Foundations of Computer Science
Computer System

ONE LOVE. ONE FUTURE.

HUST

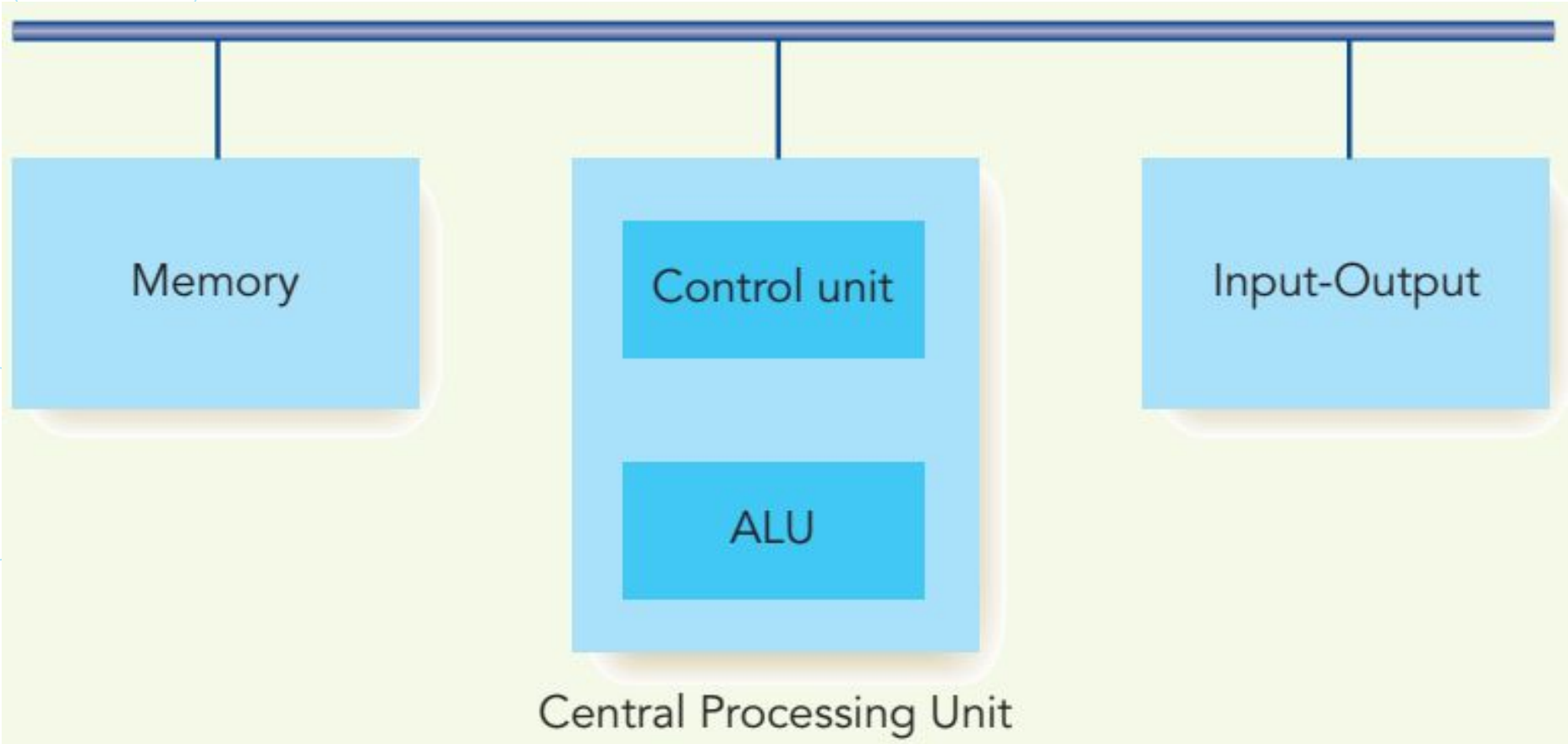
hust.edu.vn fb.com/dhbkhn

FaMI 1956

Contents

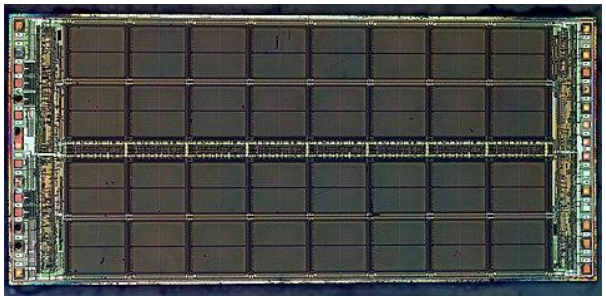
- Von Neumann Architecture
- Primary Memory
- Secondary Memory
- ALU
- Machine Language Instruction
- Von Neumann Computer
- Parallel Architectures

Component of Von Neumann Architecture



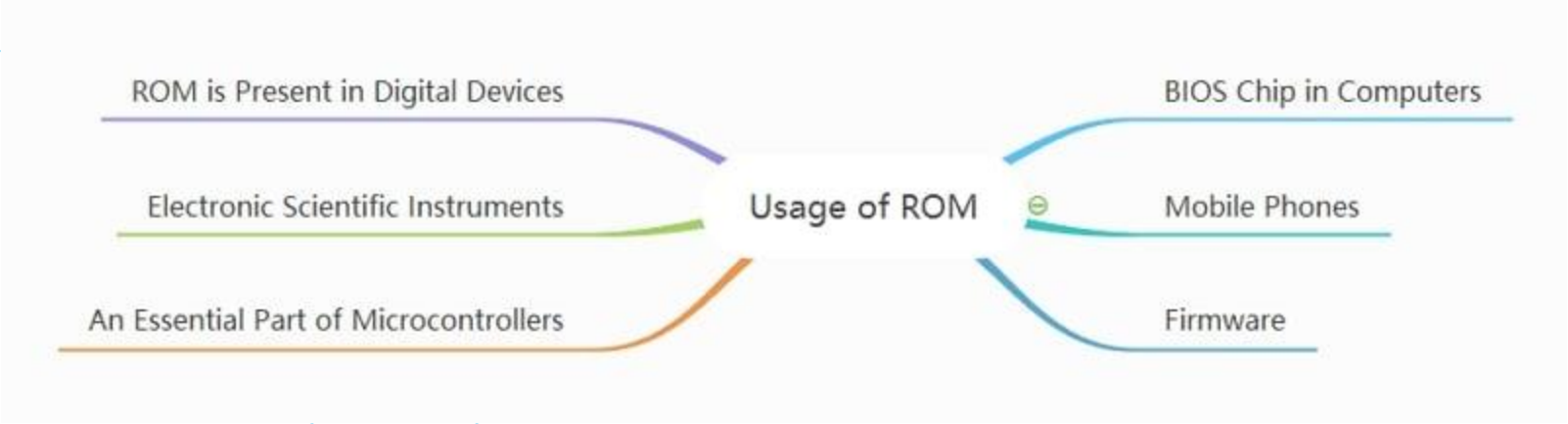
Memory

- **RAM (Random Access memory)**:
  - Divided into fixed-size units, **cells**, associated with a unique **address**
  - Fetch or store a complete cell
  - The time it takes to fetch or store the contents of a single cell is the same for all cells in memory



- **ROM (Read-only memory)**
  - Random access
  - Prerecorded during manufacture
  - Cannot be modified or removed, only fetched
  - User cannot accidentally or intentionally overwrite
  - Hold important system instructions and data
- Cell size: 8 (byte), 16, 32, or 64

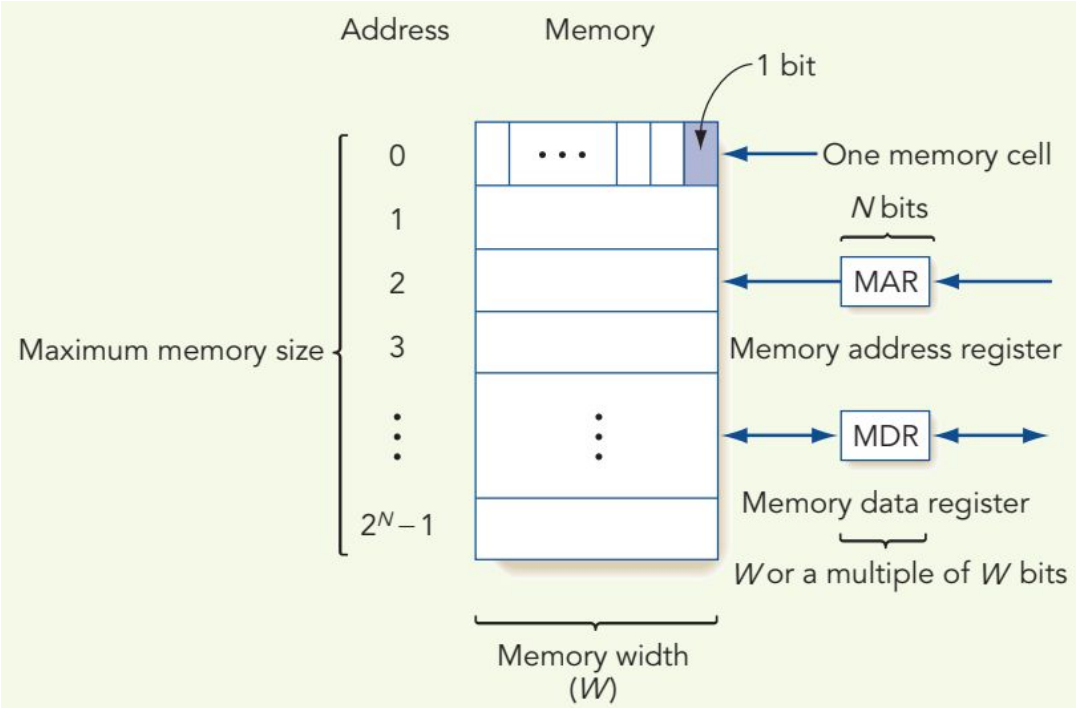
What is ROM used for?



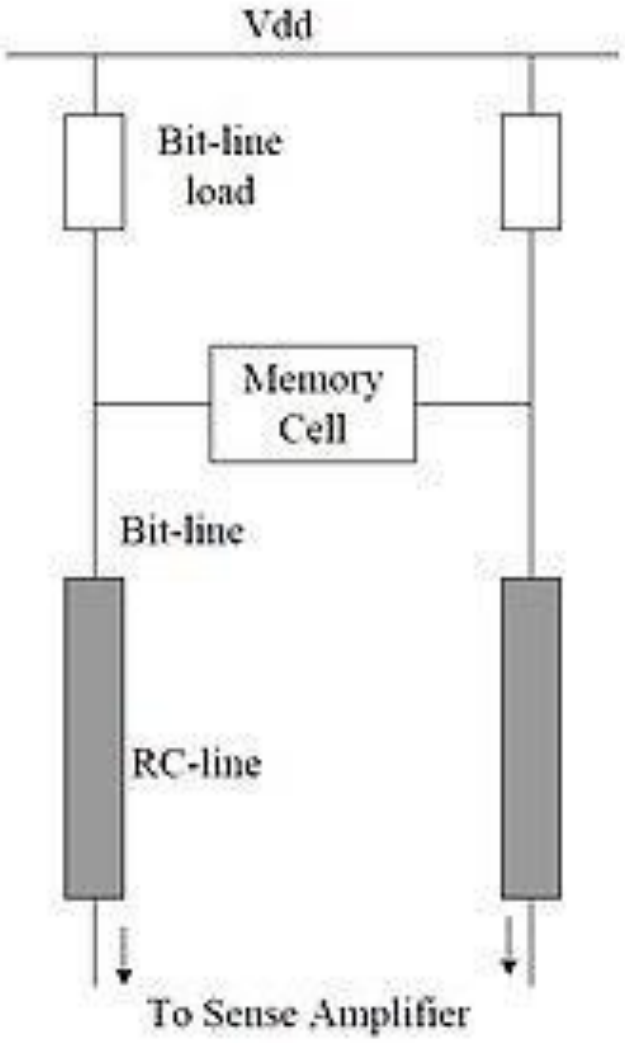
- MROM (Masked ROM): ROM is manufactured using mask supplied by the programmer.
- PROM (Programmable ROM): can be programmed
- EPROM (Erasable PROM): ROM can be reprogrammed using special devices, e.g. UV light for erasing.
- EEPROM (Electrical EPROM)
- Flash ROM

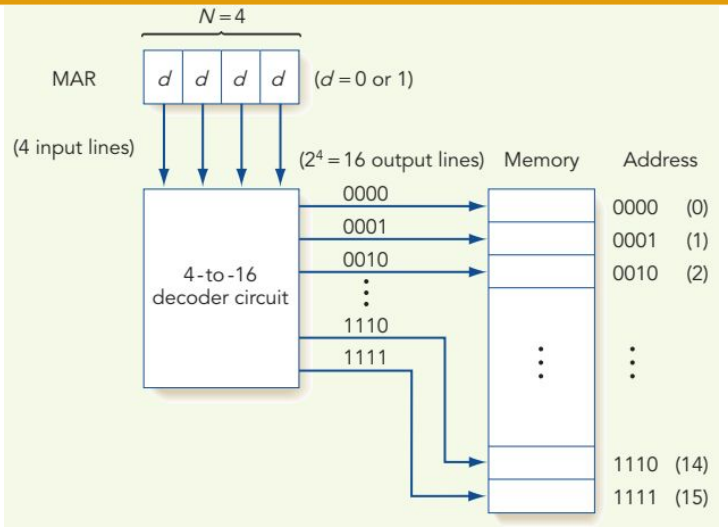
Structure of Memory

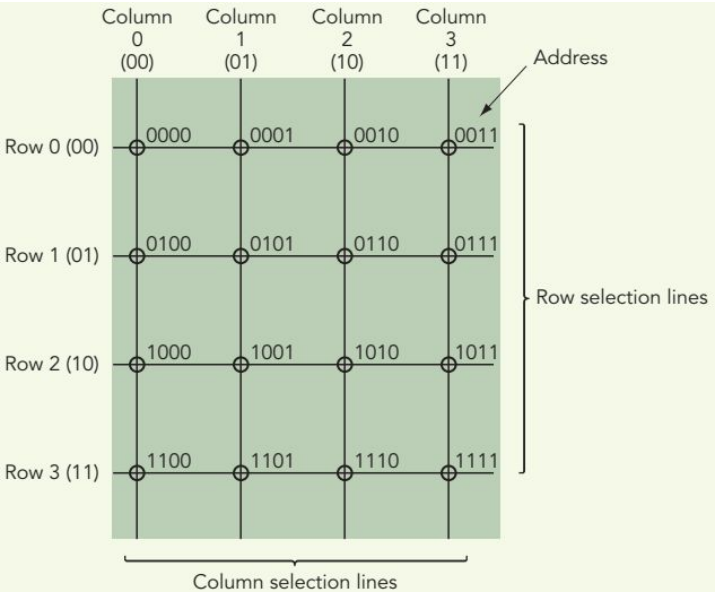
- Fetch (address)
  - Load the address → MAR.
  - Decode the address
  - Copy the contents → MDR
- Store(address, value)
  - Load the address → MAR
  - Load the value → MDR
  - Decode the address
  - Store the contents of the MDR → memory location



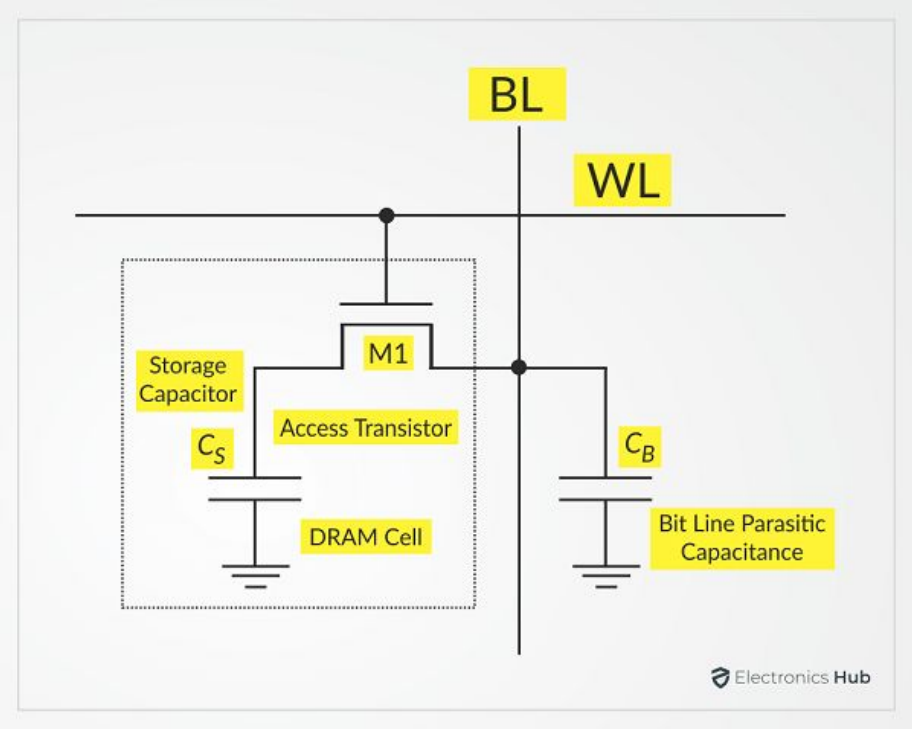
Memory Organization

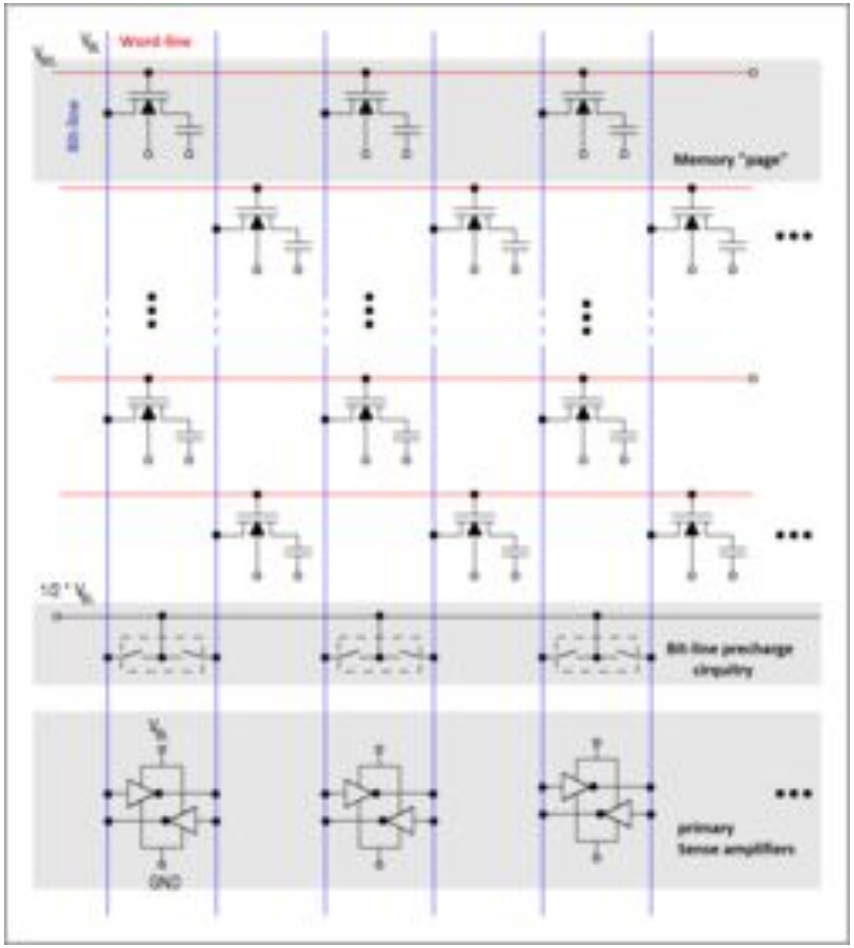




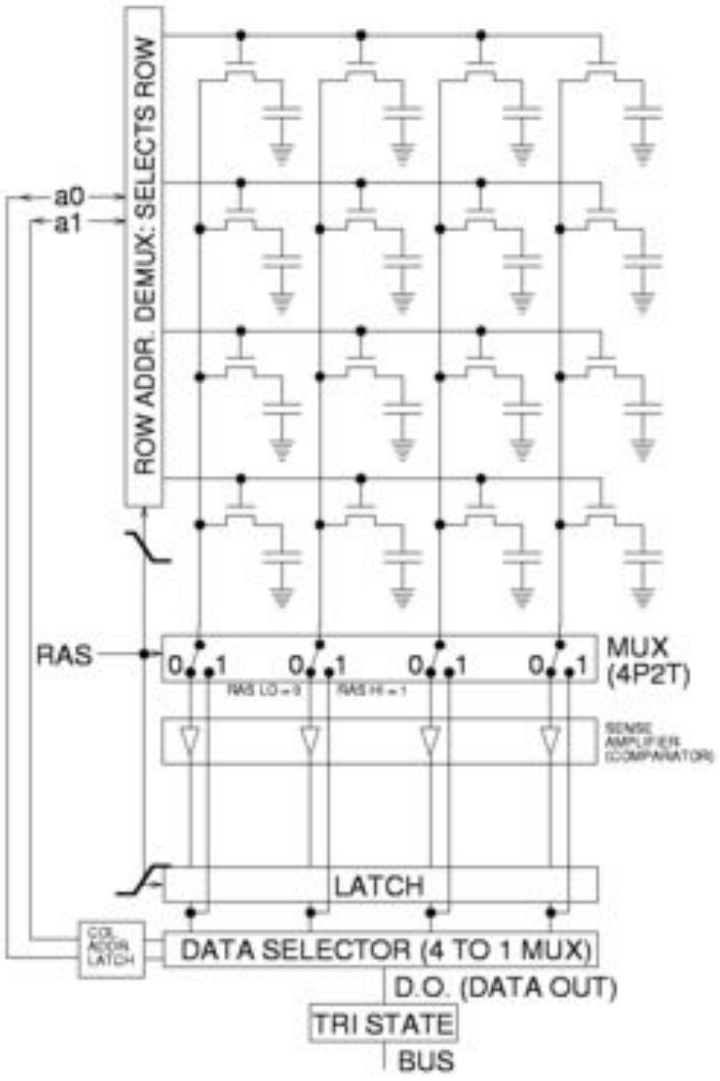


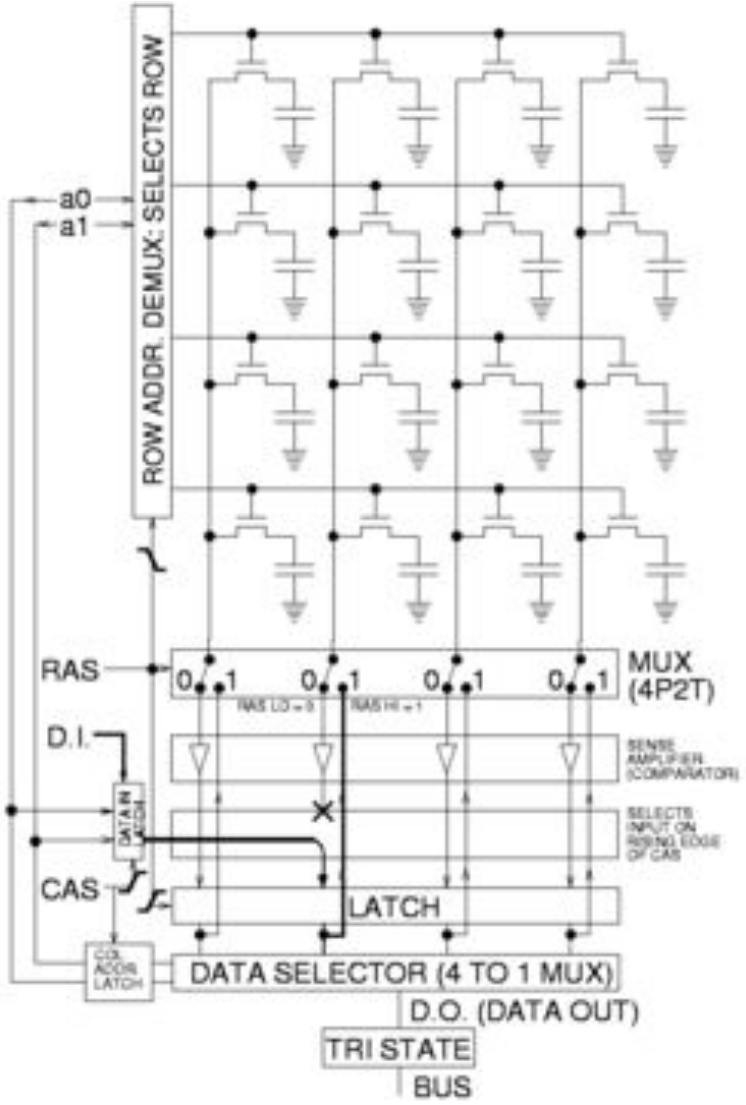
DRAM


BL
WL
M1
Storage Capacitor
Access Transistor
C_S
C_B
DRAM Cell
Bit Line Parasitic Capacitance
Electronics Hub

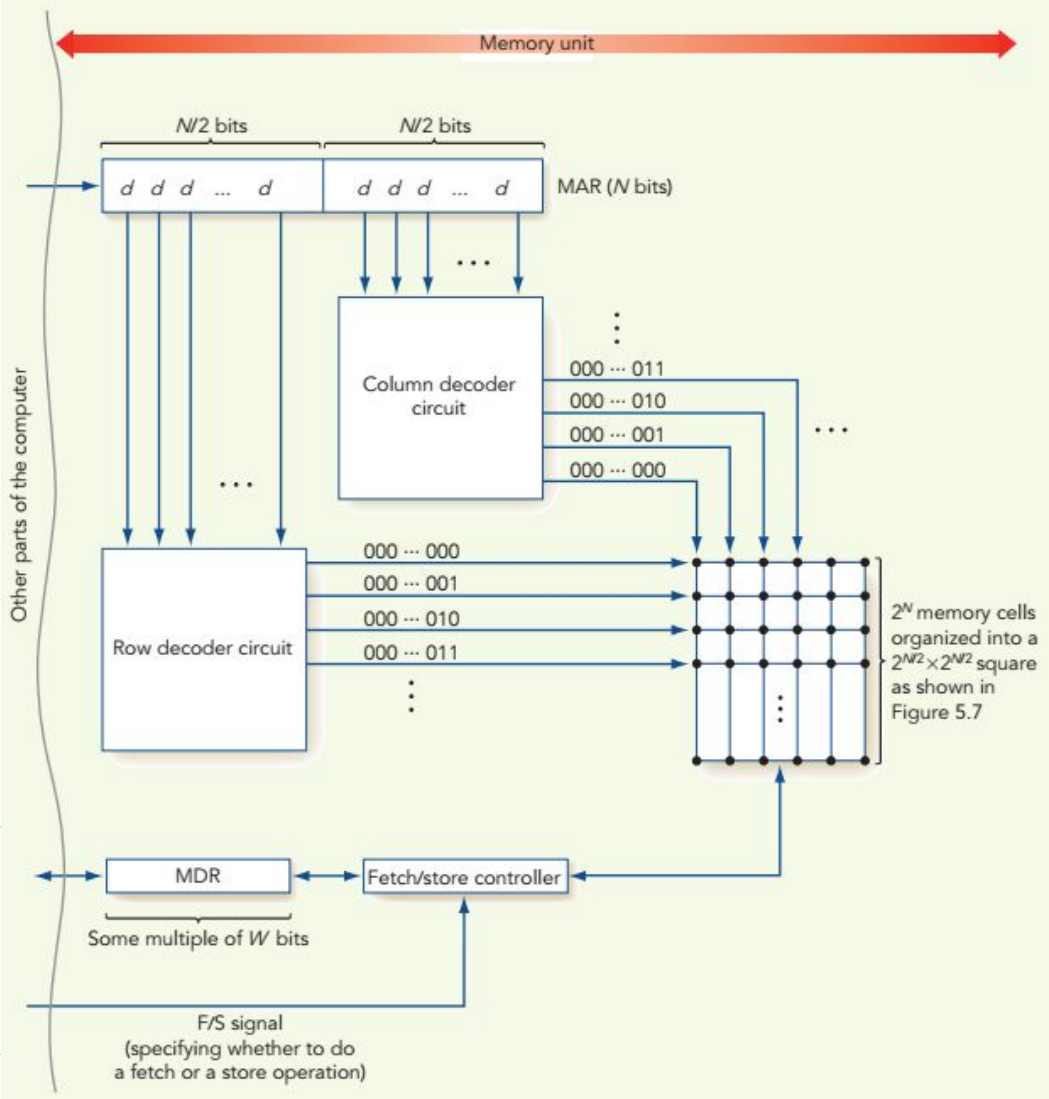

Word-line
Memory "page"
primary
sense amplifiers
GND

Read and Write process





Memory Organization



Memory Size

- Address Space: Maximum memory size

N	Maximum Memory Size (2^N)
16	65,536
20	1,048,576
22	4,194,304
24	16,777,216
32	4,294,967,296
40	1,099,511,627,776
50	1,125,899,906,842,624

2^{10} = 1K (= 1,024) — 1 KB = 1 *kilobyte* = one thousand bytes

2^{20} = 1M (= 1,048,576) — 1 MB = 1 *megabyte* = one million bytes

2^{30} = 1G (= 1,073,741,824) — 1 GB = 1 *gigabyte* = one billion bytes

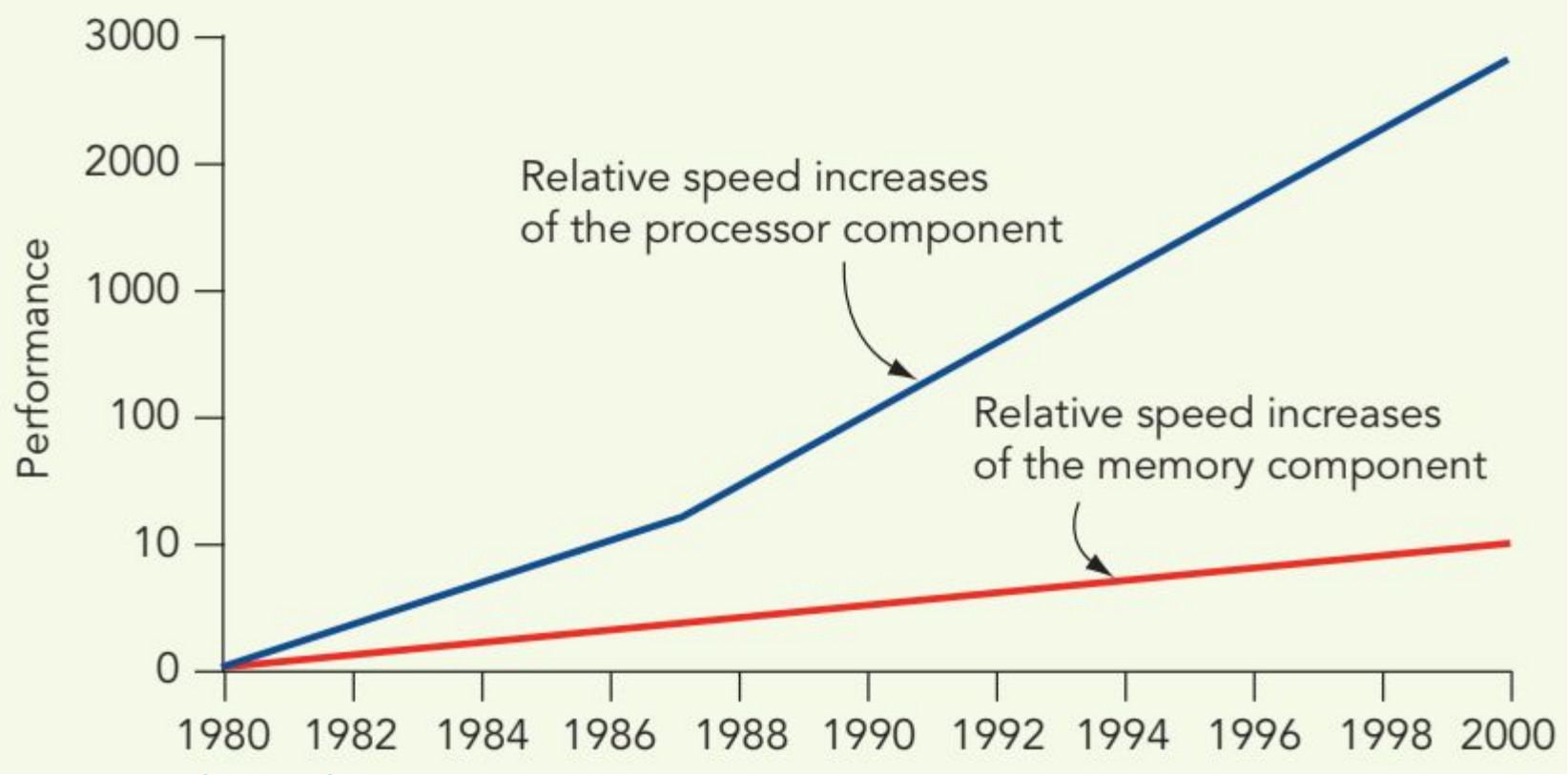
2^{40} = 1T (= 1,099,511,627,776) — 1 TB = 1 *terabyte* = one trillion bytes

2^{50} = 1P (= 1,125,899,906,842,624) — 1 PB = 1 *petabyte* = one quadrillion bytes

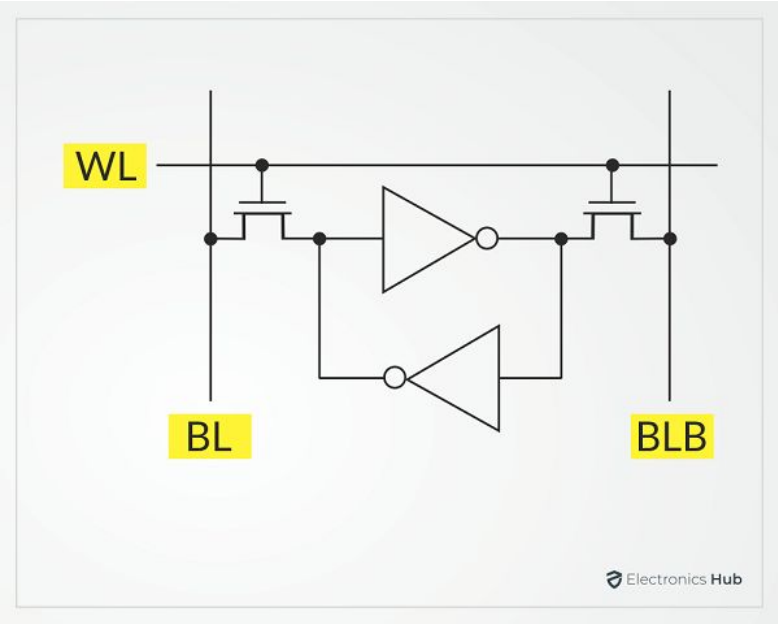
How Big?

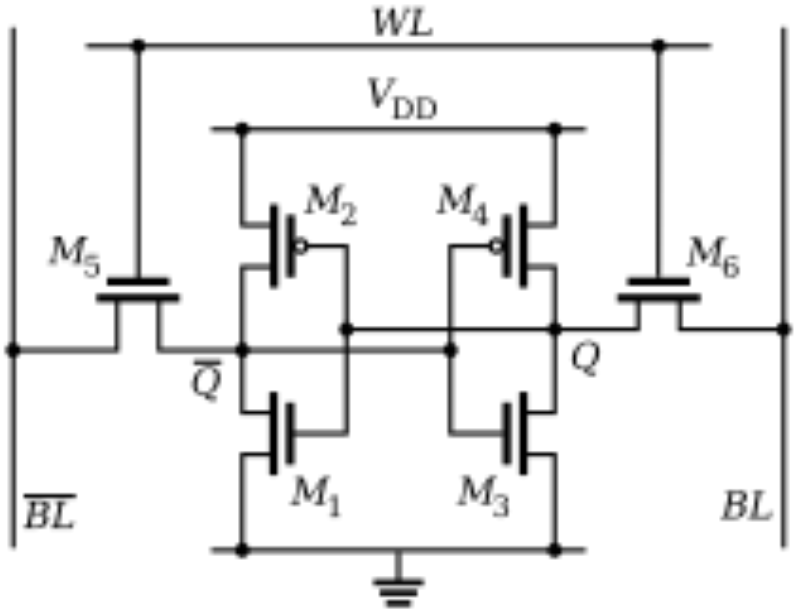
Quantity in Bytes	Base-10 Value	Amount of Textual Information
1 byte	10^0	One character
1 kilobyte	10^3	One typed page
1 megabyte	10^6	Two or three novels
1 gigabyte	10^9	A departmental library or a large personal library
1 terabyte	10^{12}	The library of a major academic research university
1 petabyte	10^{15}	All printed material in all libraries in North America
1 exabyte	10^{18}	All words ever printed throughout human history
1 zettabyte	10^{21}	As of 2014, the total amount of information stored globally on the World Wide Web was about 4–5 zettabytes
1 yottabyte	10^{24}	According to an article in Gizmodo,[1] a design and technology blog, storing 1 yottabyte of data on hard drives would require 1 million data centers that would fill the states of Rhode Island and Delaware

Speed


Performance
3000
2000
1000
100
10
0
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
Relative speed increases
of the processor component
Relative speed increases
of the memory component

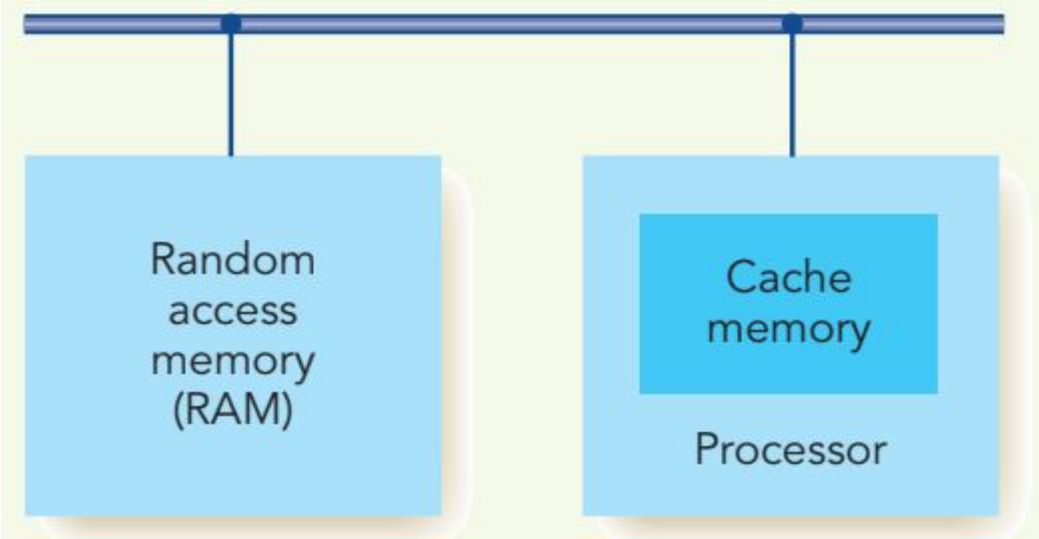
SRAM





STATIC RAM	DYNAMIC RAM
Does not require refreshing	Must be continuously refreshed
Requires multiple transistors to store one bit	Requires one transistor and one capacitor to store one bit
Delivers faster access times	Delivers slower performance
Consumes less power, especially in idle	Consumes more power
Takes up more space	Requires less space
Can hold only a small amount of data	Can hold much more data
Costs more than DRAM	Costs less than SRAM
Typically used for a processor's cache	Typically used for a computer's main memory

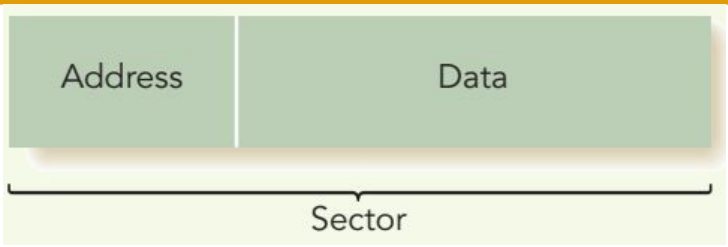
Two-level Hierarchy

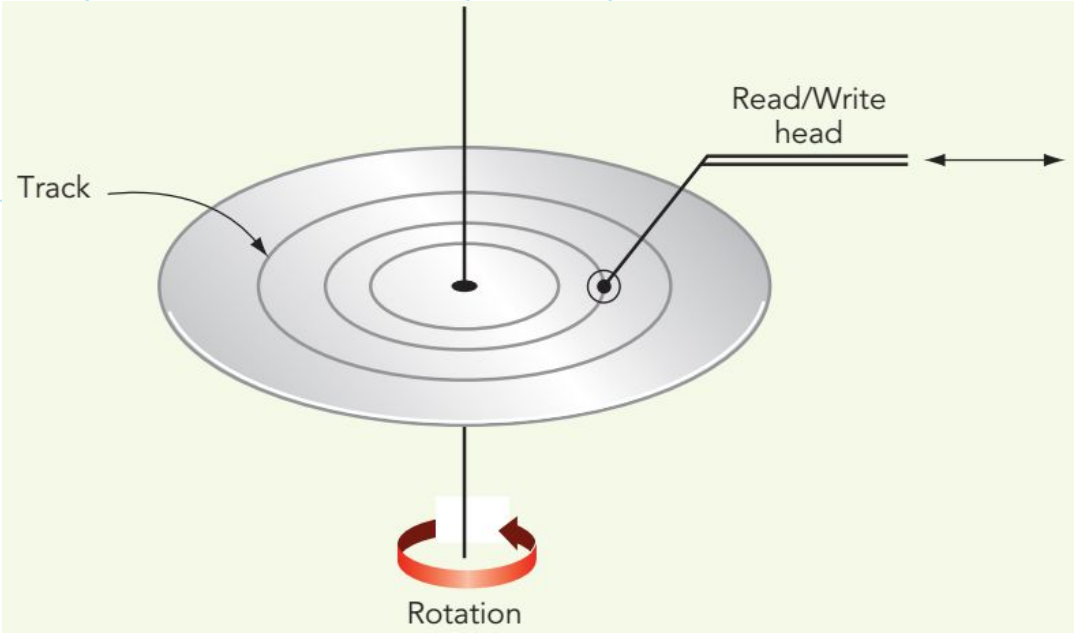


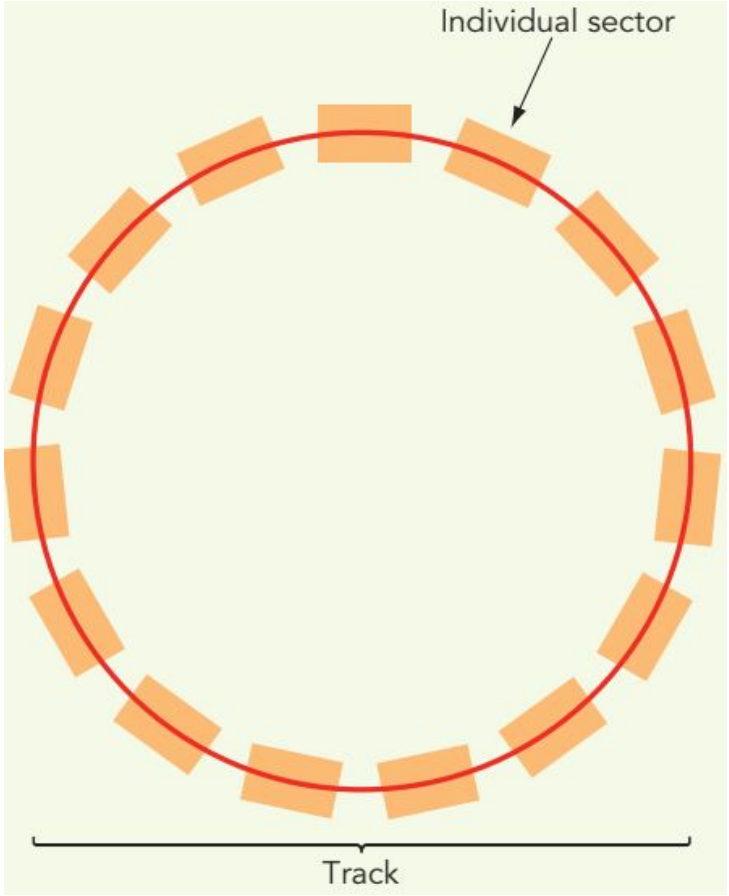
Cache Hit Rate

Hard Disk Organization

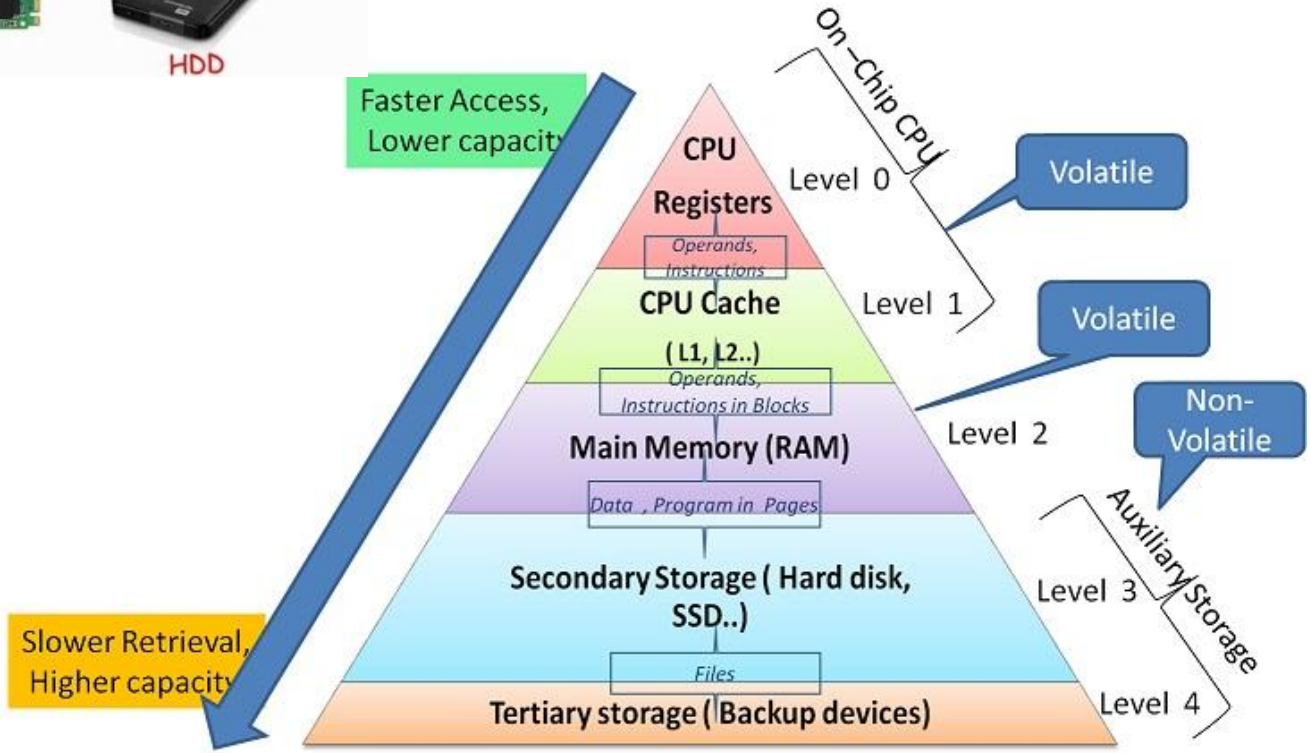
- **Seek time:** positioning the read/write head over the correct track
- **Latency:** beginning of the desired sector rotate under the head
- **Transfer time:** entire sector pass under the
head, the contents are read into or written



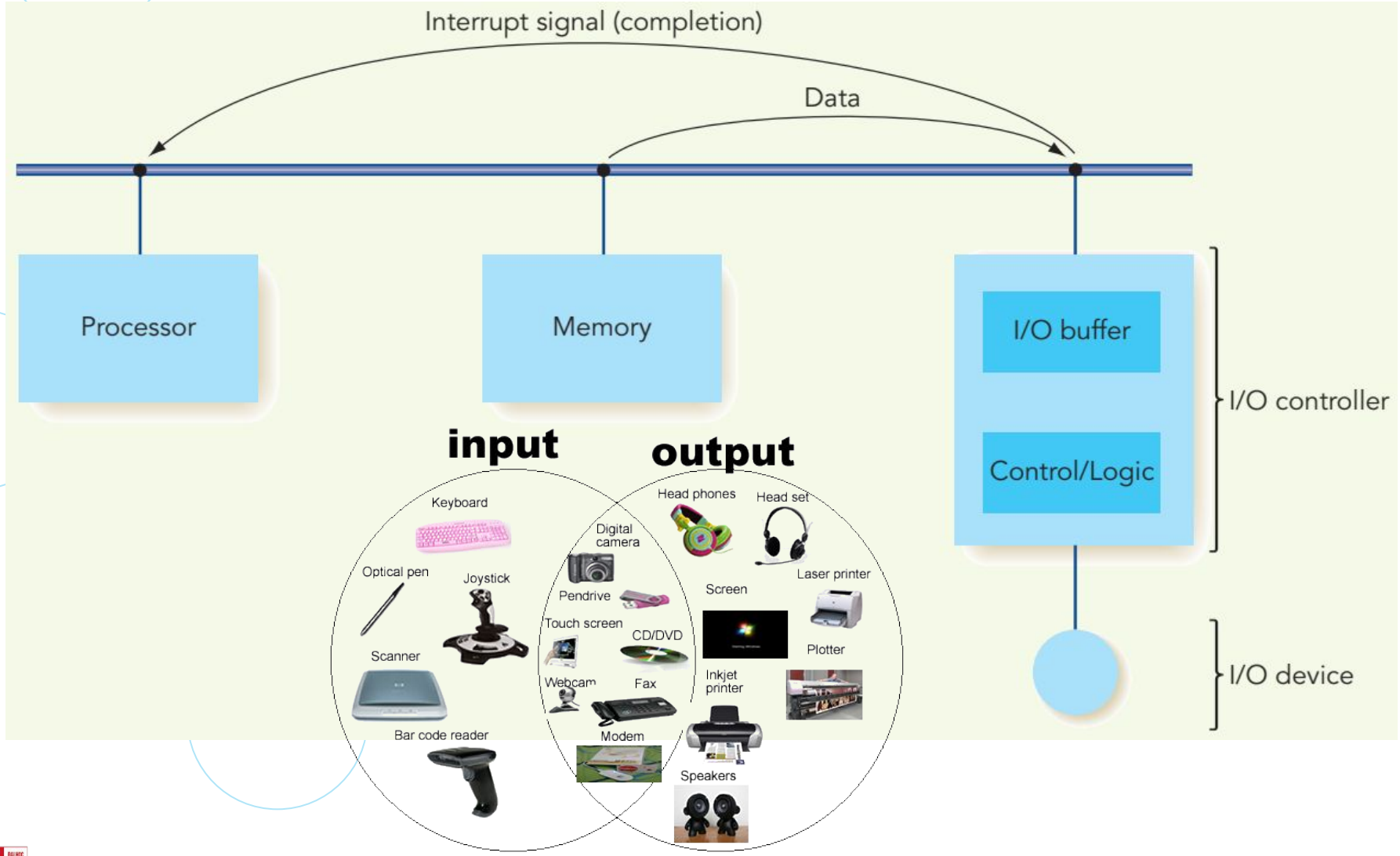




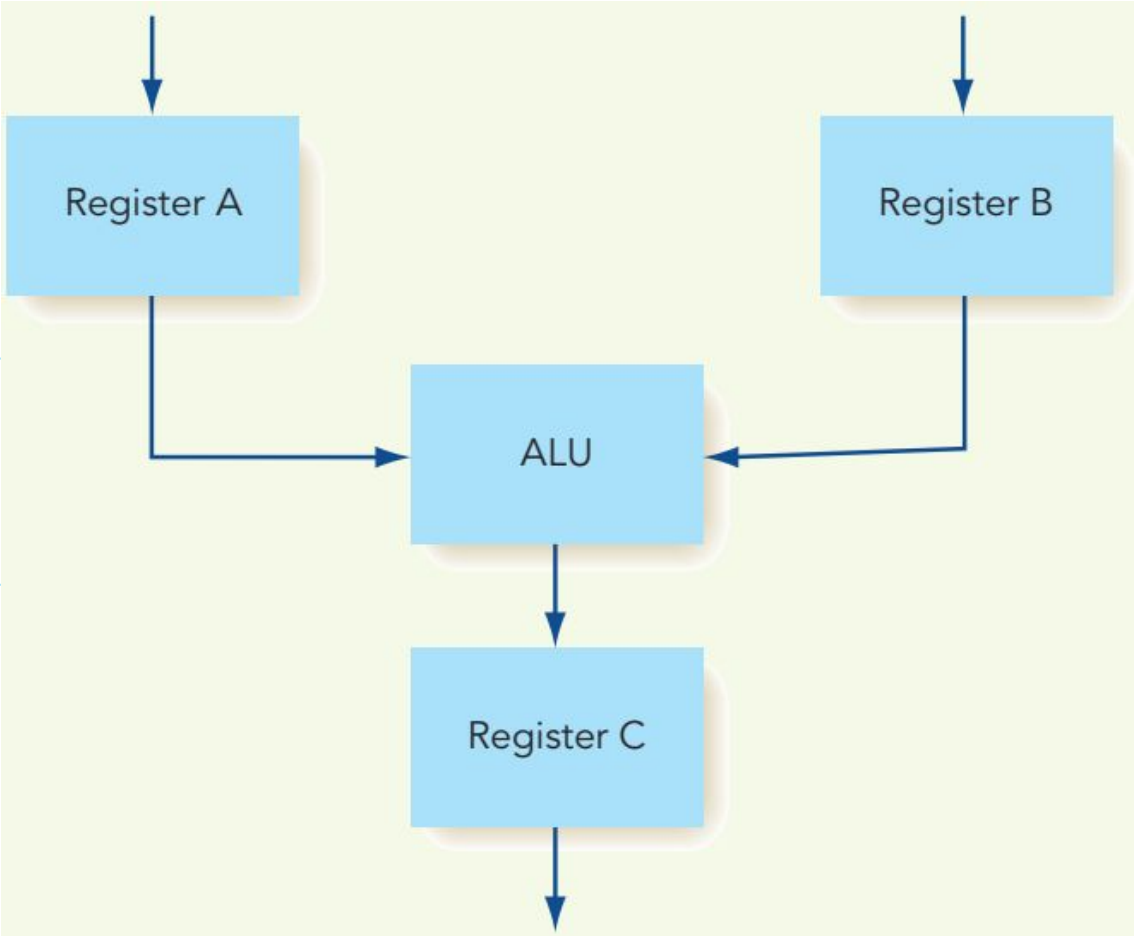
Secondary and Tertiary Storages

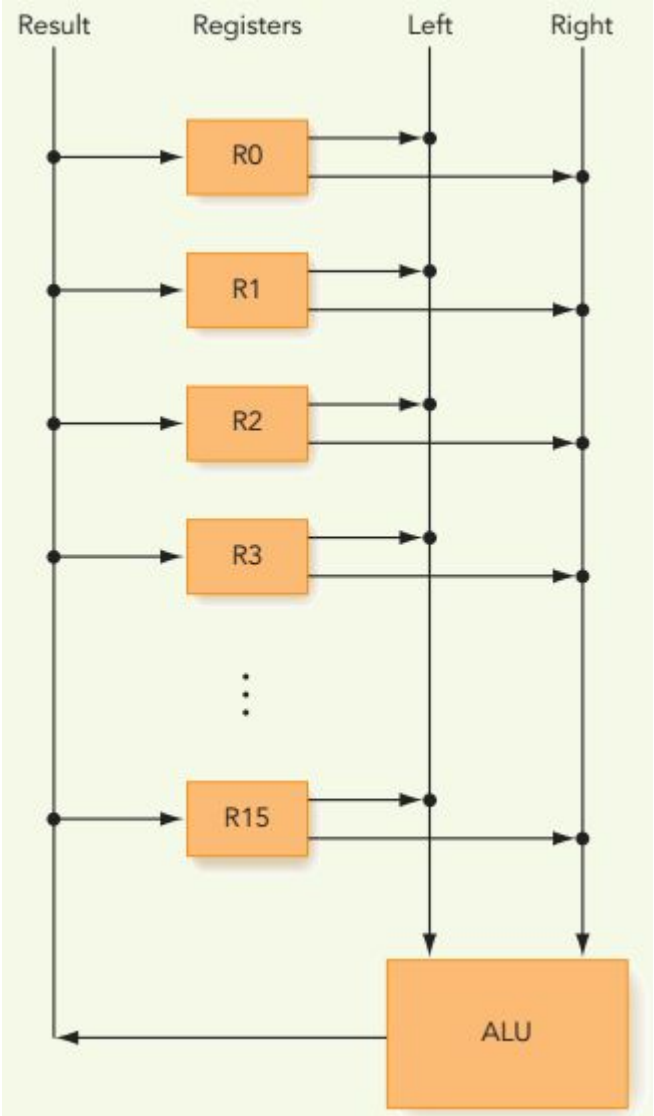


I/O Controller

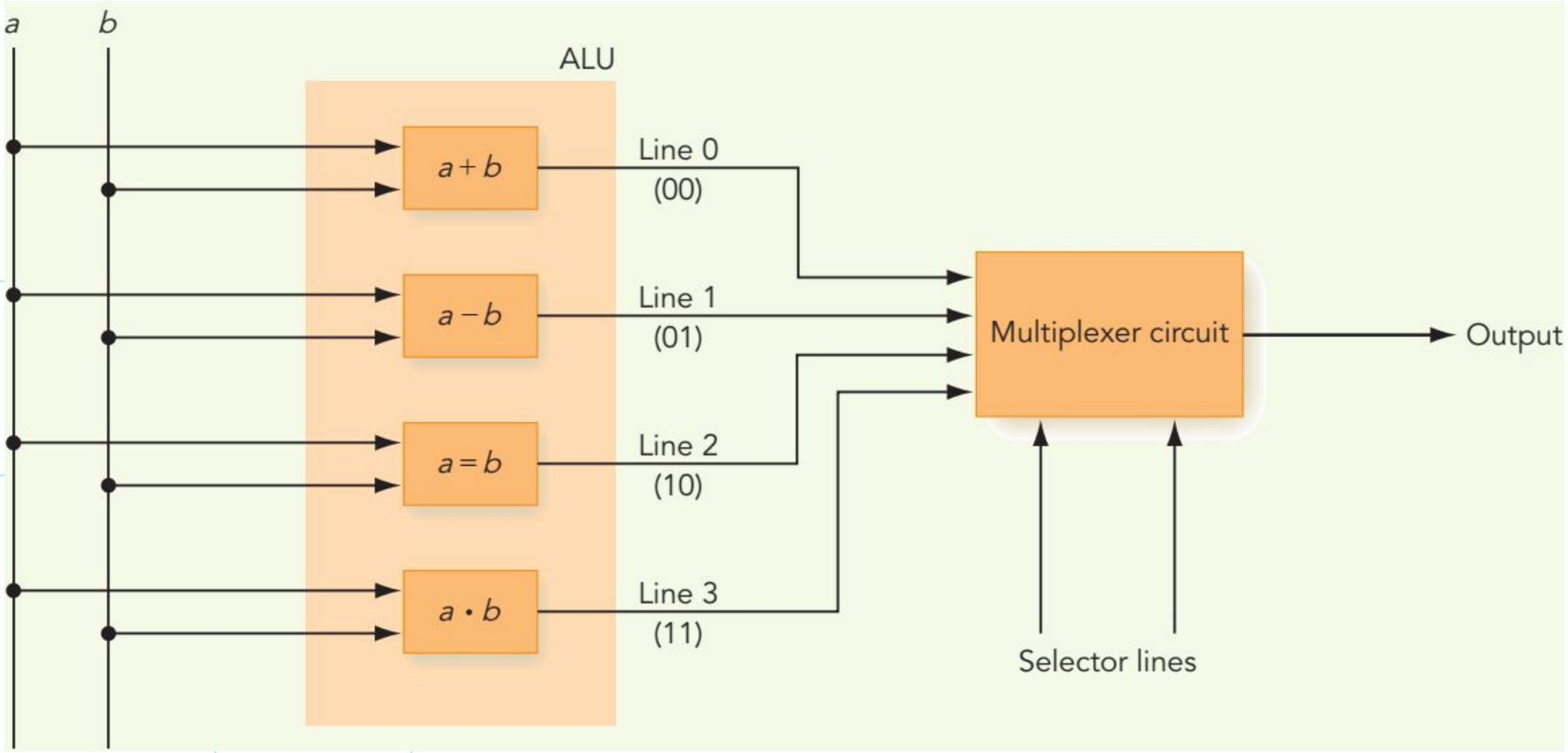


ALU Organization

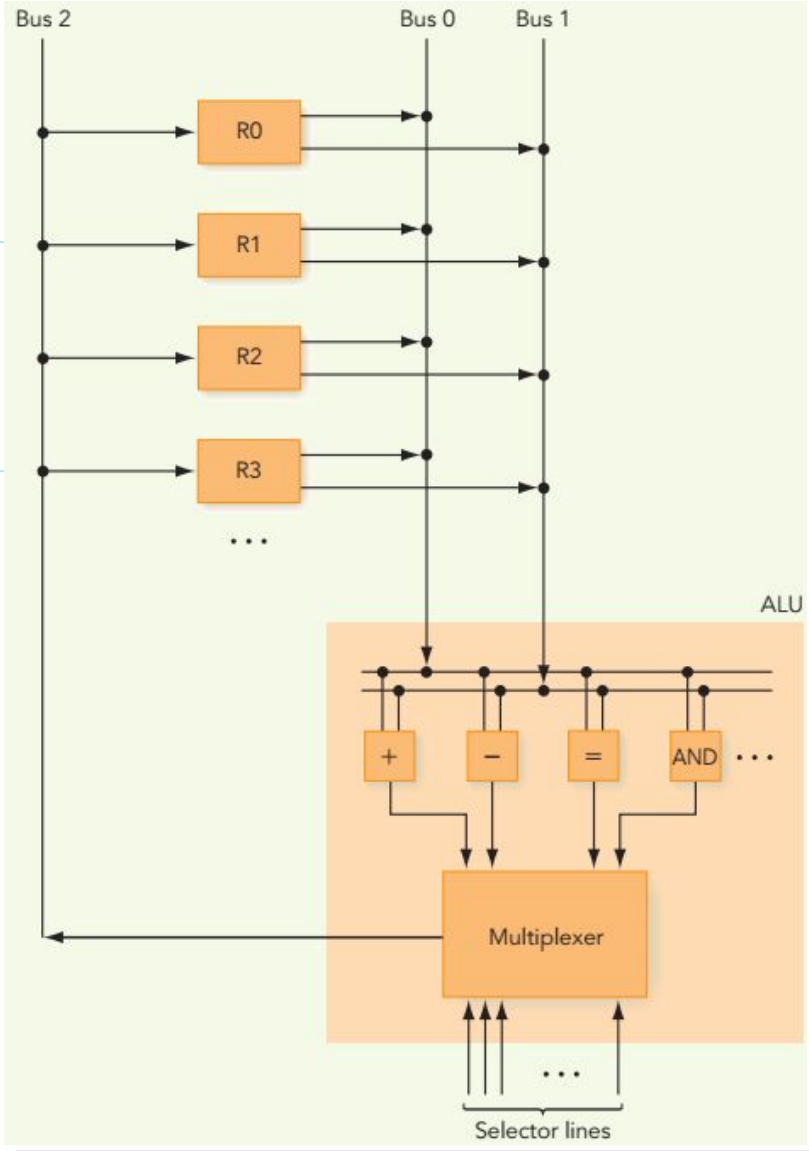




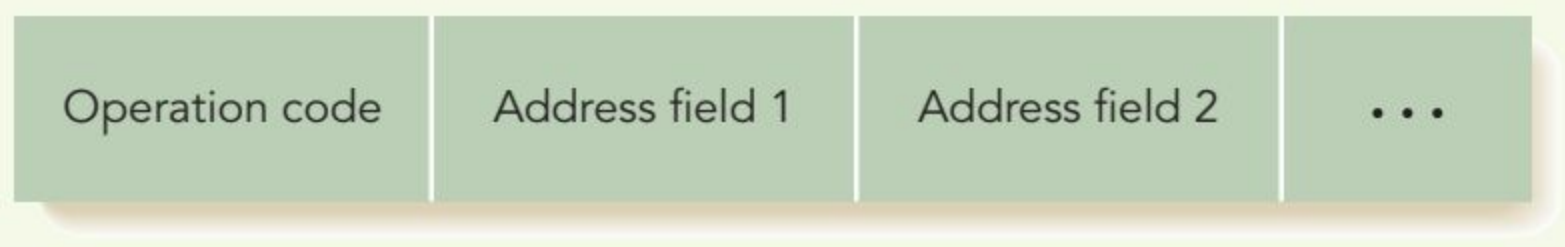
ALU with Multiplexer



Overall ALU Organization



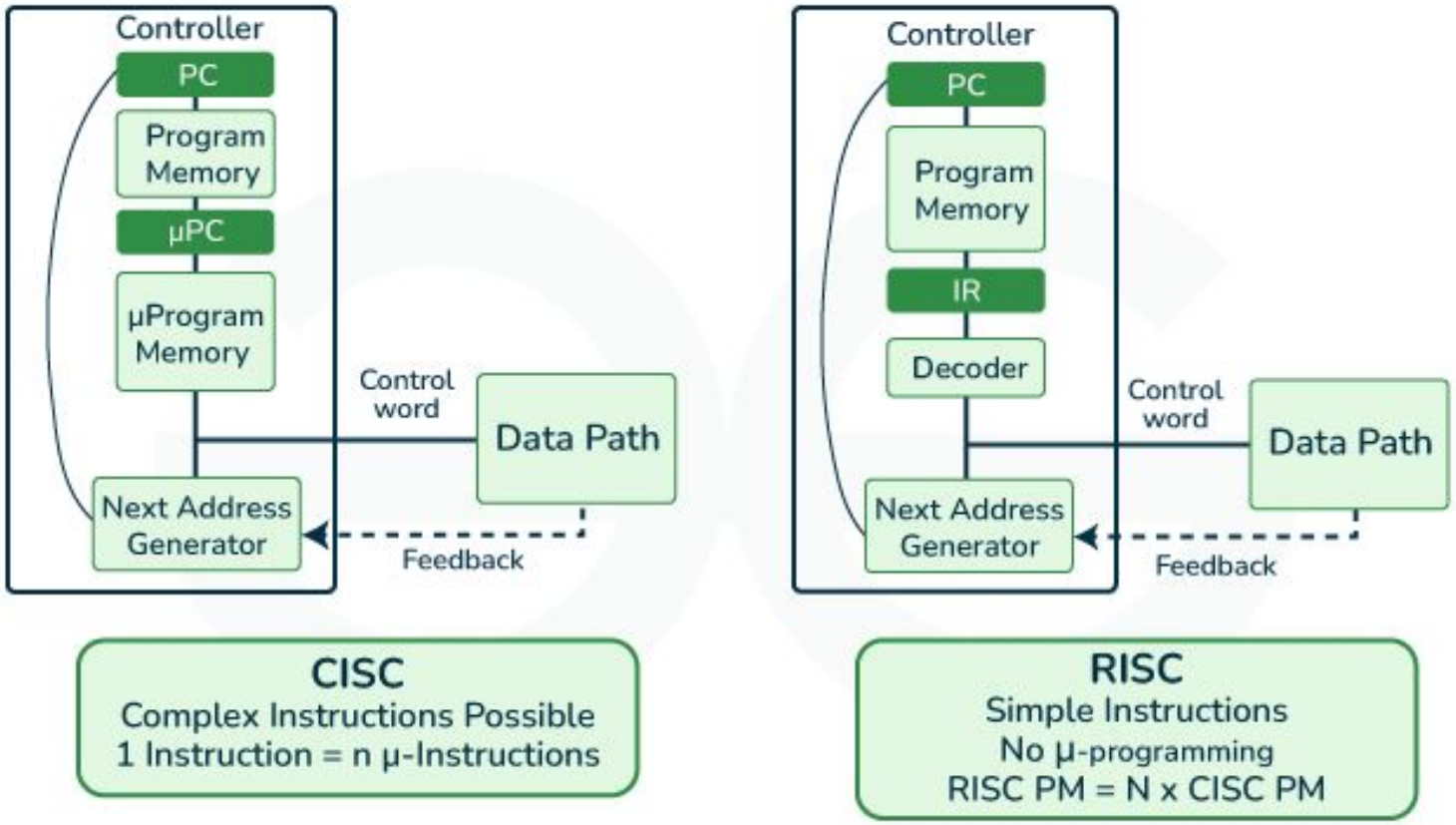
Machine Language Instruction



00001001 0000000001100011 0000000001100100

op code *address 1* *address 2*

Instruction Set



RISC and CISC

Complex Instruction Set Computer Reduced Instruction Set Computer

Instructions

- Data Transfer:
  - Memory cell → ALU register
  - ALU register → memory cell
  - One memory cell → another memory cell
  - One ALU register → another ALU register

LOAD X	Load register R with the contents of memory cell X.
STORE X	Store the contents of register R into memory cell X.
MOVE X,Y	Copy the contents of memory cell X into memory cell Y.

- Arithmetic

ADD X,Y, Z	Add the contents of memory cell X to the contents of memory cell Y and put the result into memory cell Z. This is called a *three-address instruction*, and it performs the operation CON(Z) = CON(X) + CON(Y).
ADD X,Y	Add the contents of memory cell X to the contents of memory cell Y. Put the result back into memory cell Y. This is called a *two-address instruction*, and it performs the operation CON(Y) = CON(X) + CON(Y).
ADD X	Add the contents of memory cell X to the contents of register R. Put the result back into register R. This is called a *one-address instruction*, and it performs the operation R = CON(X) + R. (Of course, R must be loaded with the proper value before executing the instruction.)

- Compare: COMPARE X, Y

CON (X) > CON (Y)	GT = 1 EQ = 0 LT = 0
CON (X) = CON (Y)	GT = 0 EQ = 1 LT = 0
CON (X) < CON (Y)	GT = 0 EQ = 0 LT = 1

- Branch

JUMP X	Take the next instruction unconditionally from memory cell X.
JUMPGT X	If the GT indicator is a 1, take the next instruction from memory cell X. Otherwise, take the next instruction from the next sequential location.

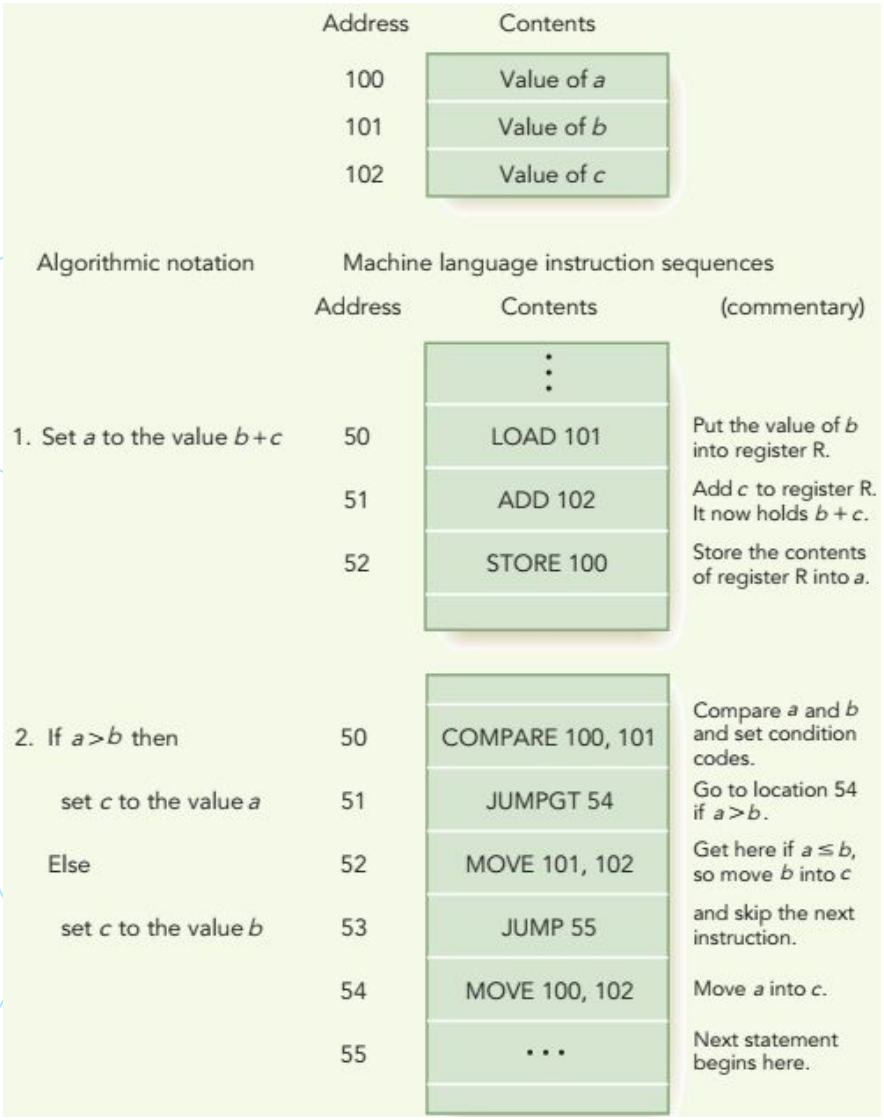
(JUMPEQ and JUMPLT would work similarly on the other two conditio codes.)

JUMPGE X	If *either* the GT or the EQ indicator is a 1, take the next instruction from memory location X. Otherwise, take the next instruction from the next sequential location.

(JUMPLE and JUMPNEQ would work in a similar fashion.)

HALT	Stop program execution. Don't go on to the next instruction.

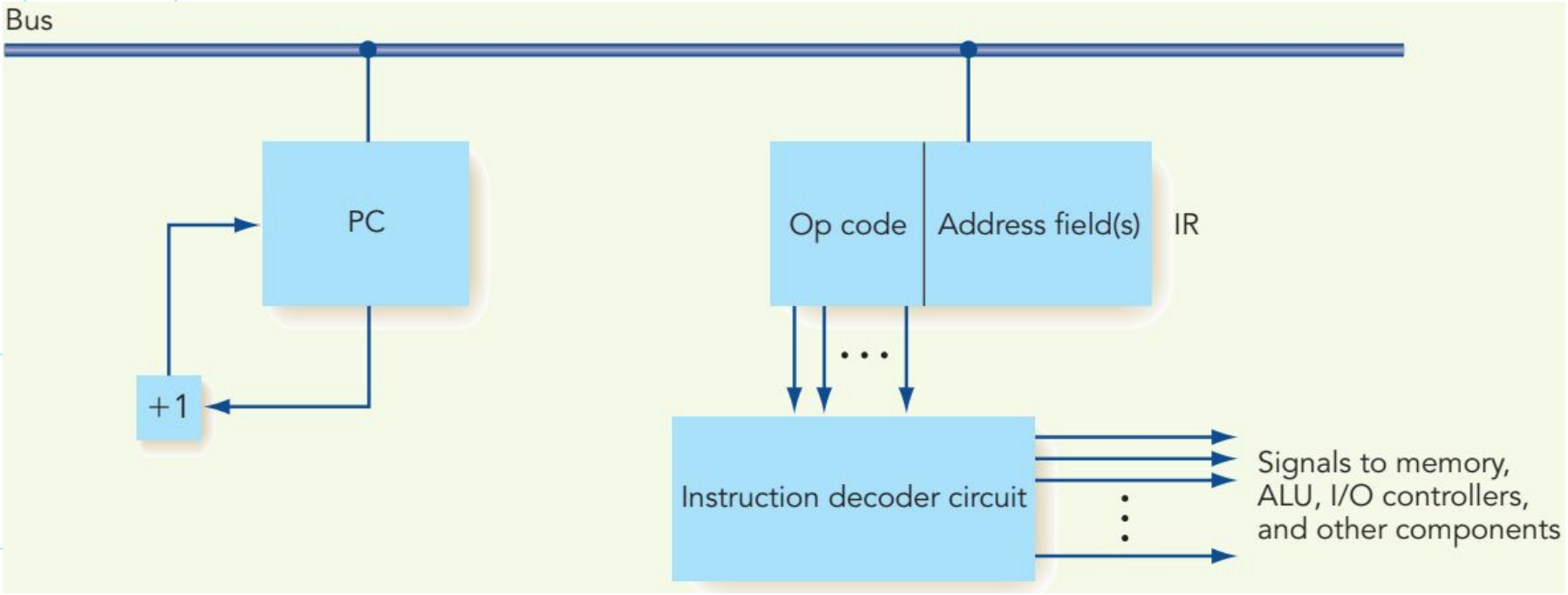
Example


Address
Contents
100
Value of a
101
Value of b
102
Value of c
Algorithmic notation
Machine language instruction sequences
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.
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 ≤ 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.

Machine Language

- First generation programming language
- Low-level programming language
- Uses binary, no natural language words, mathematical symbols, or other convenient mnemonics to make the language more readable
- Allows only numeric memory addresses (in binary). A programmer
  cannot name an instruction or a piece of data and refer to it by name.
- Difficult to change. If we insert or delete an instruction, all memory addresses following that instruction will change.
- Difficult to create data. The conversion algorithms are complicated and time consuming.

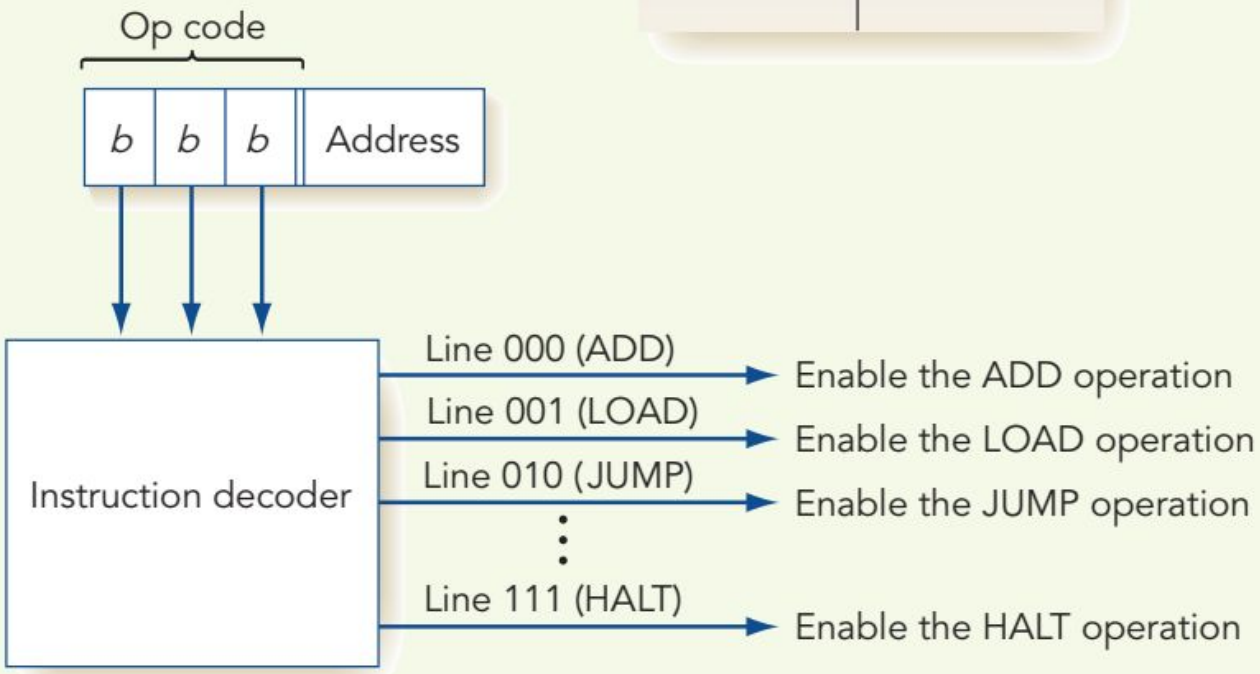
Control Registers and Circuits



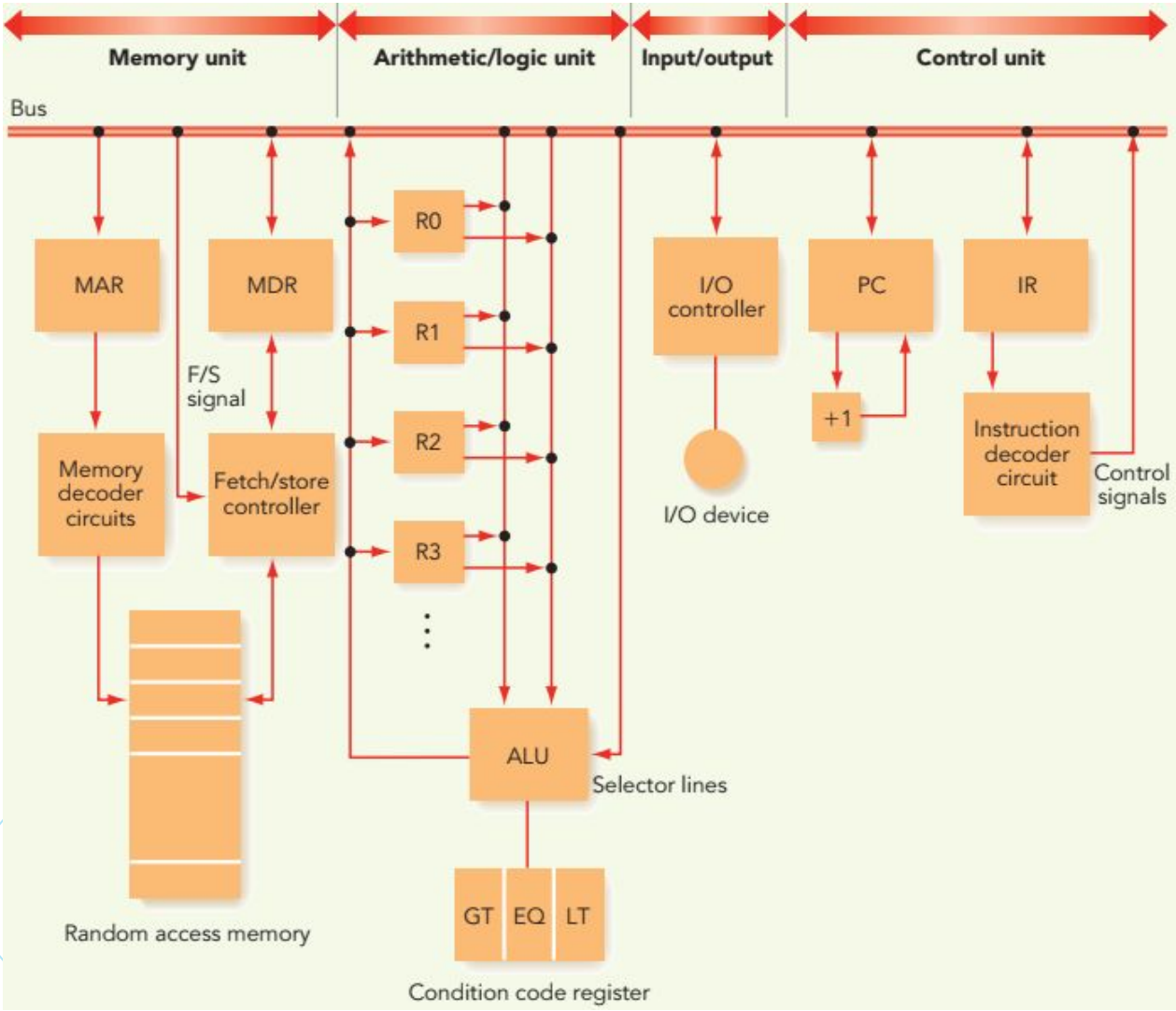
- PC – Program Counter: Hold the address of the next instruction
- IR – Instruction Register: Hold the instruction fetched from the memory

Instruction Decoder

Op code	Instruction
000	ADD
001	LOAD
010	JUMP
⋮	⋮
111	HALT



Von Neumann Computer



Instruction Set Example

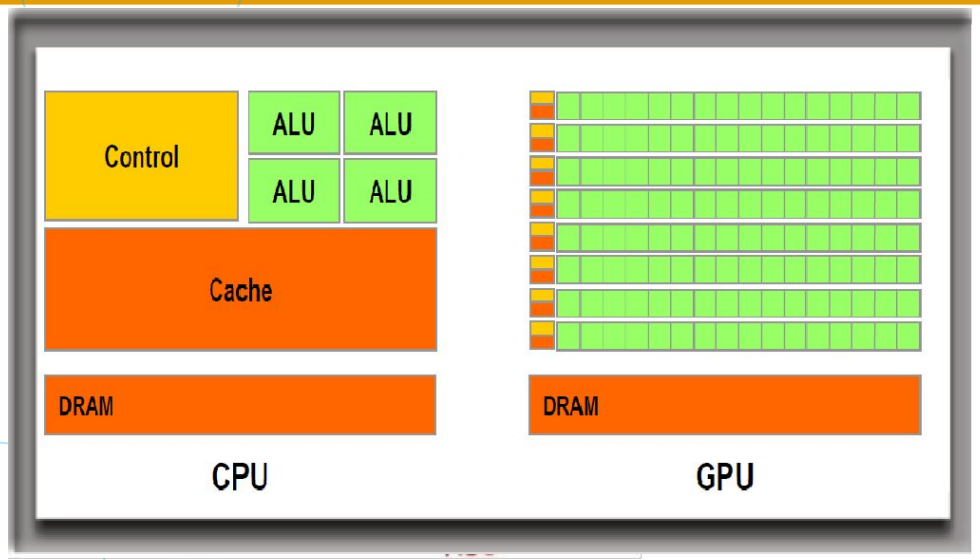
While we do not have a HALT instruction or a fatal error

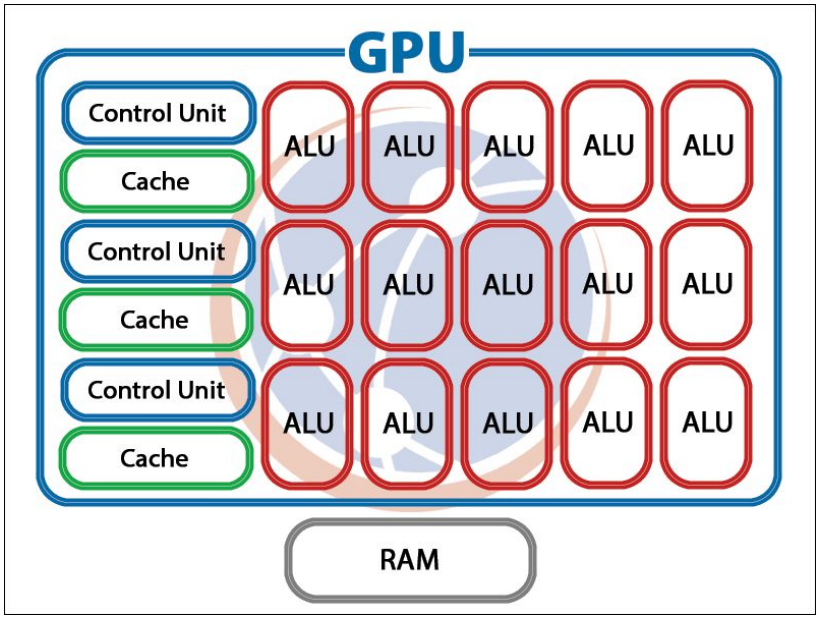
- Fetch phase
- Decode phase
- Execute phase

End of the loop

Binary Op Code	Operation	Meaning
0000	LOAD X	CON(X) → R
0001	STORE X	R → CON(X)
0010	CLEAR X	0 → CON(X)
0011	ADD X	R + CON(X) → R
0100	INCREMENT X	CON(X) + 1 → CON(X)
0101	SUBTRACT X	R – CON(X) → R
0110	DECREMENT X	CON(X) – 1 → CON(X)
0111	COMPARE X	if CON(X) > R then GT = 1 else 0 if CON(X) = R then EQ = 1 else 0 if CON(X) < R then LT = 1 else 0
1000	JUMP X	Get the next instruction from memory location X.
1001	JUMPGT X	Get the next instruction from memory location X if GT = 1.
1010	JUMPEQ X	Get the next instruction from memory location X if EQ = 1.
1011	JUMPLT X	Get the next instruction from memory location X if LT = 1.
1100	JUMPNEQ X	Get the next instruction from memory location X if EQ = 0.
1101	IN X	Input an integer value from the standard input device and store into memory cell X
1110	OUT X	Output, in decimal notation, the value stored in memory cell X.
1111	HALT	Stop program execution.

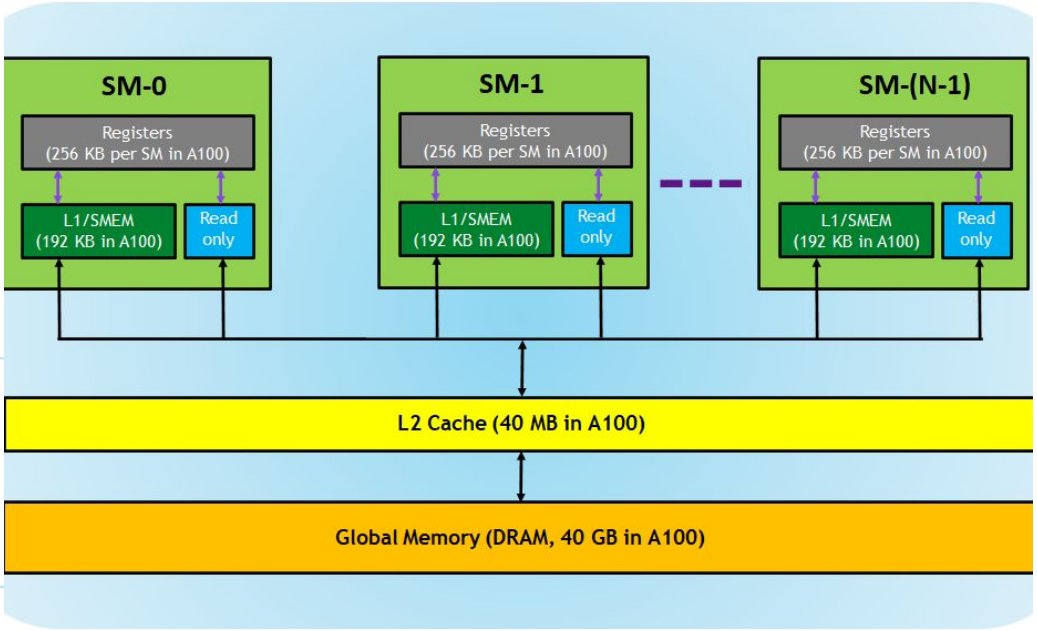
CPU vs. GPU (Graphical Processing Unit)

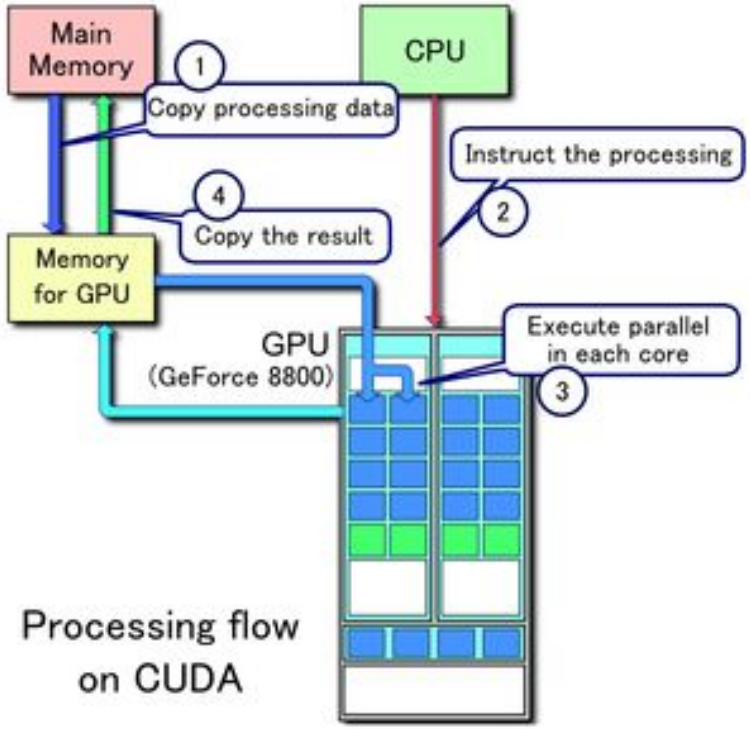




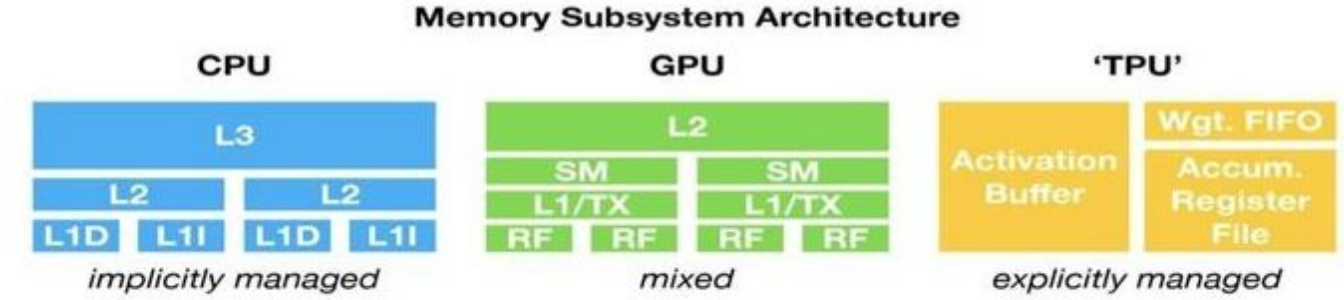
CPU	GPU
Central Processing Unit	Graphics Processing Unit
4-8 Cores	100s or 1000s of Cores
Low Latency	High Throughput
Good for Serial Processing	Good for Parallel Processing
Quickly Process Tasks That Require Interactivity	Breaks Jobs Into Separate Tasks To Process Simultaneously
Traditional Programming Are Written For CPU Sequential Execution	Requires Additional Software To Convert CPU Functions to GPU Functions for Parallel Execution

CUDA Architecture



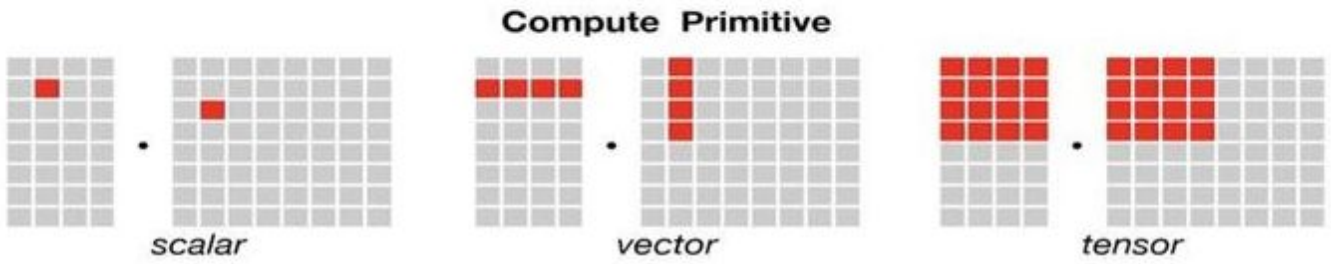


CPU vs. GPU vs. TPU (Tensor Processing Unit) - Google

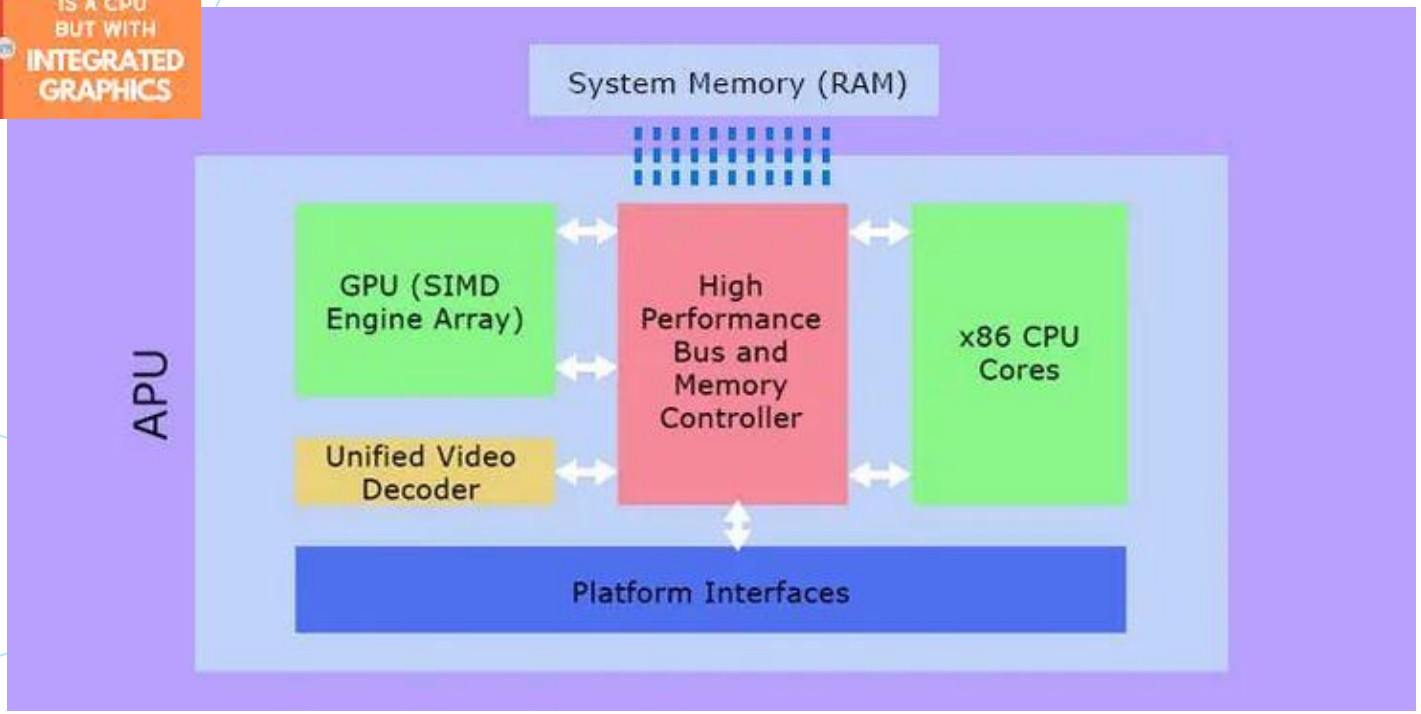


Compute Primitive

This image summarizes the compute primitive (smallest unit) in CPU, GPU and TPU:

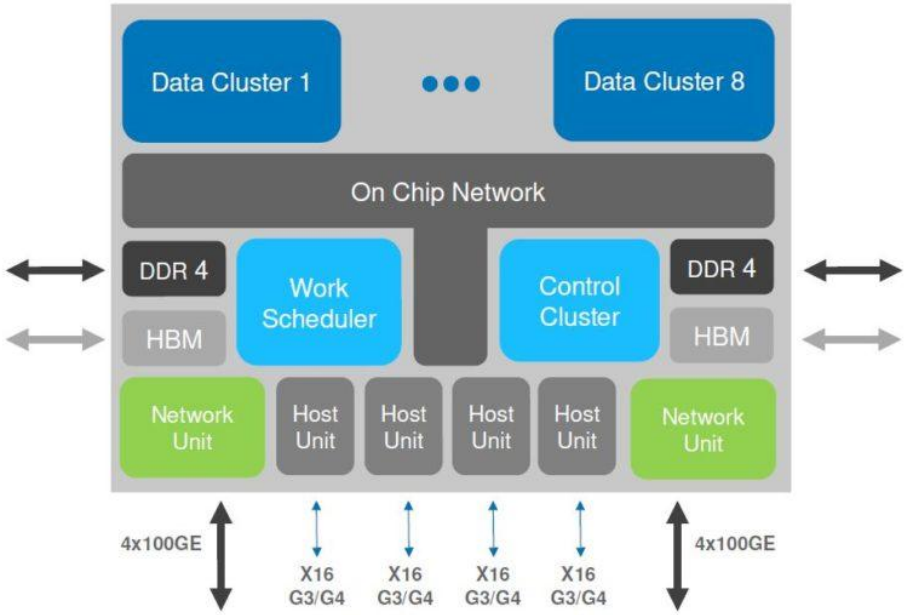


APU (Accelerated Processing Unit) - AMD



DPU (Data Processing Unit) - NVDIA

Fungible F1 DPU™ Architecture



8 Data Clusters
- 192 processor threads
- Full cache coherency
- Tightly integrated accelerators

Control Cluster
- 8 processor threads
- Secure complex
  - HSM
  - Root-of-trust
- Work scheduler

Memory & I/O Interfaces
- 800 Gbps network unit
- 4x16 G3/G4 PCIe unit
- DDR4
- HBM

On Chip Network
- High performance
- Low latency
- Any unit to any unit

FUNGIBLE

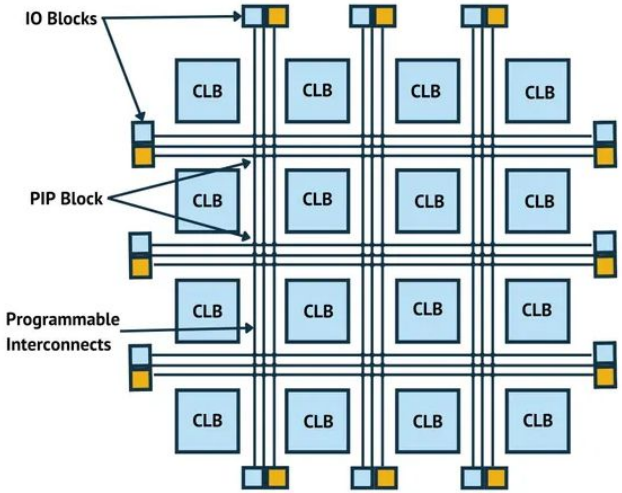
FPGA (Field Programmable Gated Array)

CPU
- Small models
- Small datasets
- Useful for design space exploration

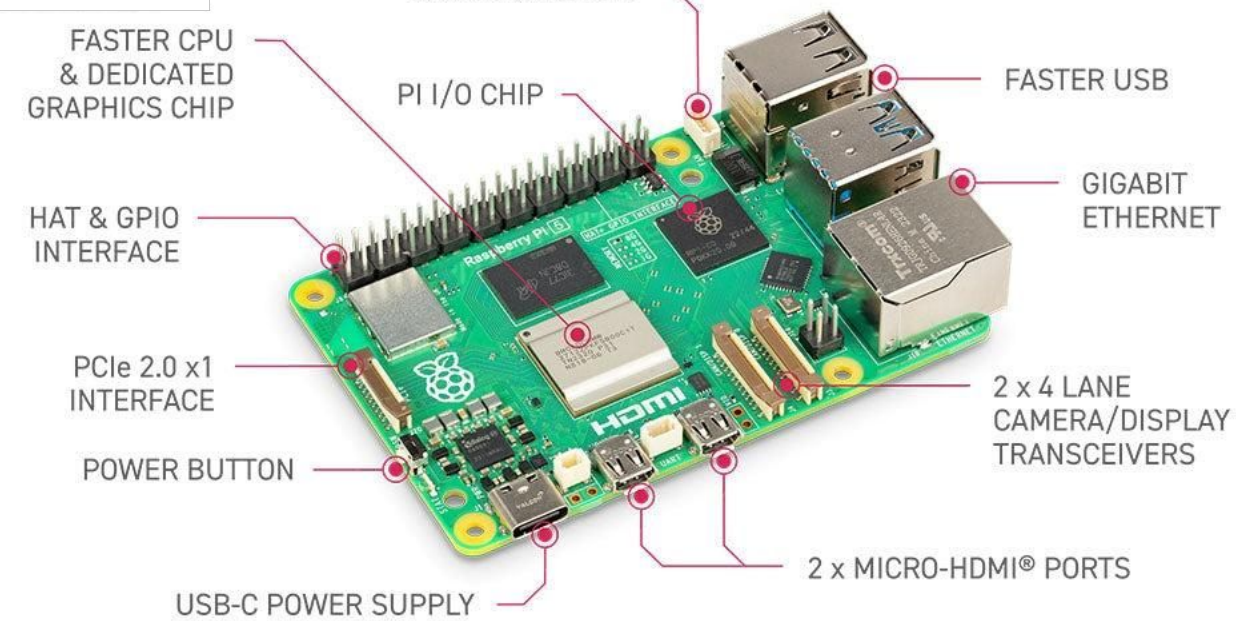
GPU
- Medium-to-large models, datasets
- Image, video processing
- Application on CUDA or OpenCL

TPU
- Matrix computations
- Dense vector processing
- No custom TensorFlow operations

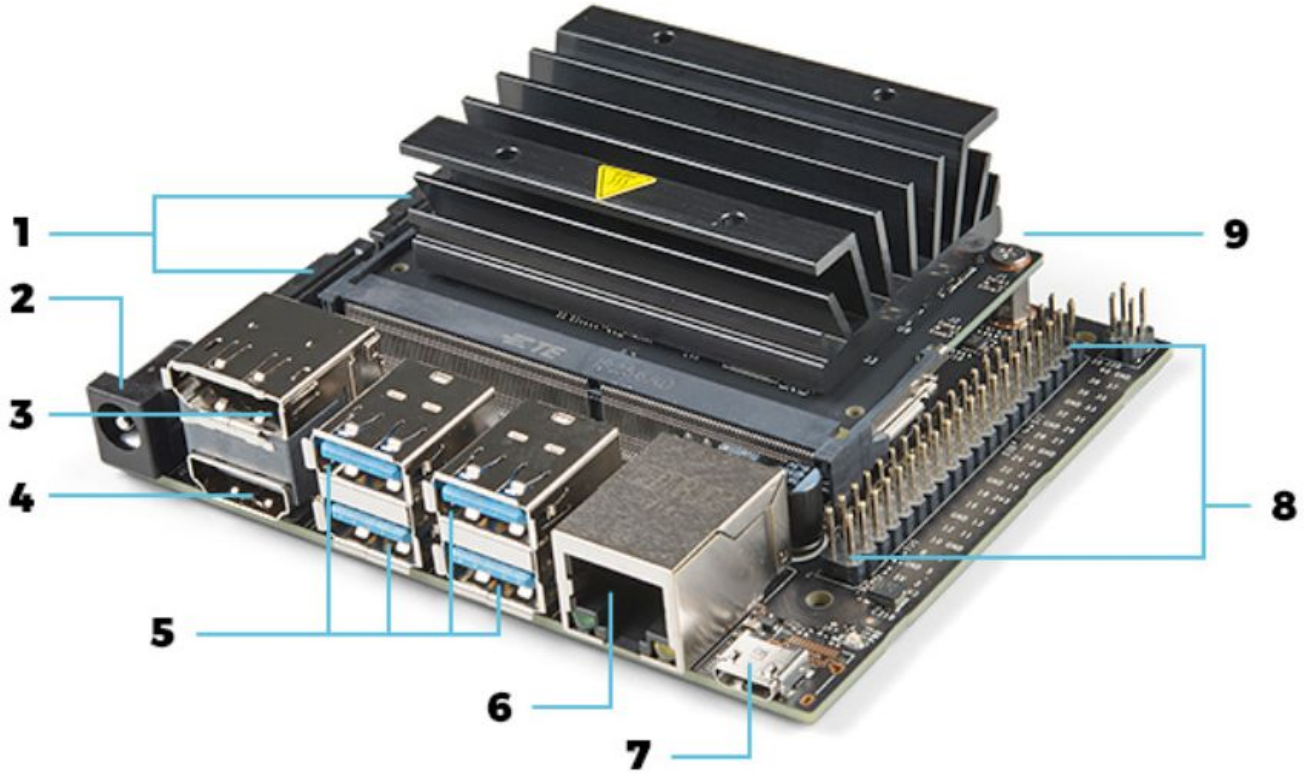
FPGA
- Large datasets, models
- Compute intensive applications
- High performance, high perf./cost ratio



Raspberry Pi



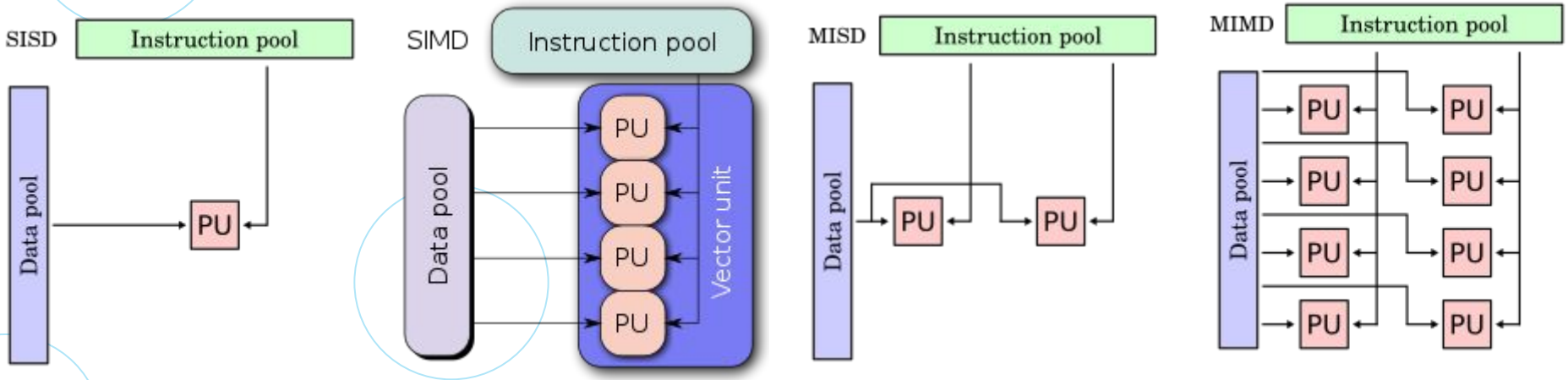
Jetson Nano

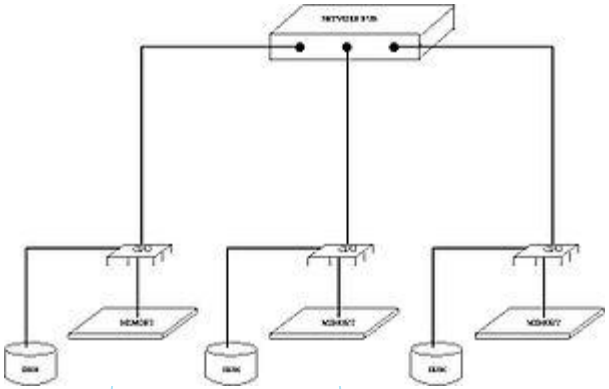


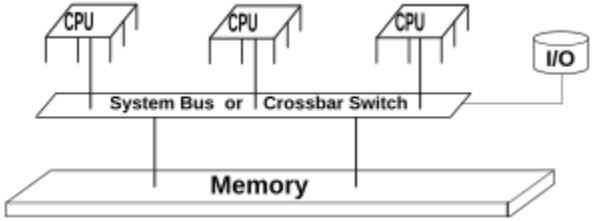
4GB edition shown above.

1. Two MIPI CSI camera connectors
2. 5V DC Barrel jack
3. DisplayPort connector
4. HDMI output port
5. 4 USB 3.0 Ports
6. Gigabit ethernet port
7. Micro-USB port for 5V power input or for data
8. 20x2 GPIO header
9. microSD storage slot

Parallel Computing



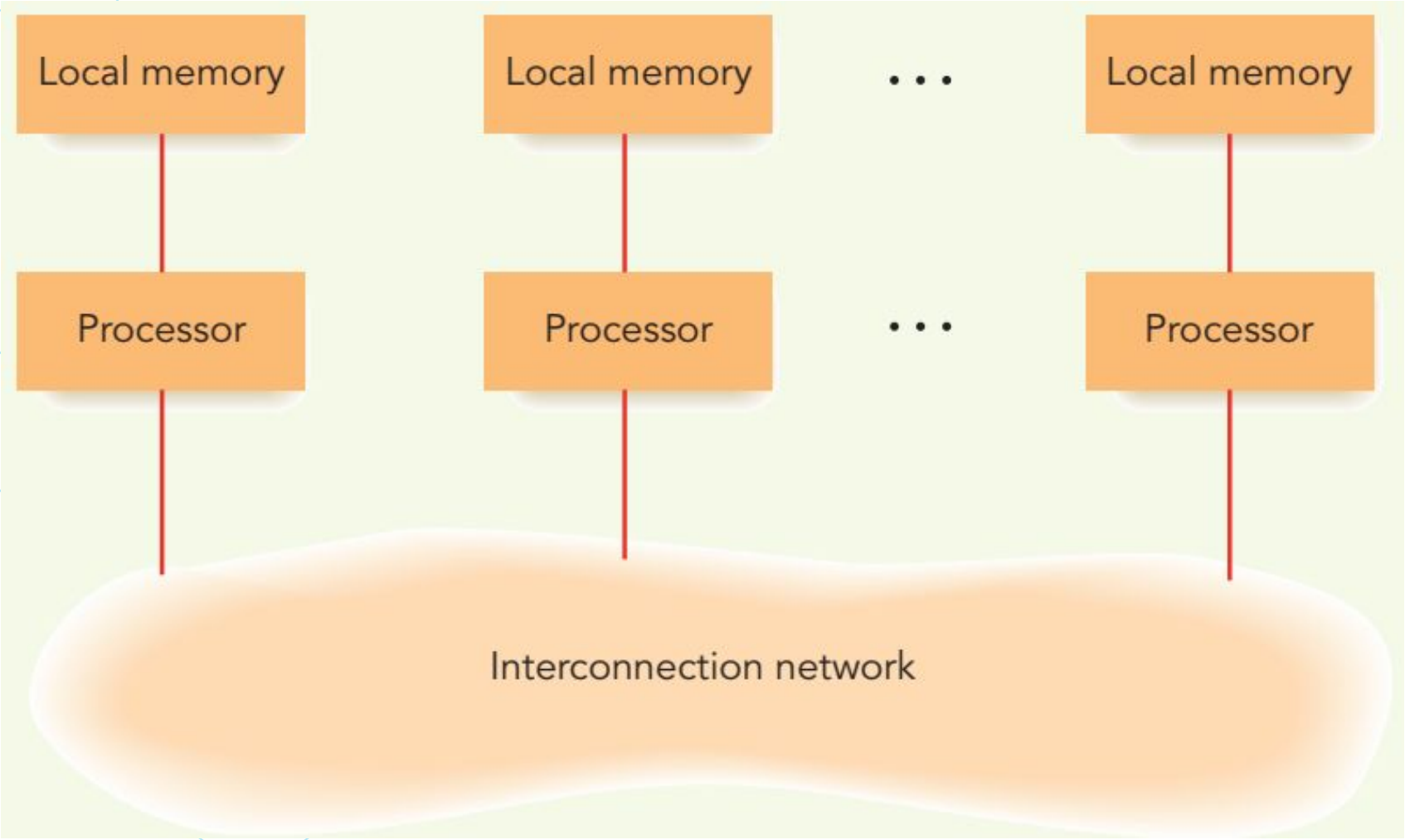




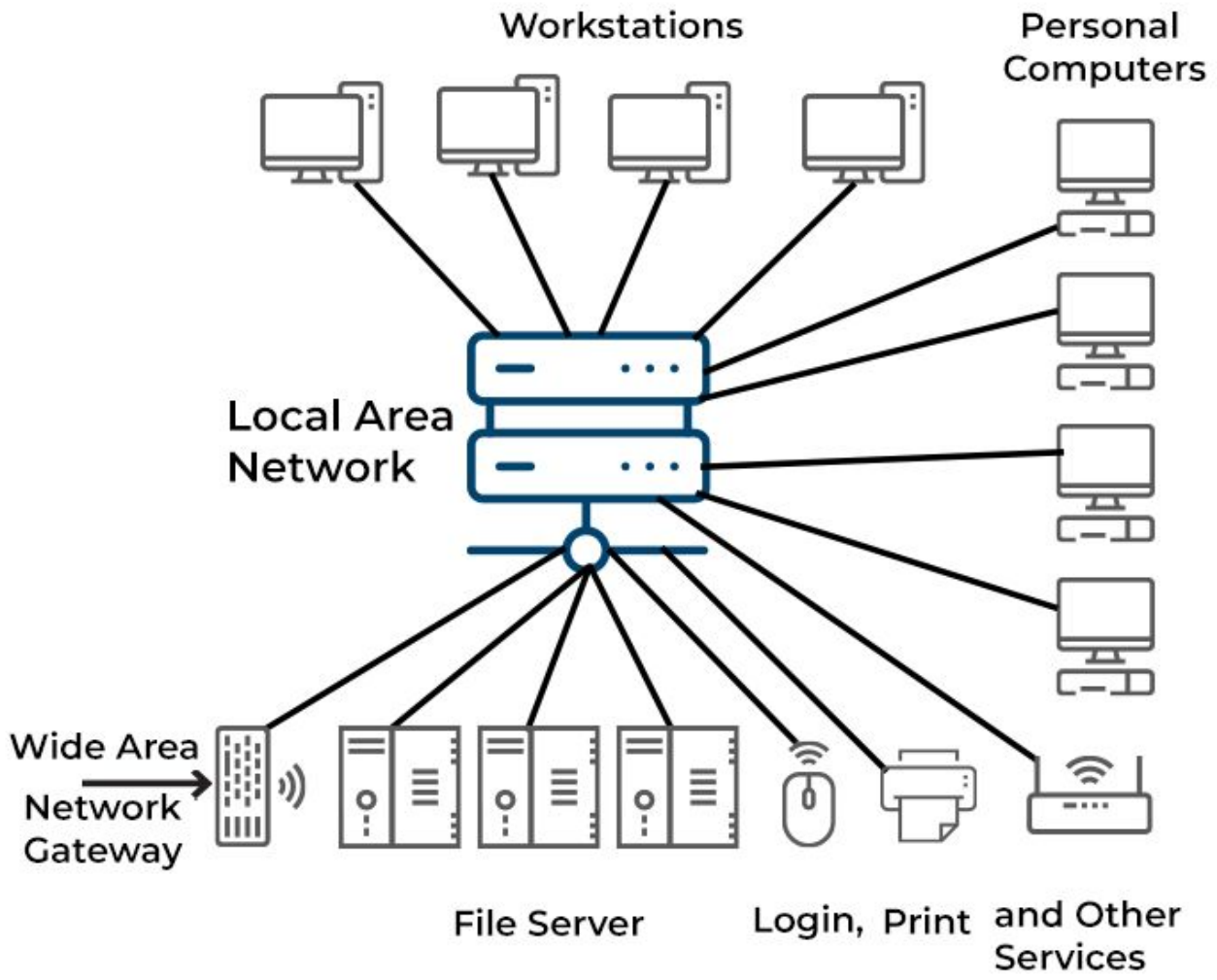
Distributed Memory

Shared Memory

Clustering Computing



Distributed System



HUST