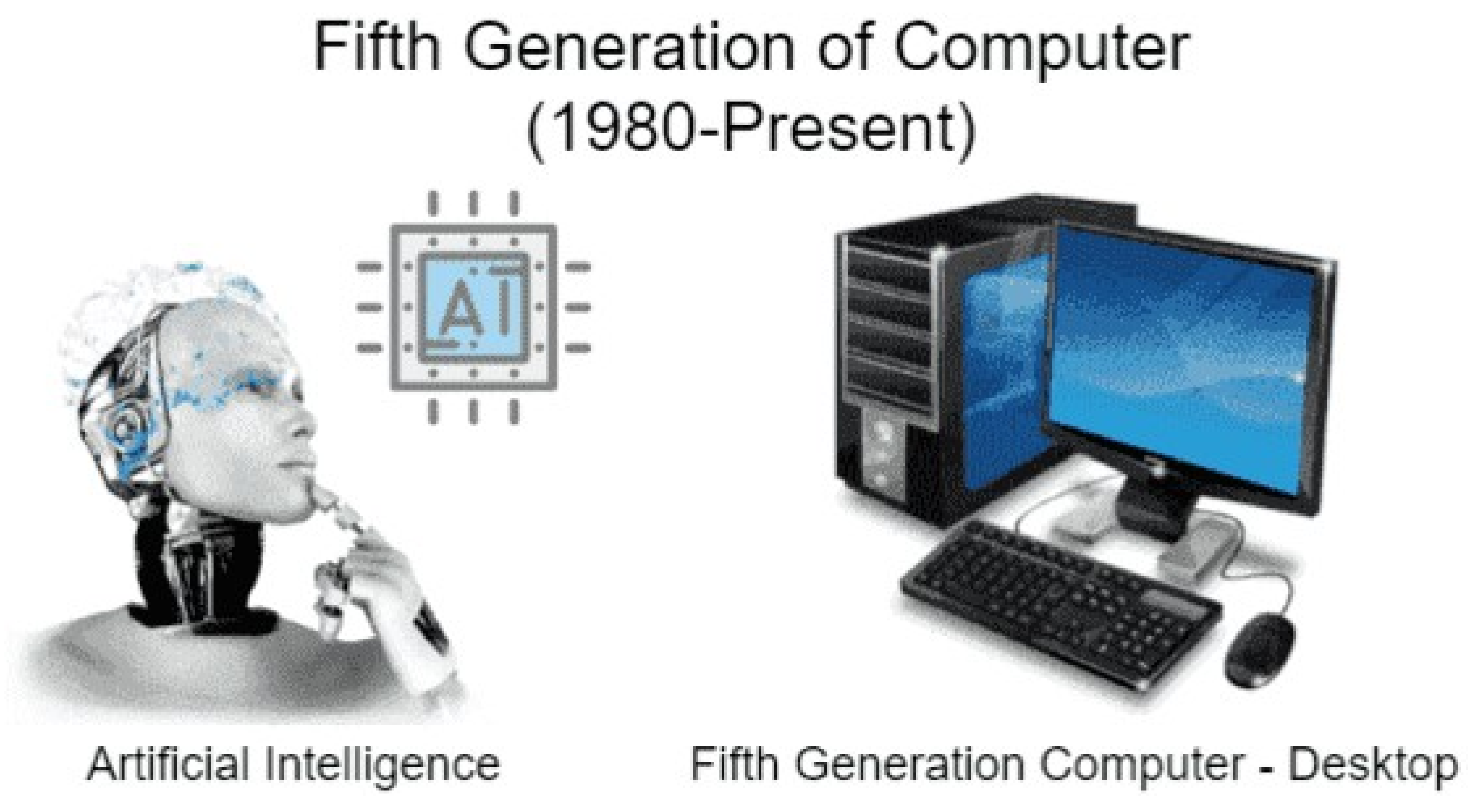
- Minicomputers accessible by multiple users from remote terminals.

Fourth generation of Computers

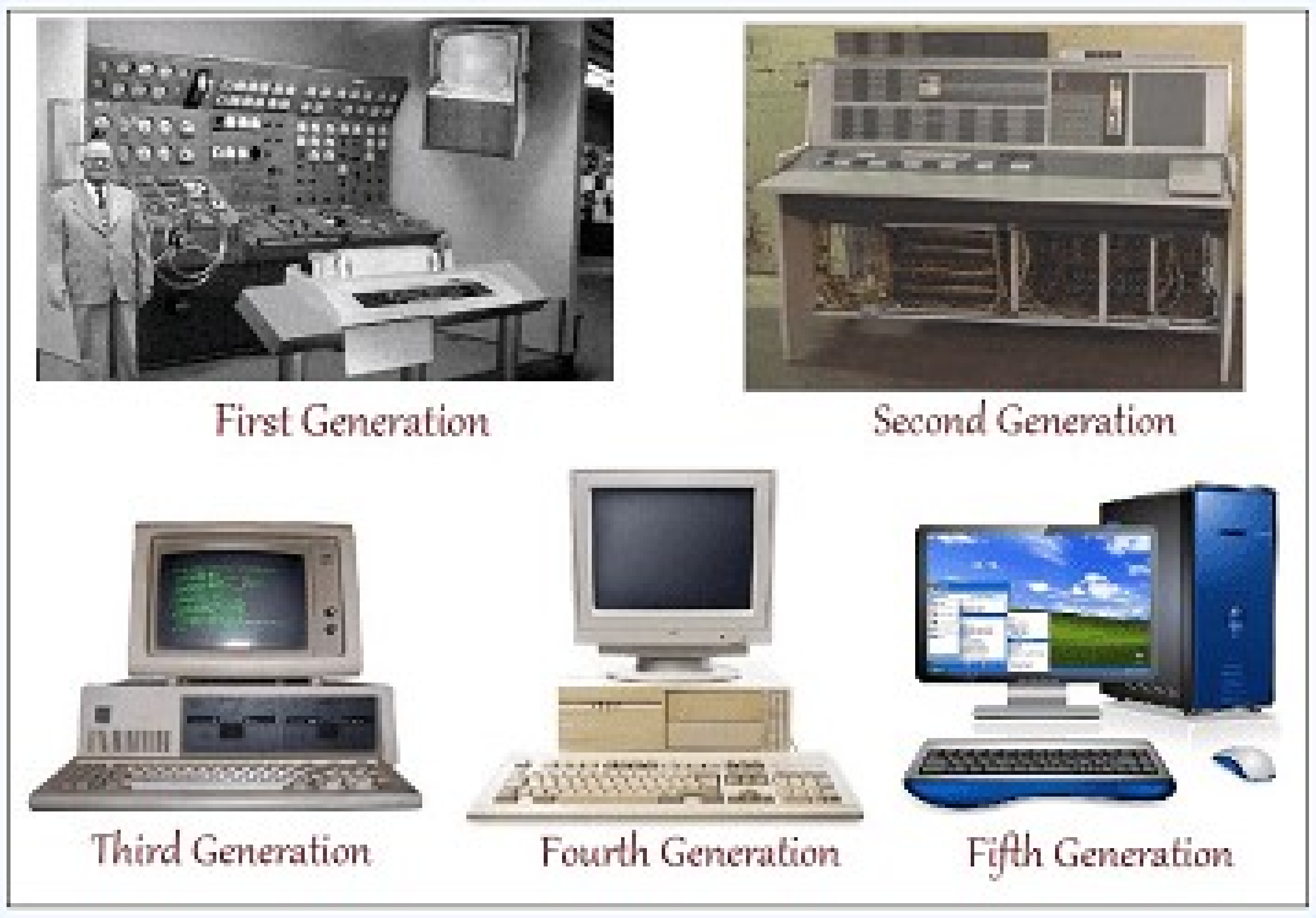
Fourth generation (1971-1980) VLSI microprocessor based

- Storage – Semiconductor memory, 1000 MB disks
- Software – FORTRAN-77,PASCAL,COBOL 74
- Applications – personal computers,graphics oriented system,

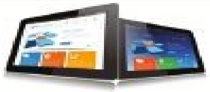
Fifth generation of computers



The Generations



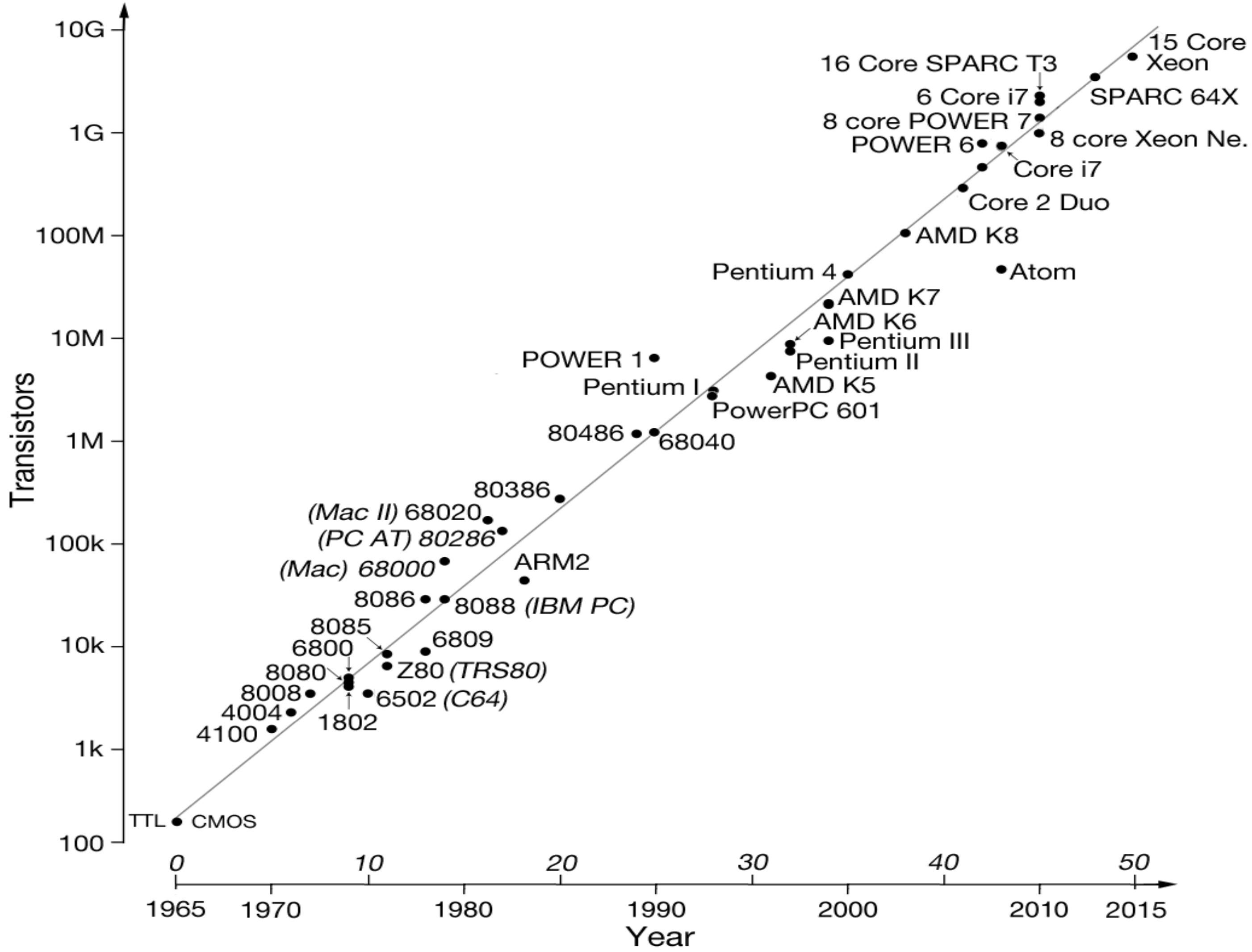
Present day computing

PHOTO	COMPUTING DEVICE	USES	PROCESSING POWER	MOBILITY
	Laptops	Laptop computers are used for almost anything, from document processing in an office environment, to graphic design and video editing, to browsing the internet and playing games. However, laptop computers are easily moved around allowing you to work anywhere and anytime.	Medium to high processing power	Fully mobile
	Desktop computers	A desktop computer's uses are exactly the same as those of a laptop except that a desktop is not mobile.	Medium to high processing power	Minimal mobility
	Smart phones	Smartphones are better than desktops at tasks that require a very mobile device, like taking photos, setting alarms, navigating the roads, making calls and sending and receiving short messages.	Medium to low processing power	Excellent mobility
	Tablets	Info on keyboard, screen size and applications. Reading of books is better on a tablet than on a smartphone.	Medium processing power	Excellent mobility
	Servers	Servers are designed for managing networks, providing access to specific files and hosting websites, as well as processing huge amounts of data.	High processing power	No mobility
	Embedded computers	Embedded devices are devices designed for a fixed purpose, whether that purpose is to wake you up in the morning, control the temperature of the air conditioning or refrigerator, or any navigation system.	Low processing power	Varies depending on the device

Moore's Law

- **Increased density of components on chip**
- **Gordon Moore – co-founder of Intel**
- **Number of transistors on a chip will double every year**
- **Since 1970's development has slowed a little**
  - Number of transistors doubles every 18 months
- **Cost of a chip has remained almost unchanged**
- **Higher packing density means shorter electrical paths, giving higher performance**
- **Smaller size gives increased flexibility**
- **Reduced power and cooling requirements**
- **Fewer interconnections increases reliability**

Moore's Law

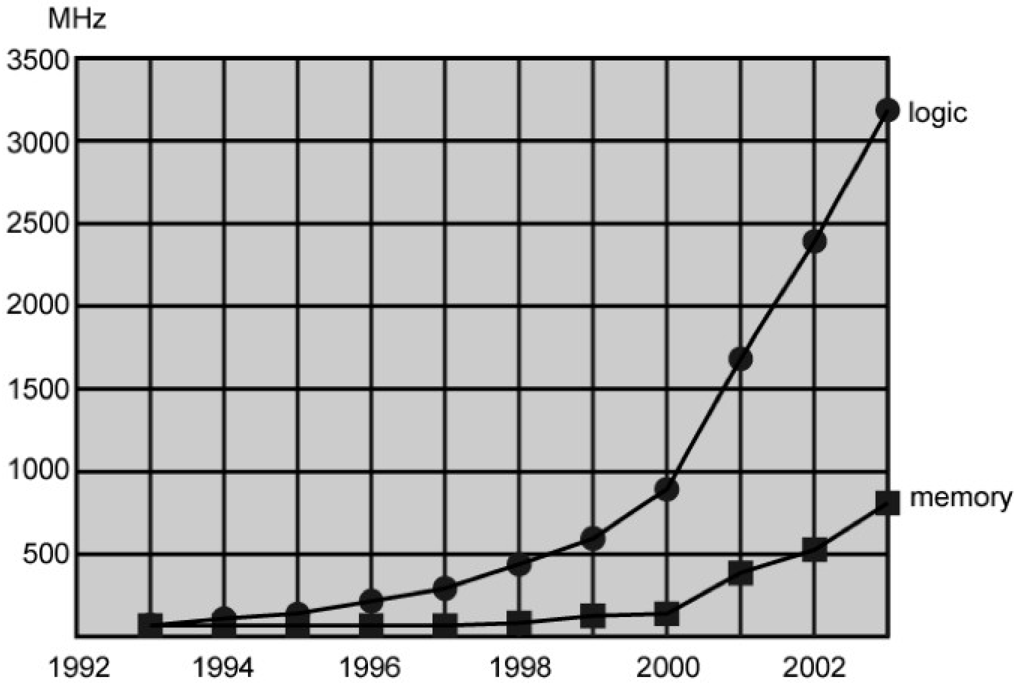


Transistors
10G
1G
100M
10M
1M
100k
10k
1k
100
15 Core Xeon
16 Core SPARC T3
6 Core i7
SPARC 64X
8 core POWER 7
8 core Xeon Ne.
POWER 6
Core i7
Core 2 Duo
AMD K8
Pentium 4
Atom
AMD K7
AMD K6
Pentium III
POWER 1
Pentium II
Pentium I
AMD K5
PowerPC 601
80486
68040
80386
(Mac II) 68020
(PC AT) 80286
(Mac) 68000
ARM2
8086
8088 (IBM PC)
8085
6800
6809
8080
Z80 (TRS80)
8008
6502 (C64)
4004
1802
4100
TTL
CMOS
0
10
20
30
40
50
1965
1970
1980
1990
2000
2010
2015
Year

Performance Balance

Processor speed increased

- Memory capacity increased
- Memory speed lags behind processor speed



Logic – Memory performance gap

Solutions

Increase number of bits retrieved at one time

- Make DRAM “wider” rather than “deeper”

Change DRAM interface

- Cache

Reduce frequency of memory access

- More complex cache and cache on chip

Increase interconnection bandwidth

- High speed buses
- Hierarchy of buses

I/O Devices

Peripherals with intensive I/O demands

Large data throughput demands

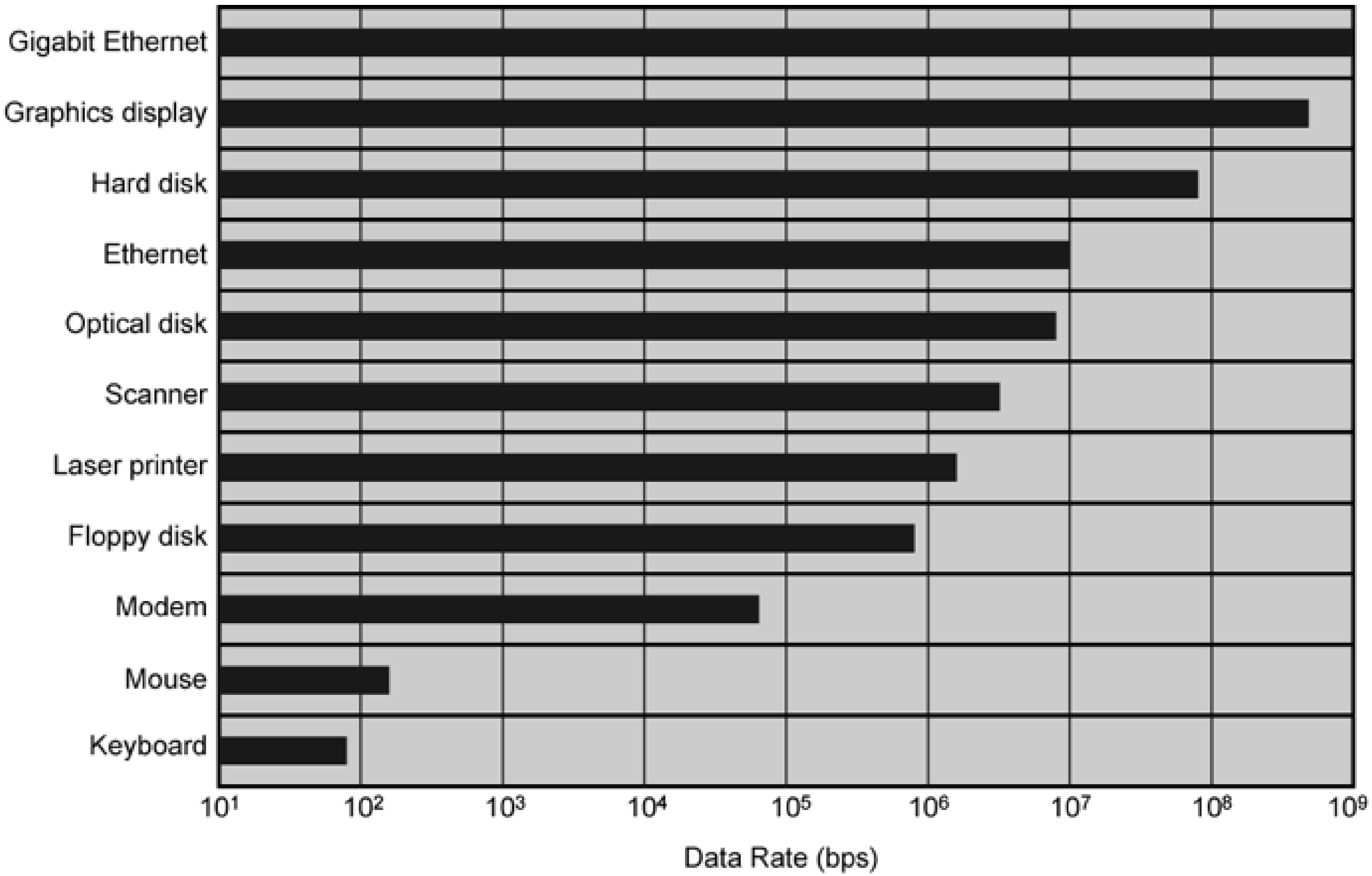
Processors can handle this

Problem moving data

Solutions:

- Caching
- Buffering
- Higher-speed interconnection buses
- More elaborate bus structures
- Multiple-processor configurations

I/O Device data rate



Improvement in Chip Organization and Architecture

Increase hardware speed of processor

- Fundamentally due to shrinking logic gate size
  - More gates, packed more tightly, increasing clock rate
  - Propagation time for signals reduced

Increase size and speed of caches

- Dedicating part of processor chip
  - Cache access times drop significantly

Change processor organization and architecture

- Increase effective speed of execution
- Parallelism

Problems with Clock Speed and Logic density

Power

- Power density increases with density of logic and clock speed
- Dissipating heat

RC delay

- Speed at which electrons flow limited by resistance and capacitance of metal wires connecting them
- Delay increases as RC product increases
- Wire interconnects thinner, increasing resistance
- Wires closer together, increasing capacitance

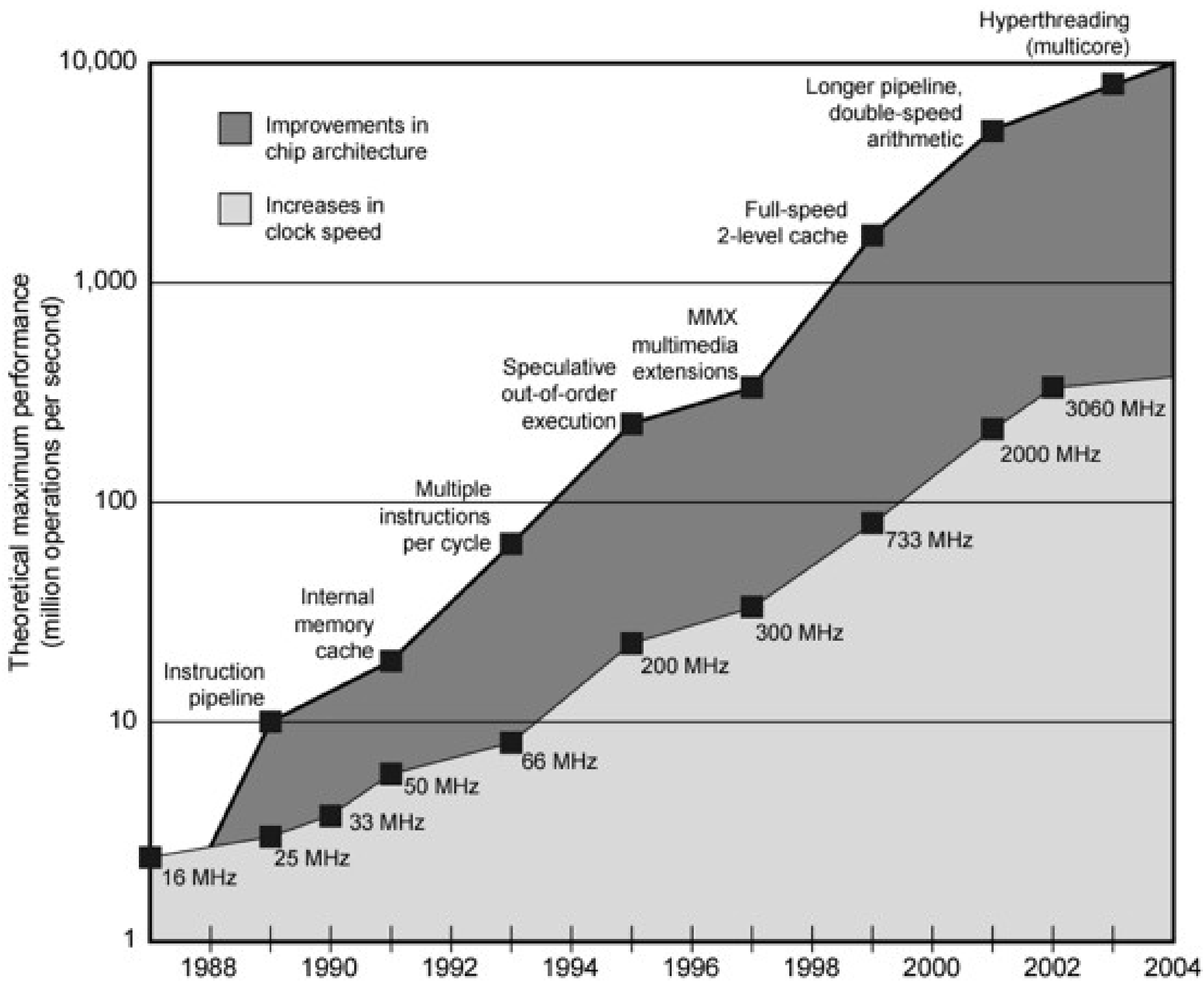
Memory latency

- Memory speeds lag processor speeds

Solution:

- More emphasis on organizational and architectural approaches

Intel Microprocessor Performance



Increased cache Capacity

Typically two or three levels of cache between processor and main memory

Chip density increased

- More cache memory on chip

Faster cache access

Pentium chip devoted about 10% of chip area to cache

Pentium 4 devotes about 50%

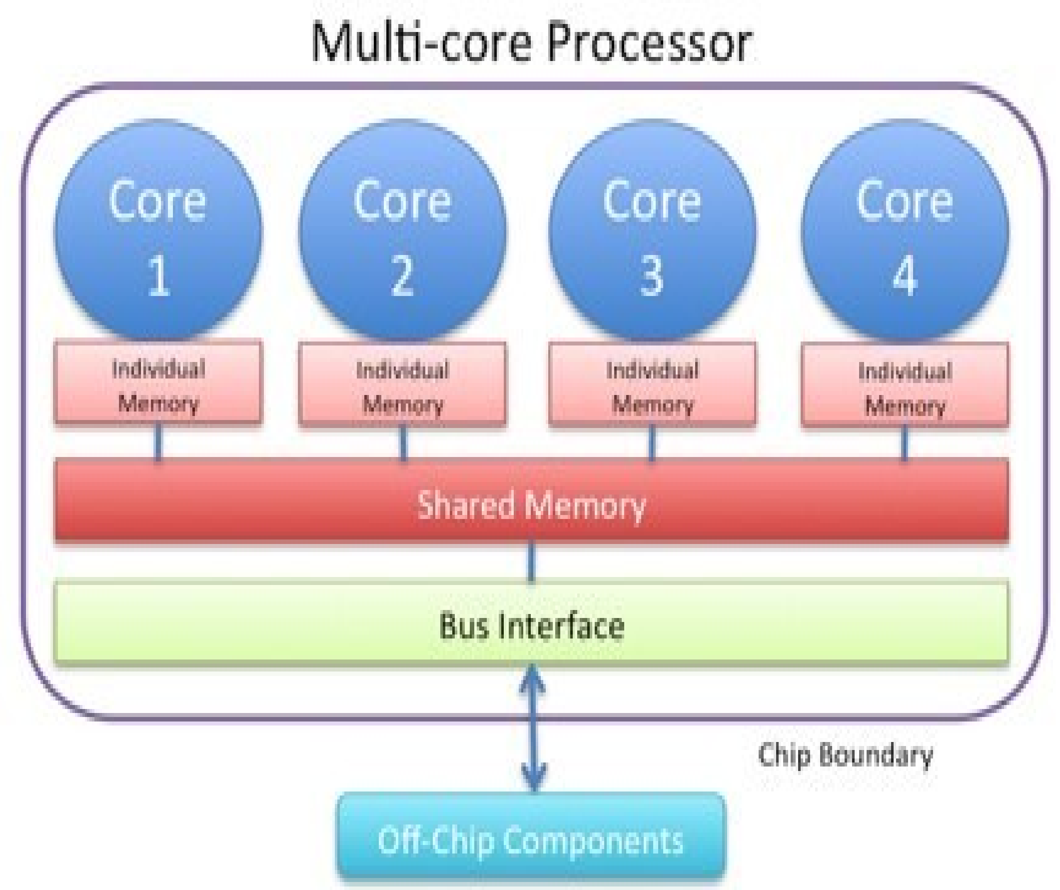
More Complex Execution Logic

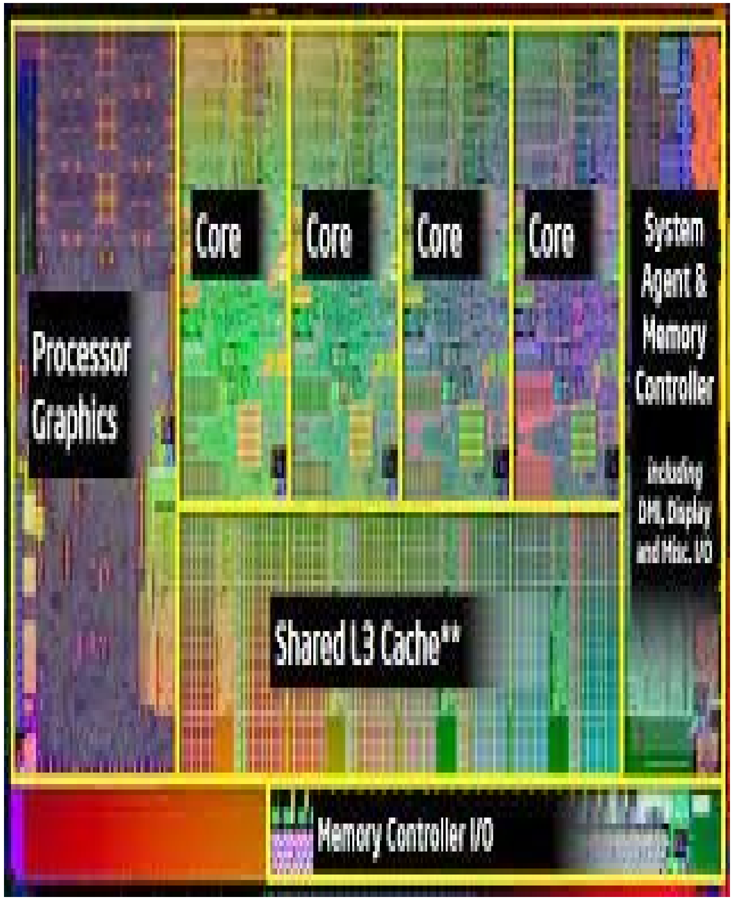
- Enable parallel execution of instructions
- Pipeline works like assembly line
  - Different stages of execution of different instructions at same time along pipeline
- Superscalar allows multiple pipelines within single processor
  - Instructions that do not depend on one another can be executed in parallel

New Approach - Multi-core chips

- Multiple processors on single chip
  - Large shared cache
- Within a processor, increase in performance proportional to square root of increase in complexity
- If software can use multiple processors, doubling number of processors almost doubles performance
- So, use two simpler processors on the chip rather than one more complex processor
- With two processors, larger caches are justified
  - Power consumption of memory logic less than processing logic

Multi-core processor





X86 evolution

- **8080 : first general purpose microprocessor**
- 8 bit data path
- Used in first personal computer – Altair
- **8086 – 5MHz – 29,000 transistors**
- much more powerful
- 16 bit
- instruction cache, prefetch few instructions
- **8088 (8 bit external bus) used in first IBM PC 80286**
- 16 Mbyte memory addressable
- up from 1Mb
- **80386: 32 bit**
- Support for multitasking
- **80486**
- sophisticated powerful cache and instruction pipelining
- built in maths co-processor

X86 evolution

- **Pentium**
  - Superscalar
  - Multiple instructions executed in parallel
- **Pentium Pro**
  - Increased superscalar organization
  - Aggressive register renaming
  - branch prediction
  - data flow analysis
  - speculative execution
- **Pentium II**
  - MMX technology
  - graphics, video & audio processing

Pentium III

- Additional floating point instructions for 3D graphics

X86 evolution

- **Pentium 4**
  - Arabic rather than Roman numerals
  - Further floating point and multimedia enhancements
- **Core**
  - First x86 with dual core
- **Core 2**
  - 64 bit architecture
- **Core 2 Quad – 3GHz – 820 million transistors**
  - Four processors on chip

x86 architecture dominant outside embedded systems
Organization and technology changed dramatically
Instruction set architecture evolved with backwards compatibility
~1 instruction per month added
500 instructions available
See Intel web pages for detailed information on processors