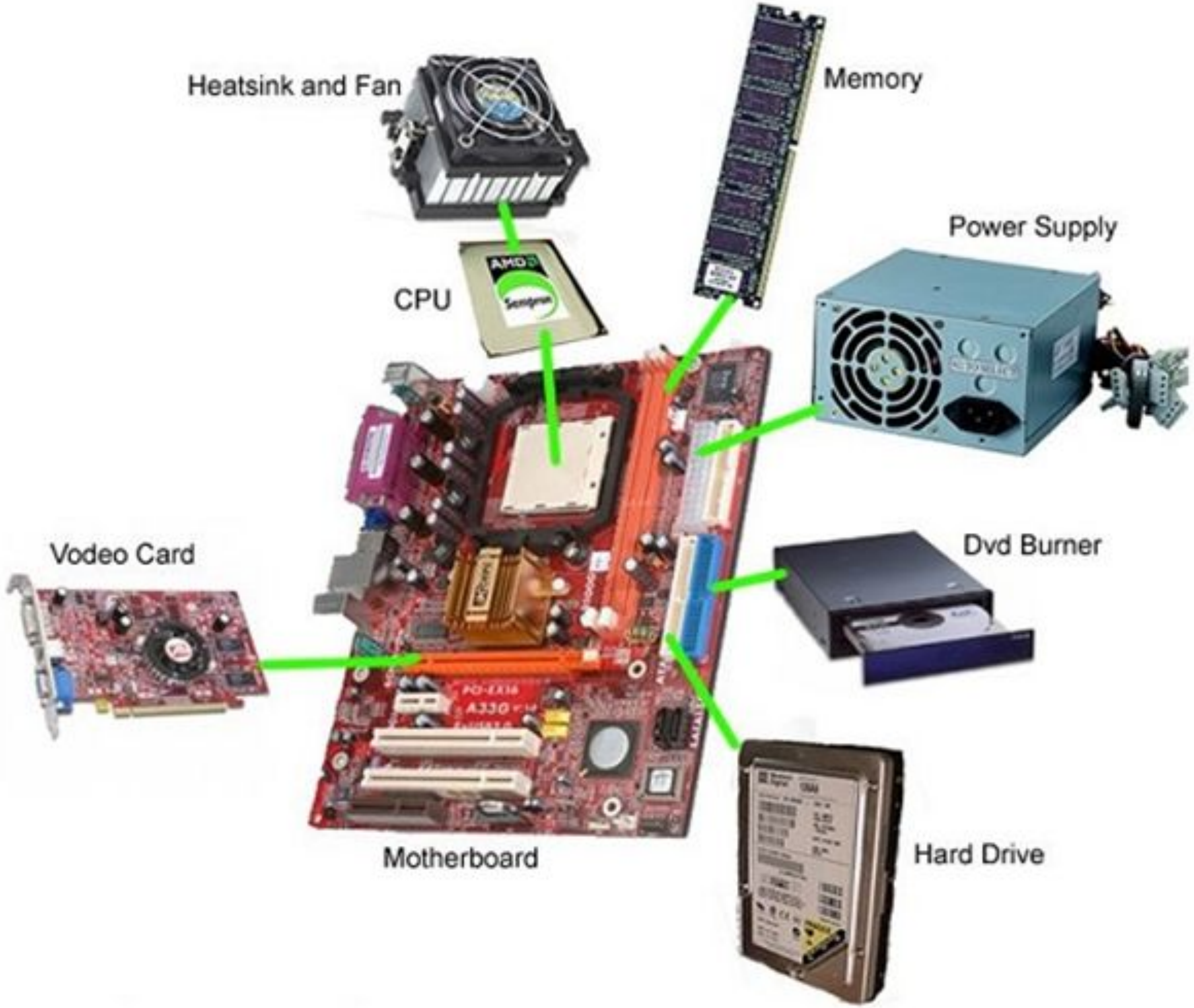
SUMMER 2024-CSE332

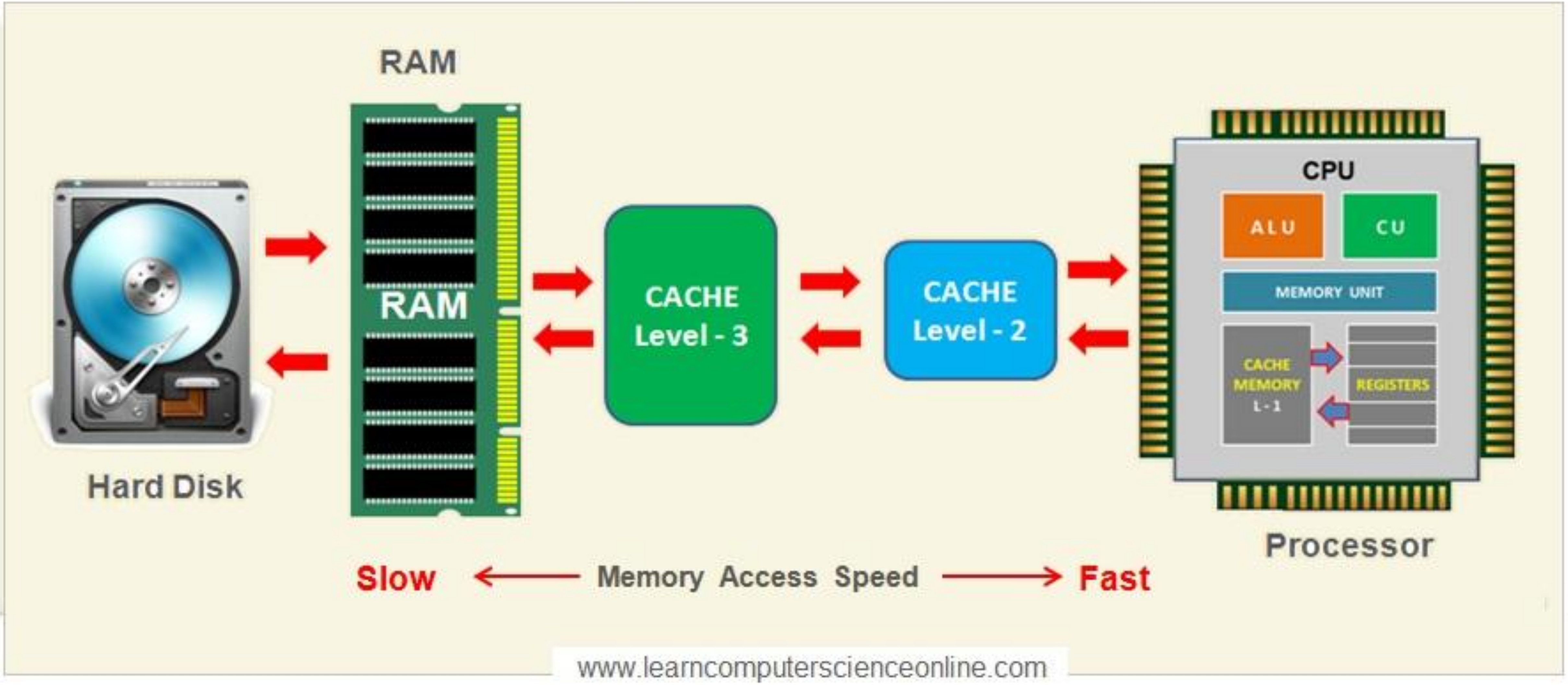
Computer Organization

and

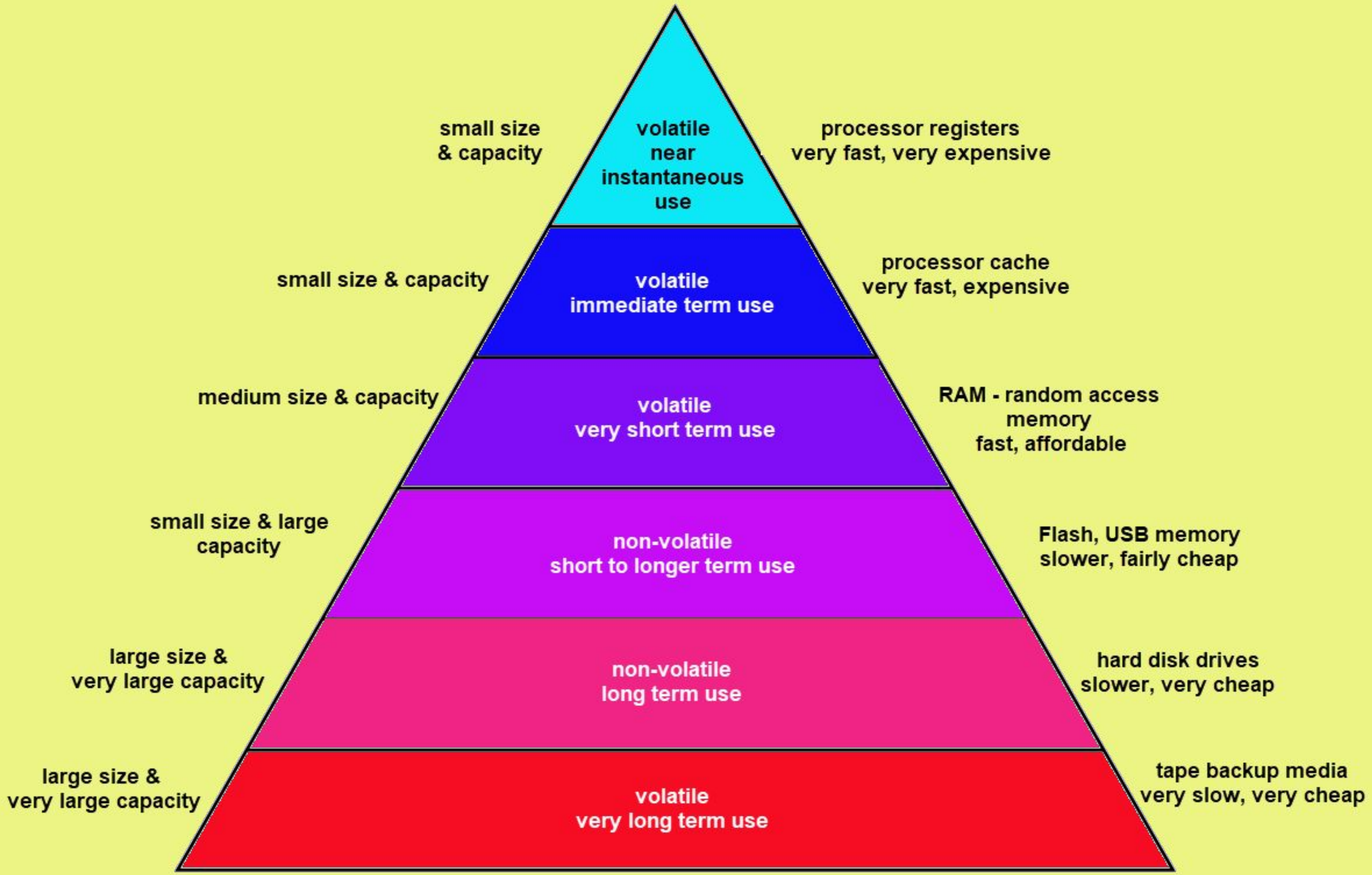
Architecture


Heatsink and Fan
Memory
Power Supply
CPU
Vodeo Card
Dvd Burner
Motherboard
Hard Drive

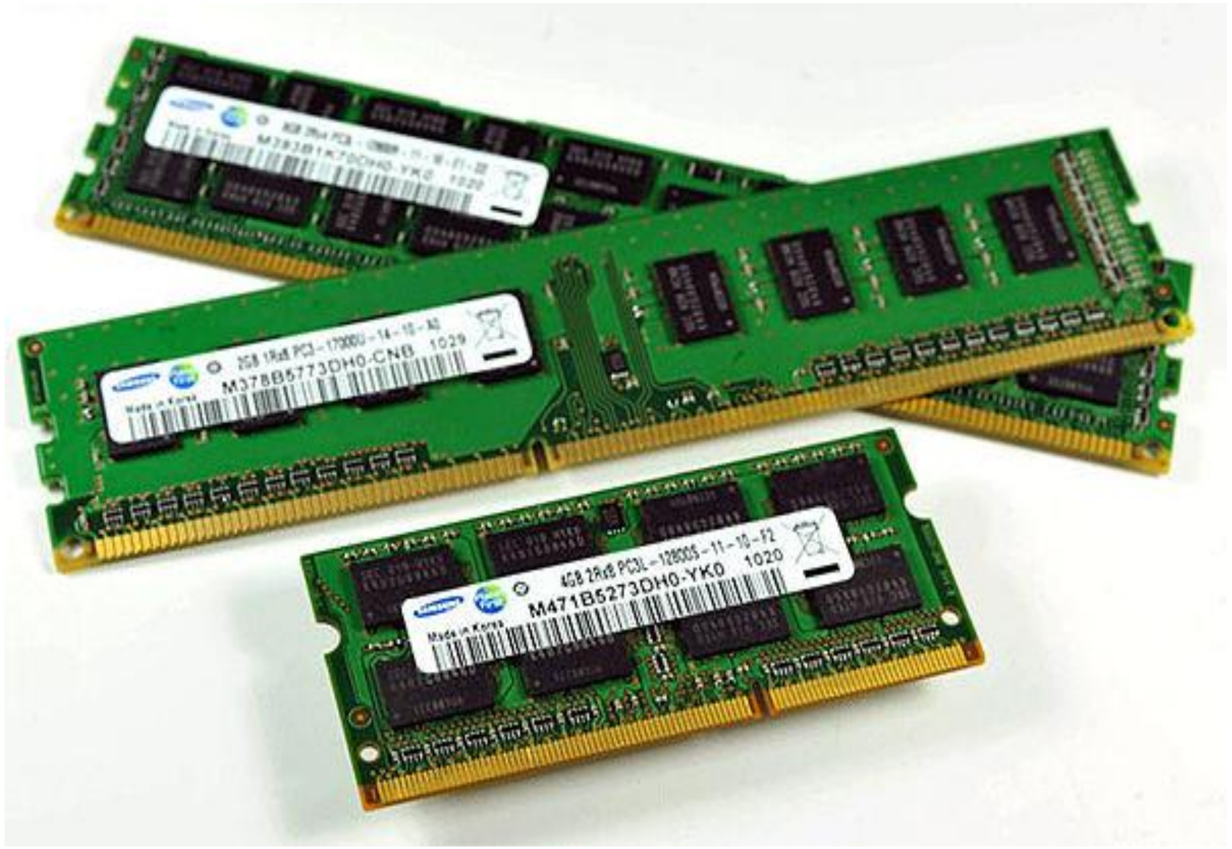
Computer System Memory Hierarchy



Computer Memory Hierarchy



Memory



Memory

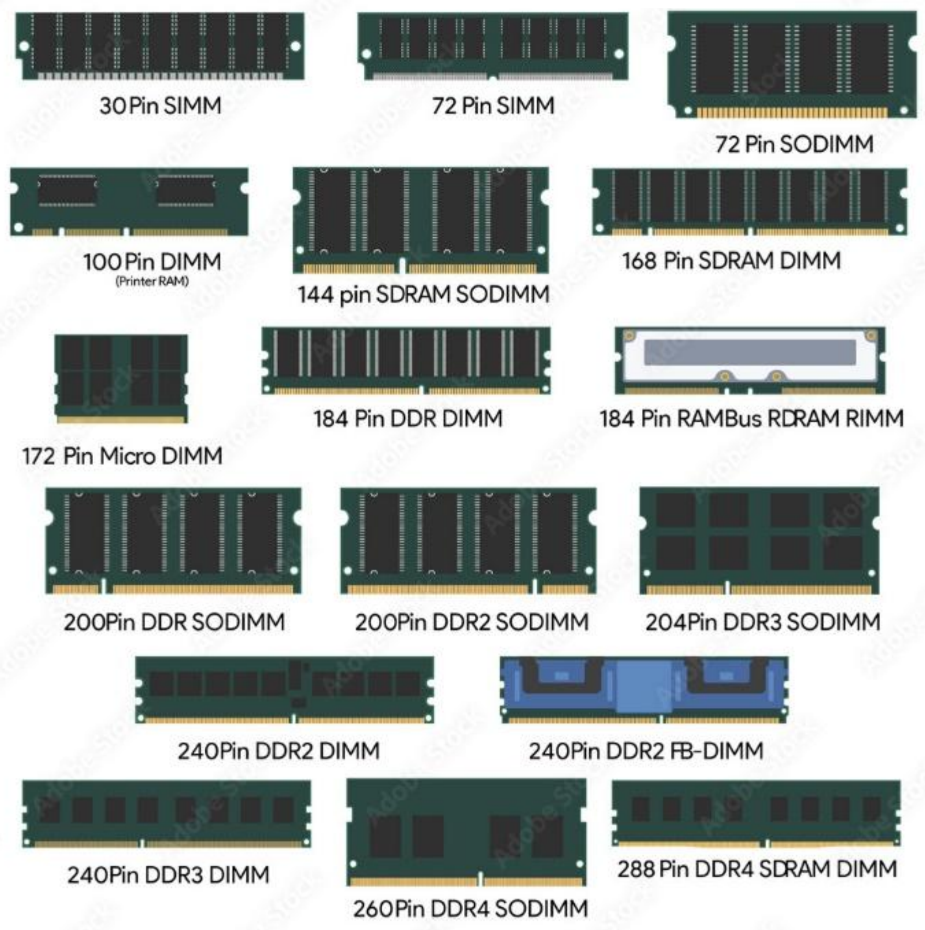
- CPU has only few registers but most computational tasks require a lot more memory.
- Main memory is the next fastest memory within a computer and is much larger in size.
- RAM (Random Access Memory) is the most common form of Main Memory. RAM is normally located on the motherboard.
- ROM (Read Only Memory) is like RAM except that its contents cannot be overwritten and its contents are not lost if power is turned off (ROM is non-volatile).

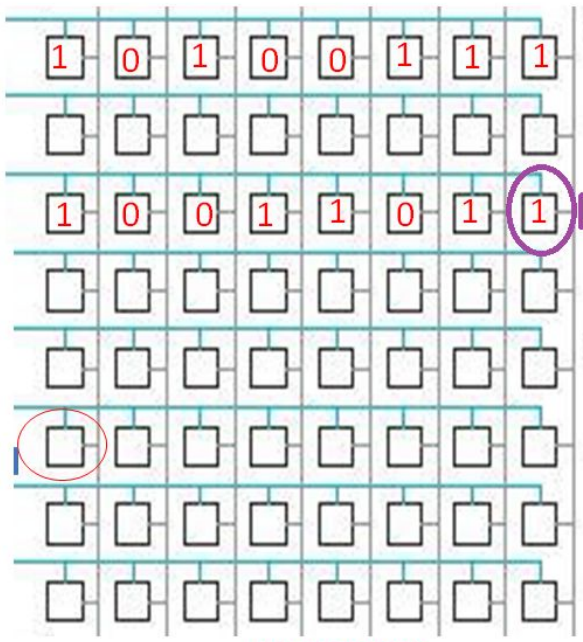
- Although slower than register memory, the contents of any location in RAM can still be “read” or “written” very quickly. The time to read or write is referred to as the **access time** and is constant for all RAM locations.
- RAM is used to hold both program code (instructions) and data (numbers, strings etc).
- Programs are “loaded” into RAM from a disk prior to execution by the CPU.
- Locations in RAM are identified by an **addressing scheme** *e.g.* numbering the bytes in RAM from 0 onwards.

Types of RAM

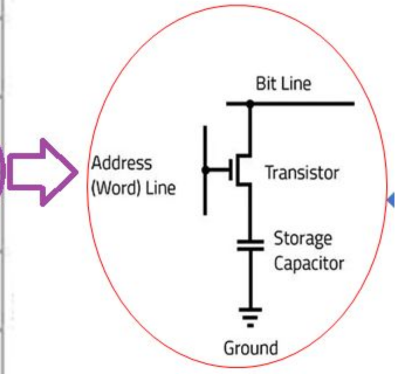
- There are many kinds of RAM and new ones are invented all the time. One of aims is to make RAM access as fast as possible in order to keep up with the increasing speed of CPUs.
- SRAM (Static RAM) is the fastest form of RAM but also the most expensive. Due to its cost it is not used as main memory but rather for cache memory. Each bit requires a 6-transistor circuit.
- DRAM (Dynamic RAM) is not as fast as SRAM but is cheaper and is used for main memory. Each bit uses a single capacitor and single transistor circuit. Since capacitors lose their charge, DRAM needs to be refreshed every few milliseconds. The memory system does this transparently. There are many implementations of DRAM, two well-known ones are SDRAM and DDR SDRAM.
- SDRAM (Synchronous DRAM) is a form of DRAM that is synchronised with the clock of the CPU's system bus, sometimes called the front-side bus (FSB). As an example, if the system bus operates at 167Mhz over an 8-byte (64-bit) data bus , then an SDRAM module could transfer 167 x 8 ~ 1.3GB/sec.
- DDR SDRAM (Double-Data Rate DRAM) is an optimisation of SDRAM that allows data to be transferred on both the rising edge and falling edge of a clock signal. Effectively doubling the amount of data that can be transferred in a period of time. For example a PC-3200 DDR-SDRAM module operating at 200Mhz can transfer 200 x 8 x 2 ~ 3.2GB/sec over an 8-byte (64-bit) data bus.

Memory is an array of storage, each having capacity of 8 bits or so and holds program and data in binary format

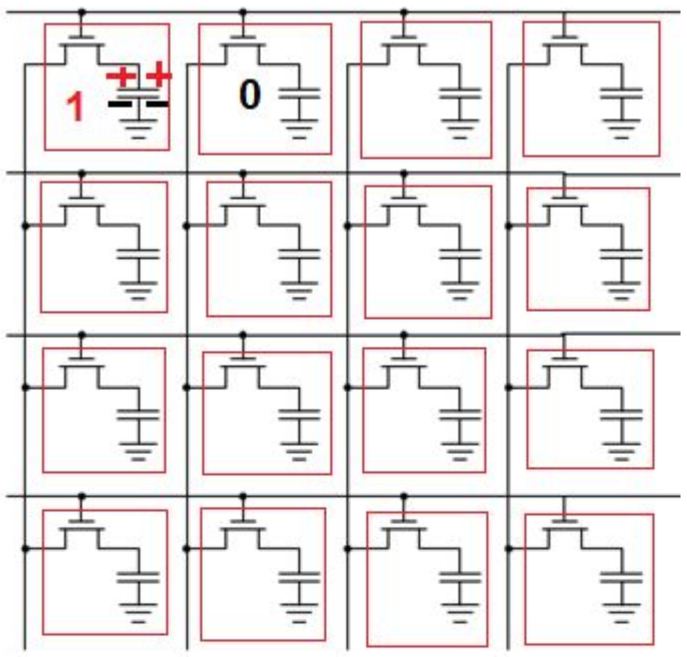




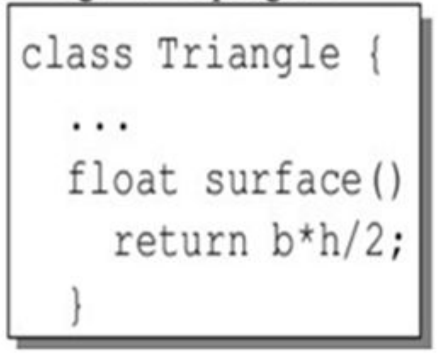
Internal of IC



Single Memory Cell



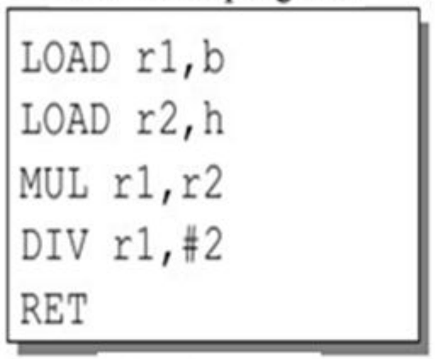
High-level program



```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

COMPILER

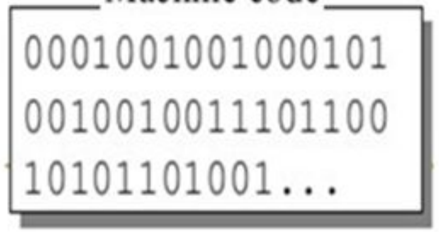
Low-level program



```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

INTERPRETER

Machine code



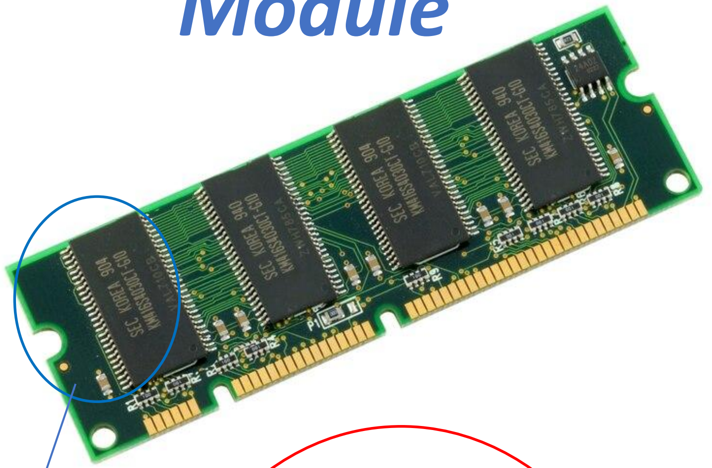
```
0001001001000101
0010010011101100
10101101001...
```

Memory is an array of storage, each having capacity of 8 bits or so and holds program and data in binary format

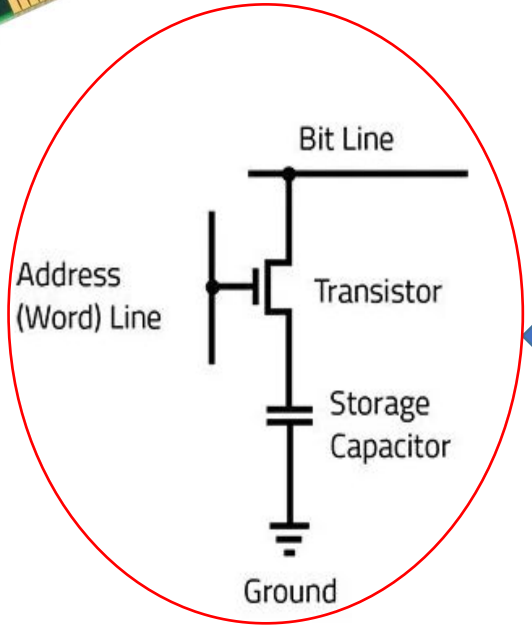
User Program & Data

0101000001111111
1100000111100000
0101010100001111
1110000000111111
0101000001111111
1100000111100000
0101010100001111
1110000000111111

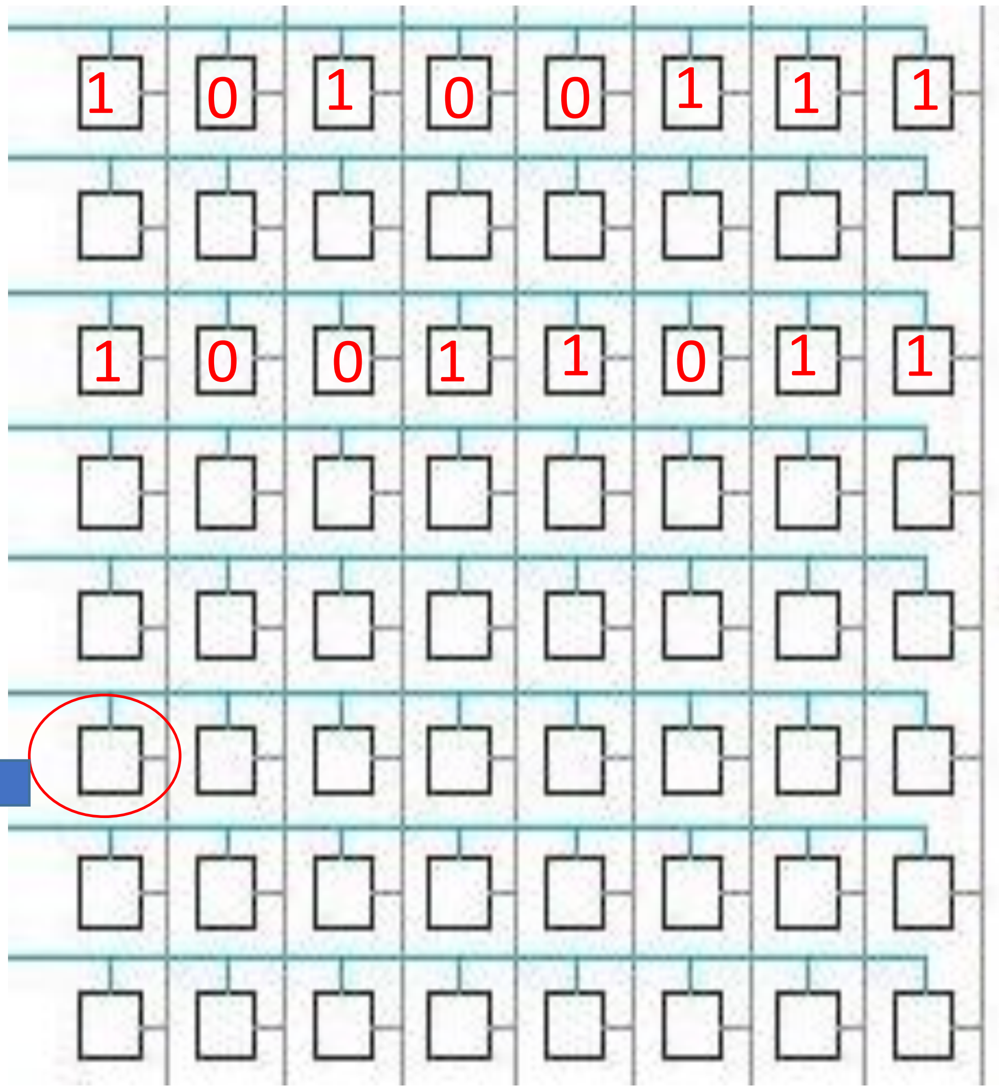
Module



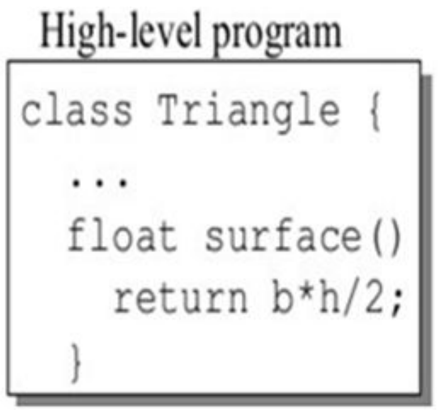
IC



Single Memory Cell



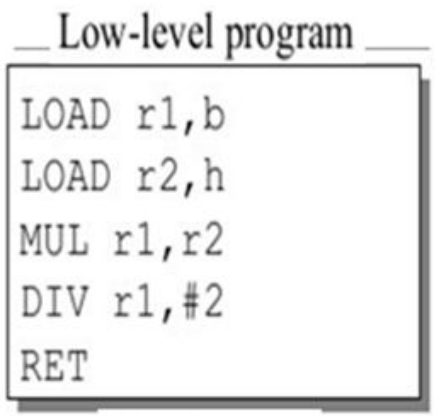
Internal of IC



High-level program

```
class Triangle {
  ...
  float surface()
    return b*h/2;
  }
```

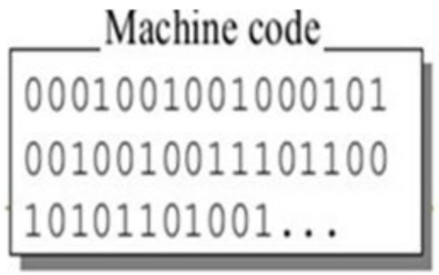
COMPILER



Low-level program

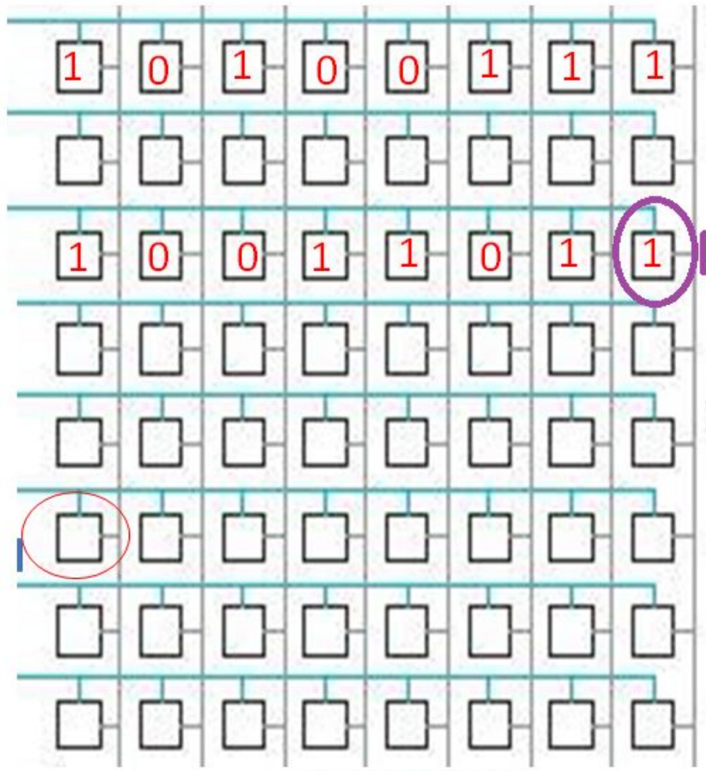
```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

INTERPRETER

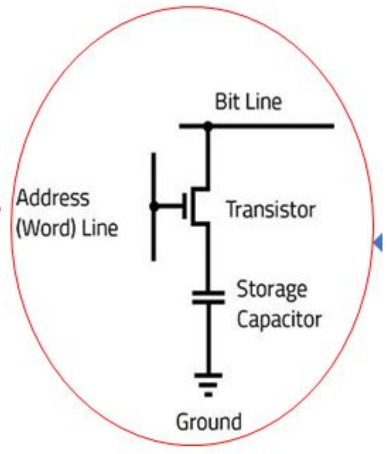


Machine code

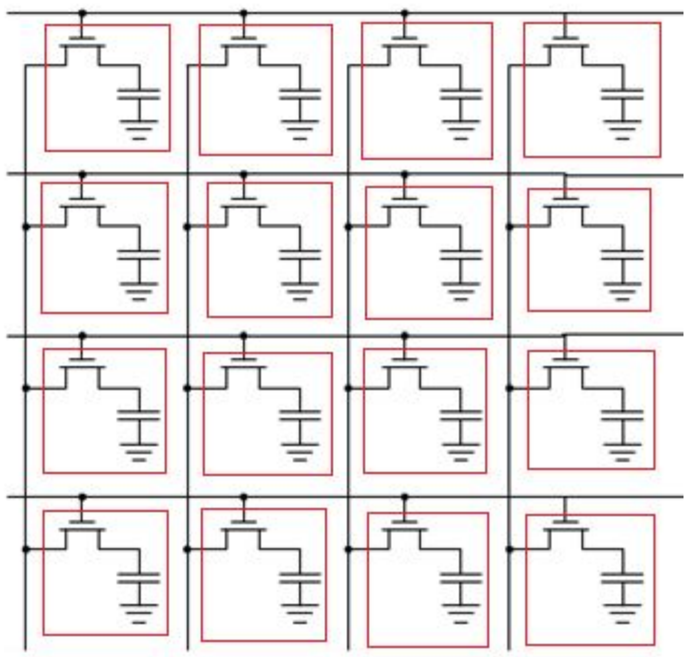
```
0001001001000101
0010010011101100
10101101001...
```

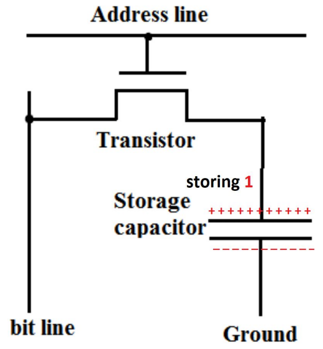


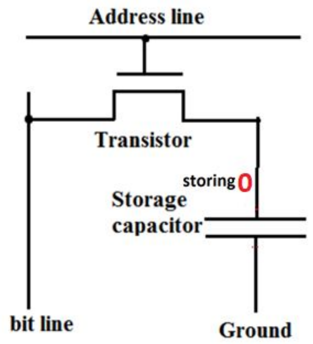
Internal of IC



Single Memory Cell

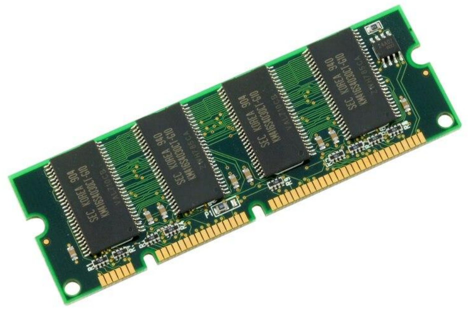


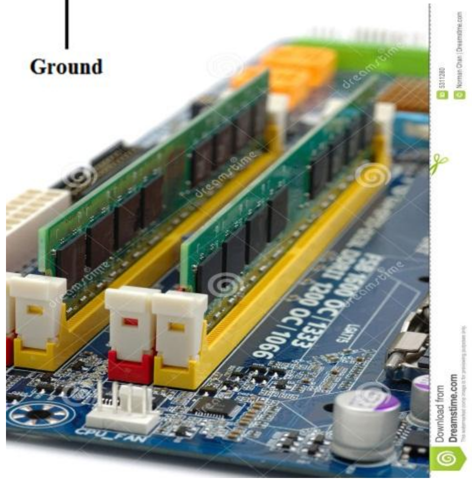




Program and Data

```
0101000001111111
1100000111100000
0101010100001111
1110000000111111
0101000001111111
1100000111100000
0101010100001111
1110000000111111
```

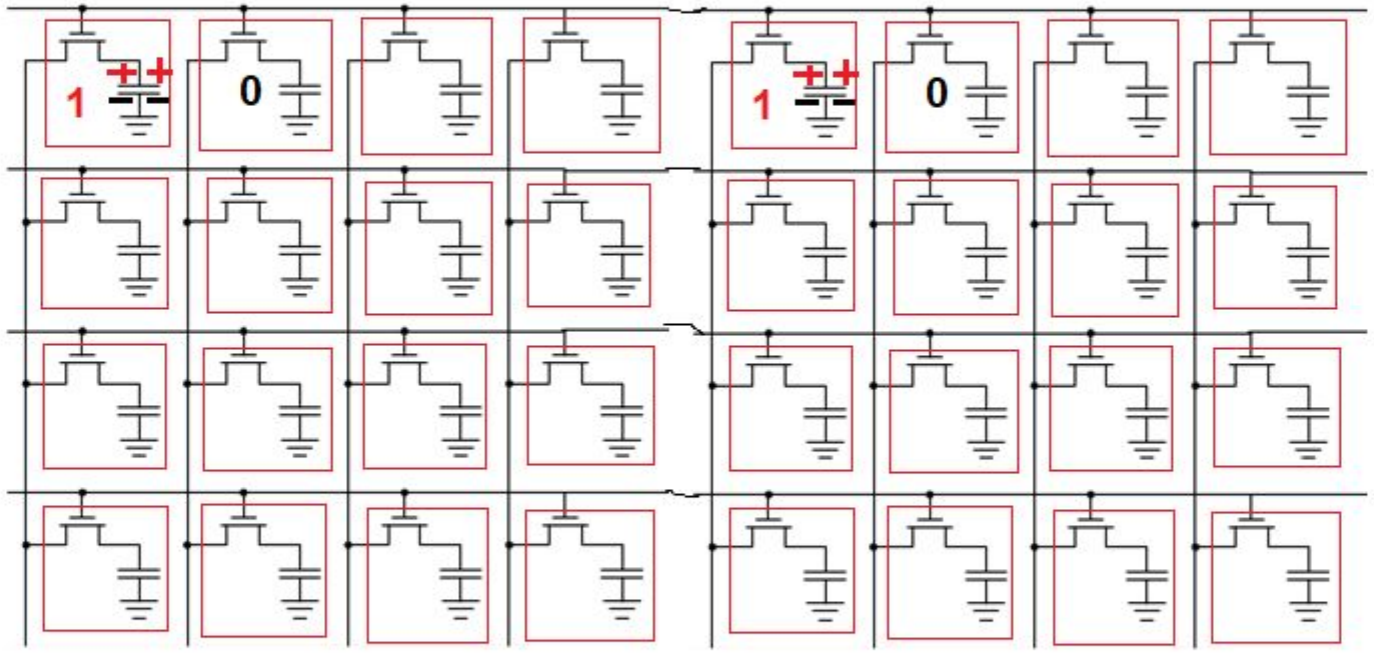


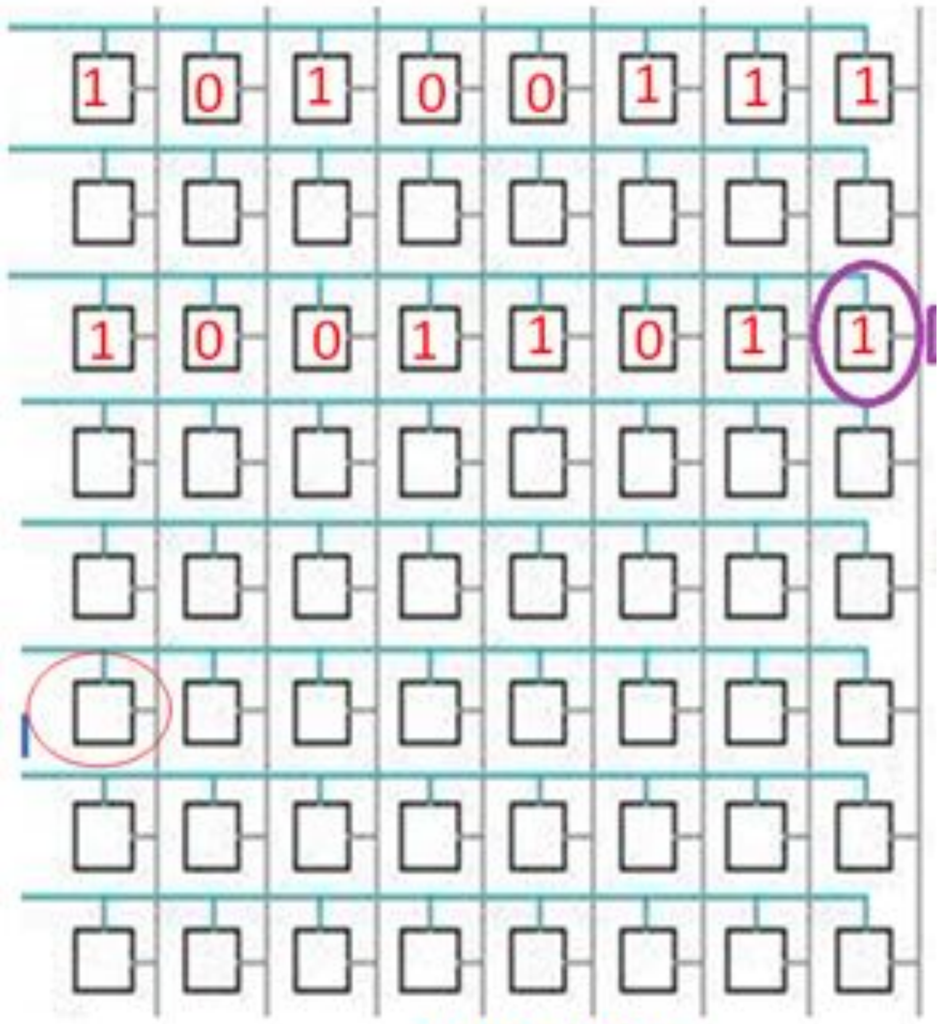


Stored in RAM

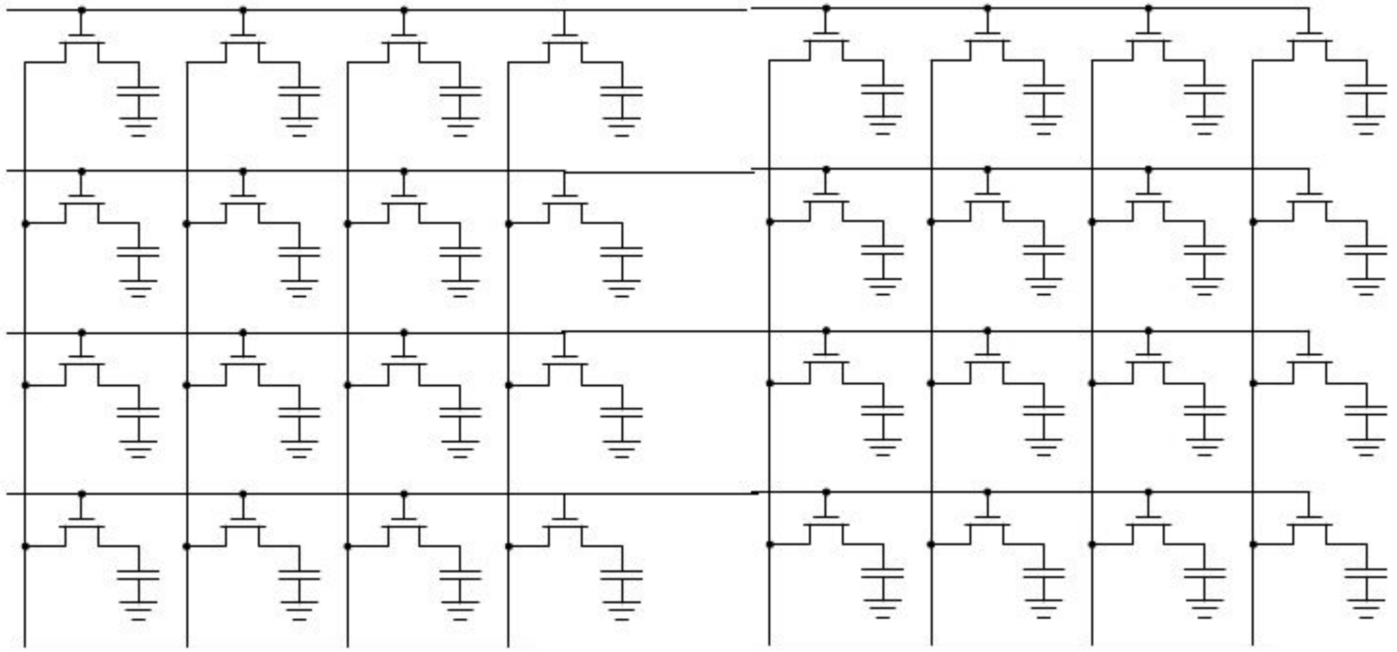
Main Memory

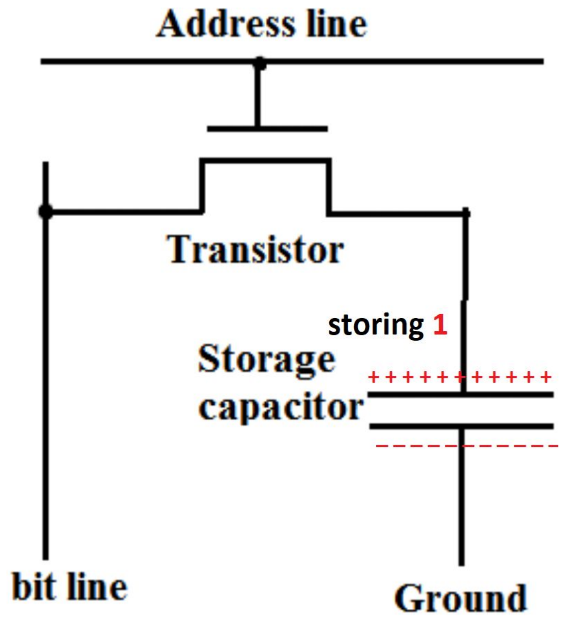
- Main memory can be considered to be organised as a matrix of bits.
- Each row represents a memory location, typically this is 1 Byte (8 cells contain 8 bits)

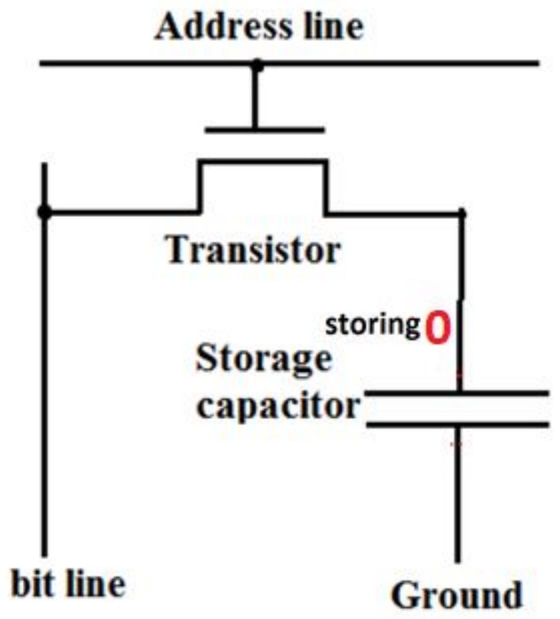


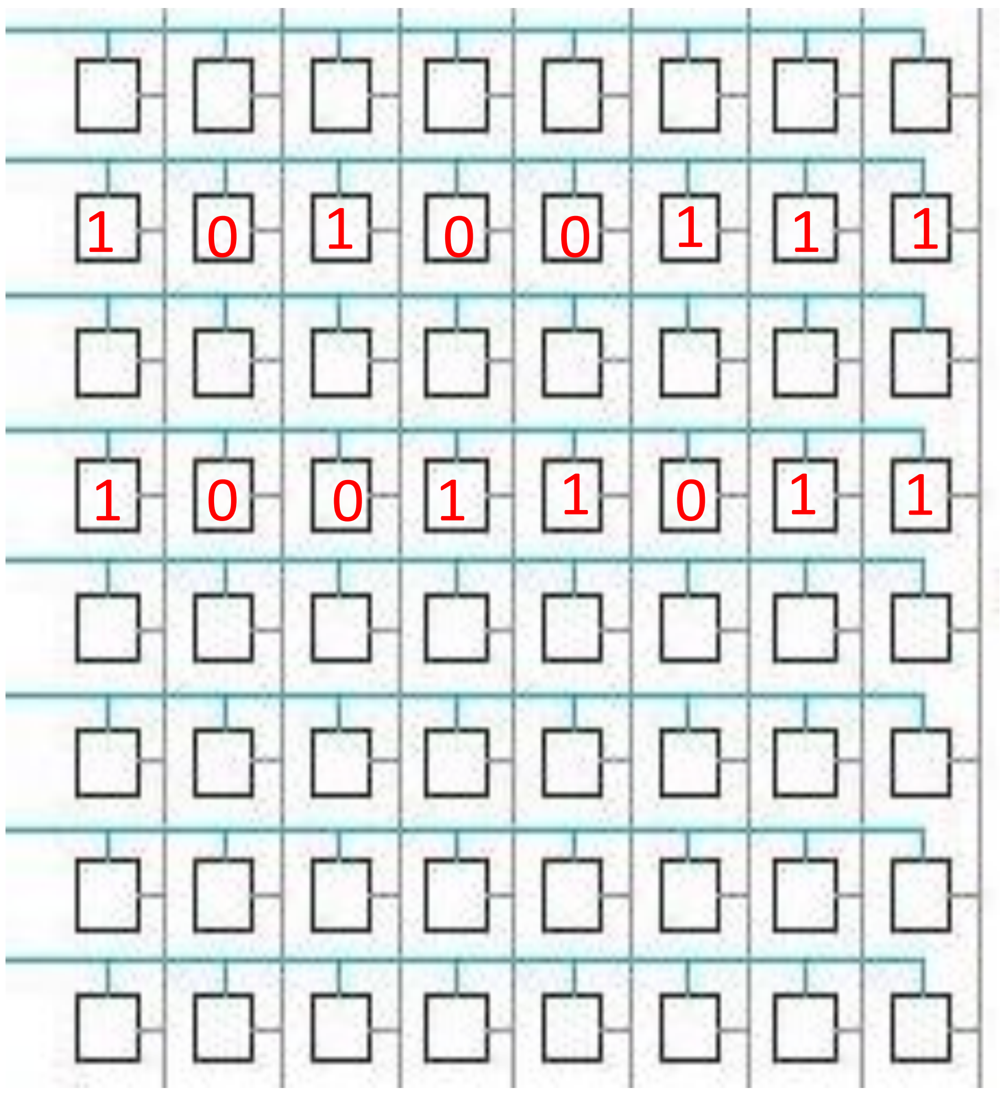


Internal of IC

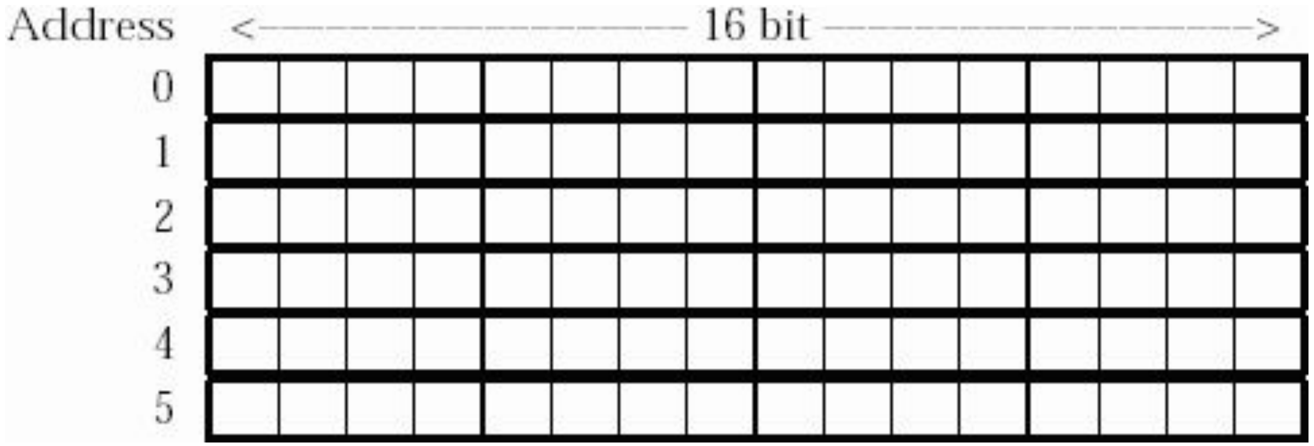


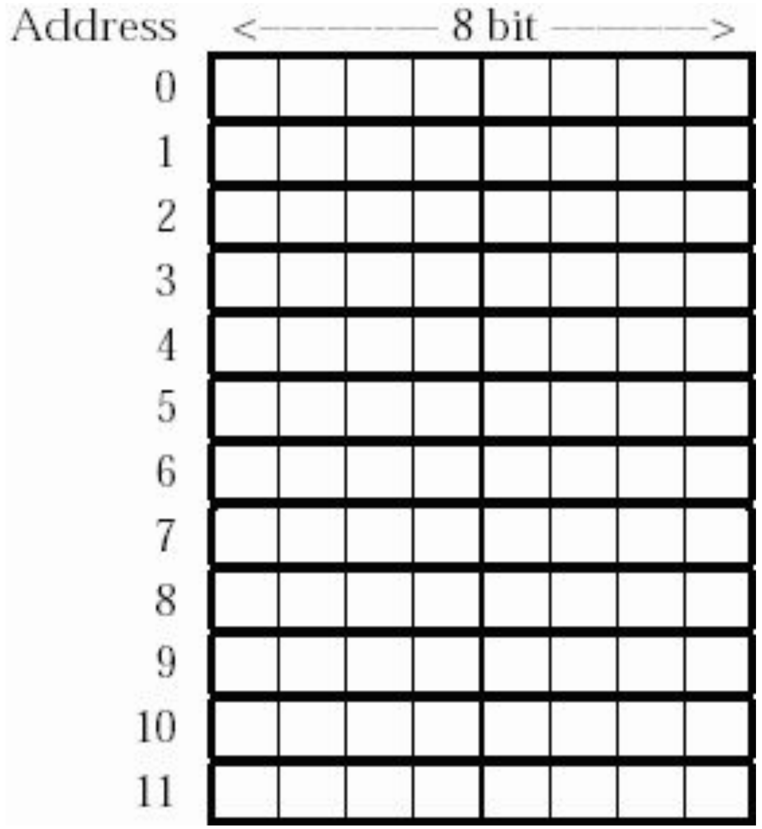

Address line
Transistor
storing 1
Storage capacitor
bit line
Ground

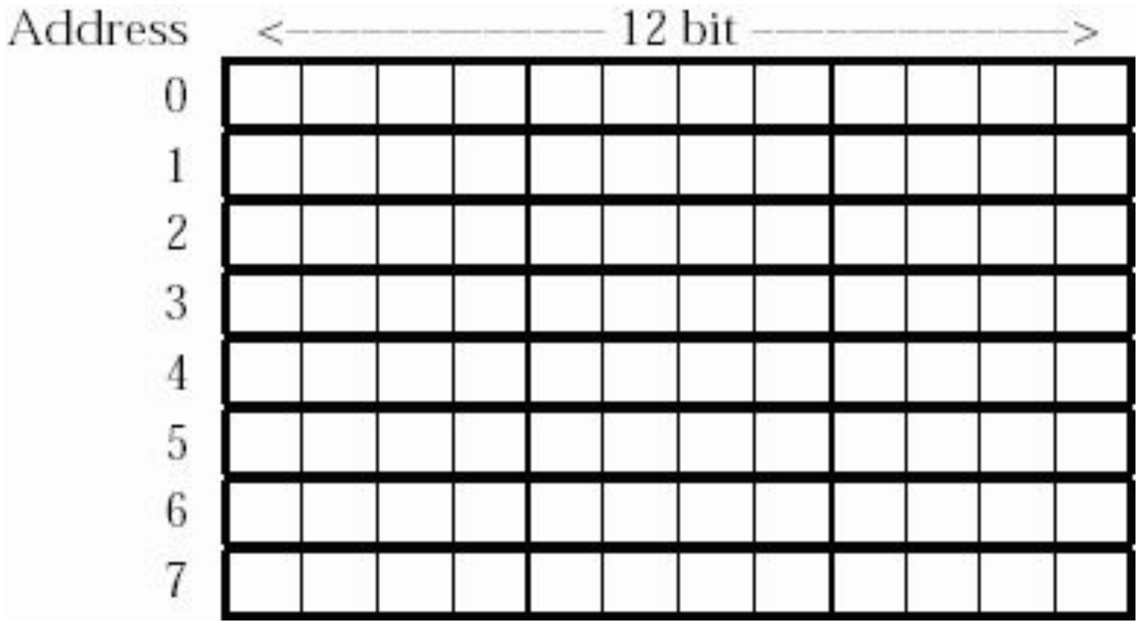

Address line
Transistor
storing 0
Storage capacitor
bit line
Ground


1 0 1 0 0 1 1 1
1 0 0 1 1 0 1 1

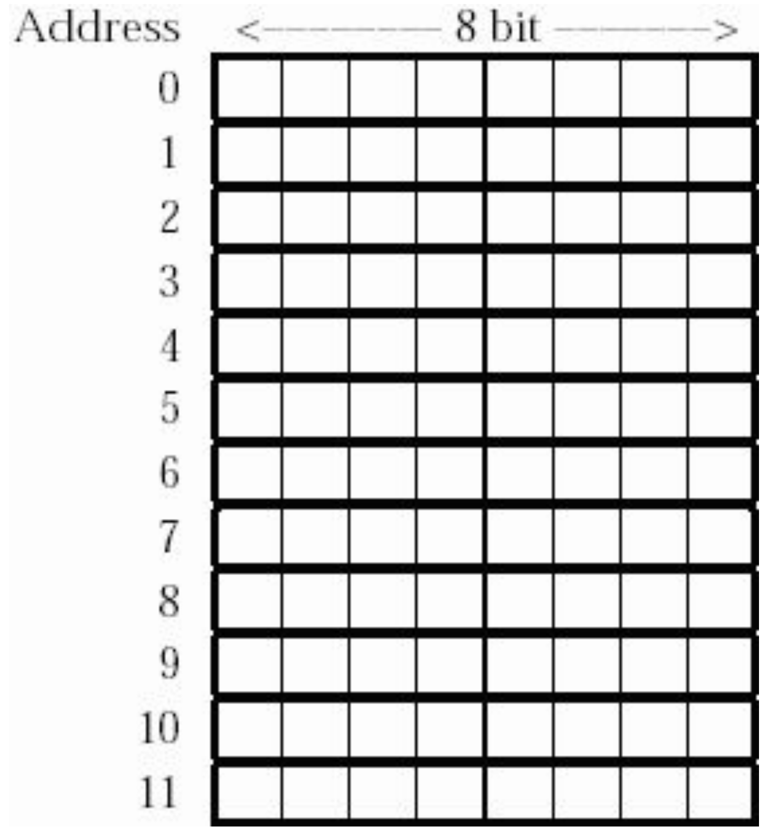
- For a 96-bit memory we could organise the memory as 12x8 bits, or 8x12 bits or, 6x16 bits, or even as 96x1 bits or 1x96 bits.
- Each row also has a natural number called its **address** which is used for selecting the row







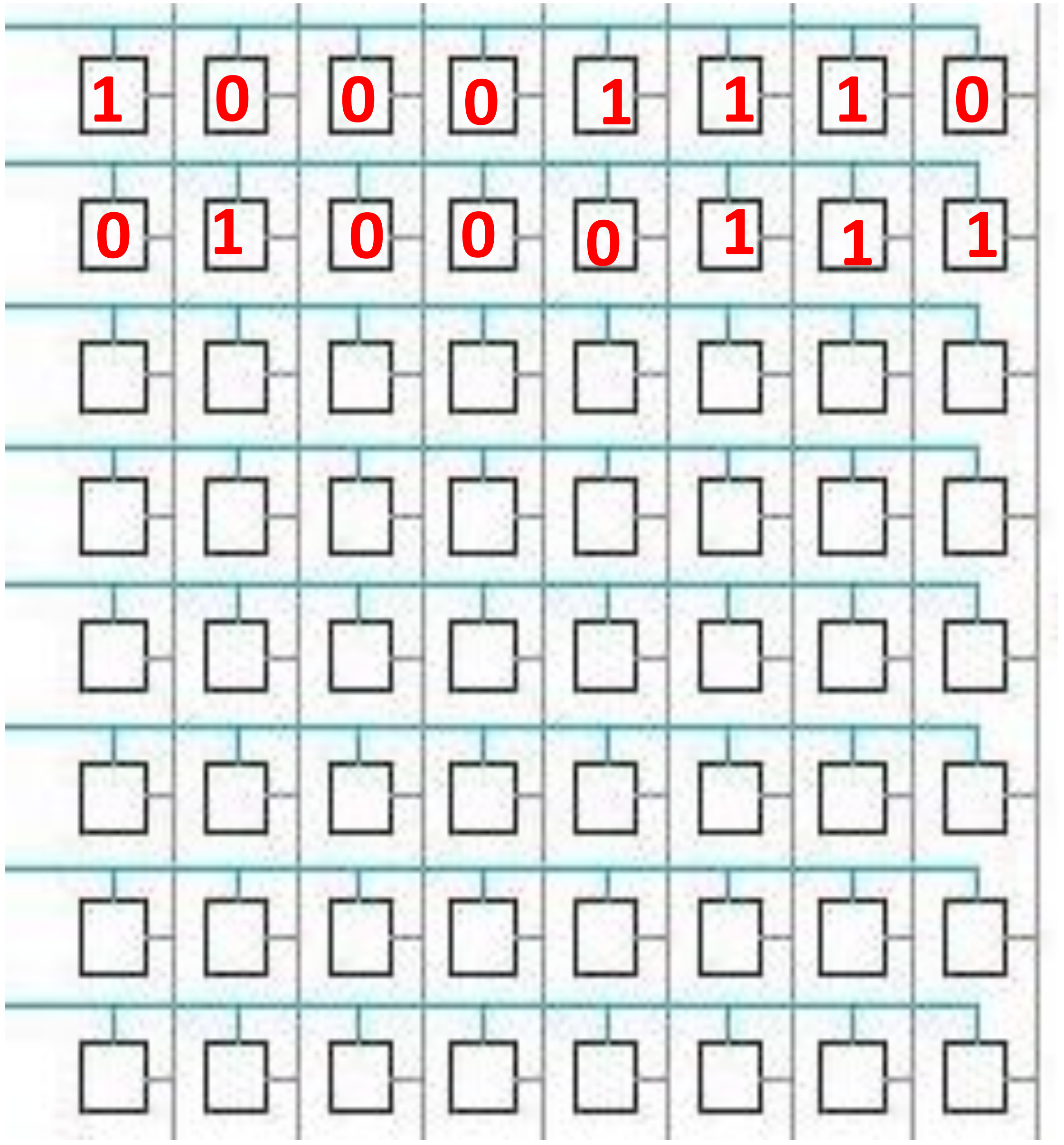
- Most architectures however, make Main Memory byte-addressable.
- In such architectures the CPU and/or the Main Memory hardware is capable of reading/writing any individual byte.



Each row is uniquely identified by a number/code, starting from ‘0’, called Address, usually represented in Hexadecimal form and used in Assembly language programming

Address in HEX	Address in Binary			
0H	0	0	0	0
1H	0	0	0	1
2H	0	0	1	0
3H	0	0	1	1
7H	0	1	1	1

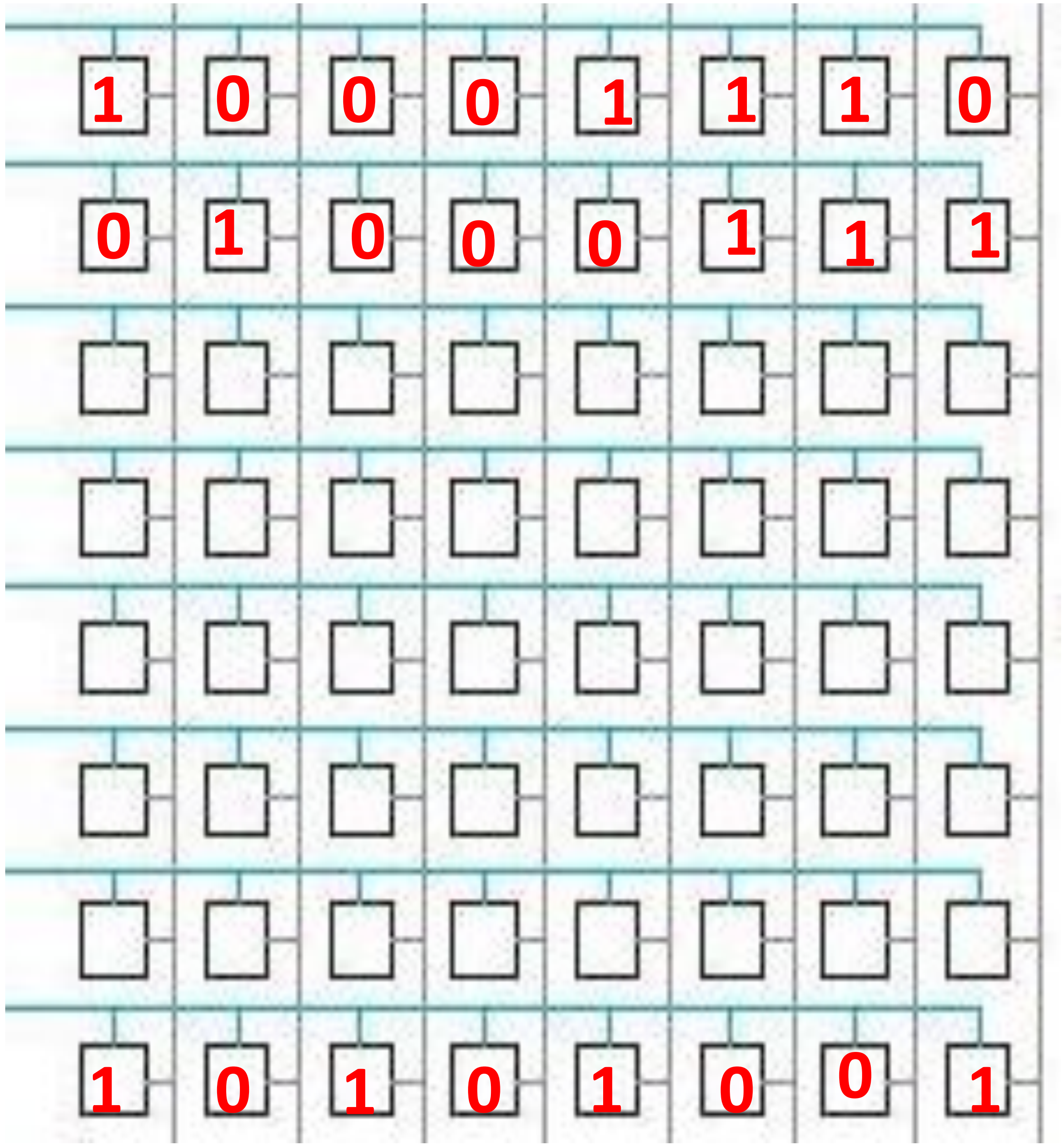
Contents (Machine code & data)



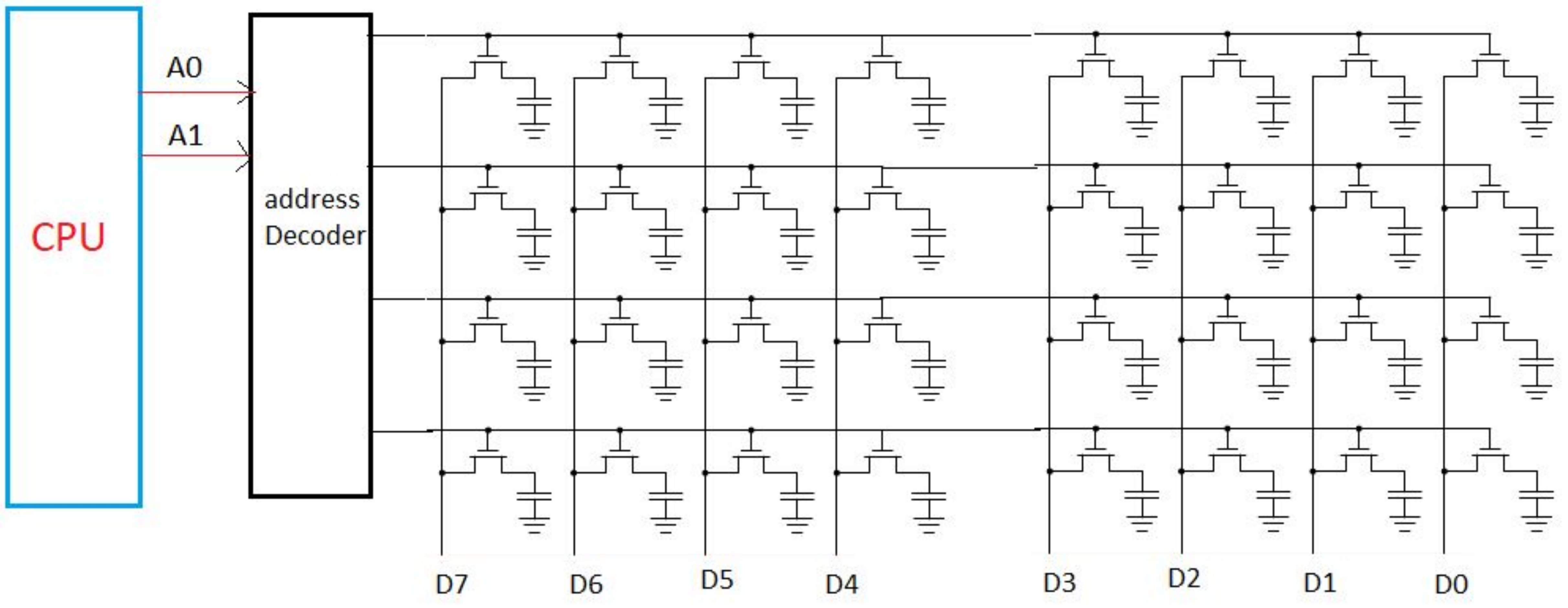
If each location of memory contains 8 bits or 1 byte, then it is called

Byte-Addressable memory

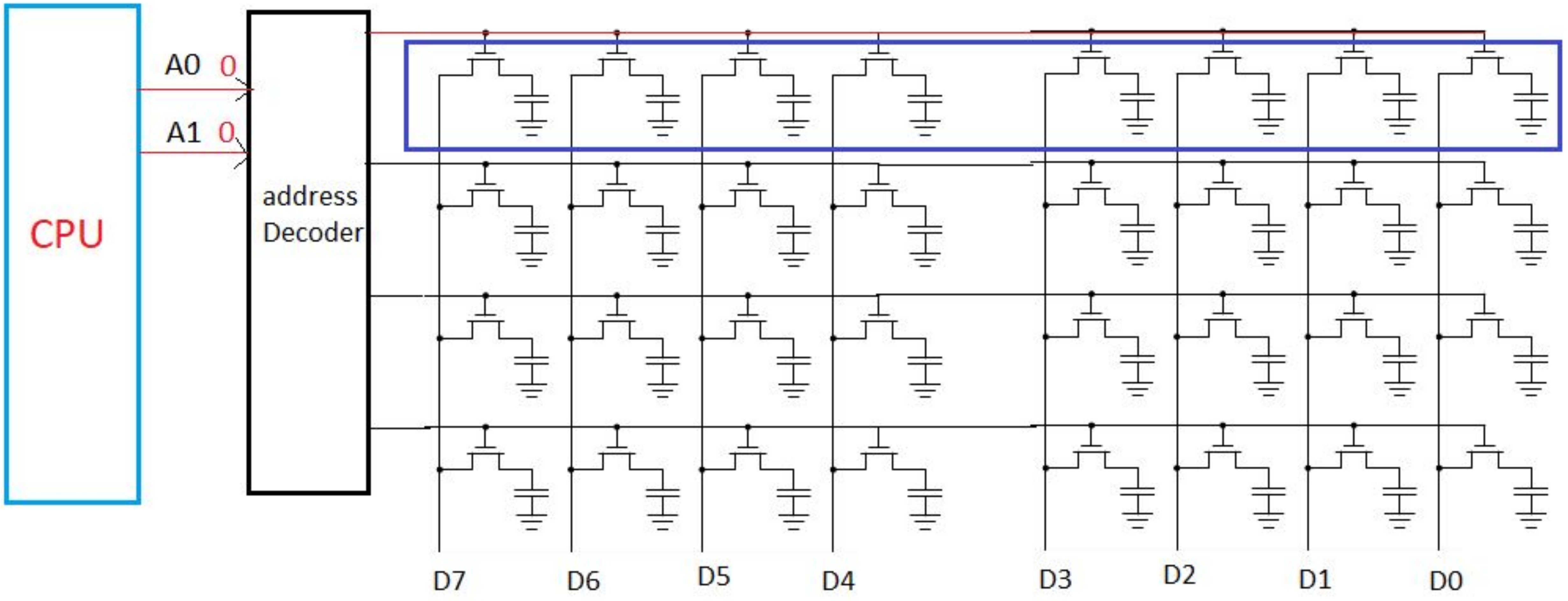
Address in HEX	Address in Binary				Contents (Machine code or data)
0H	0	0	0	0	1 0 0 0 1 1 1 0
1H	0	0	0	1	0 1 0 0 0 1 1 1
2H	0	0	1	0	
3H	0	0	1	1	
7H	0	1	1	1	1 0 1 0 1 0 0 1



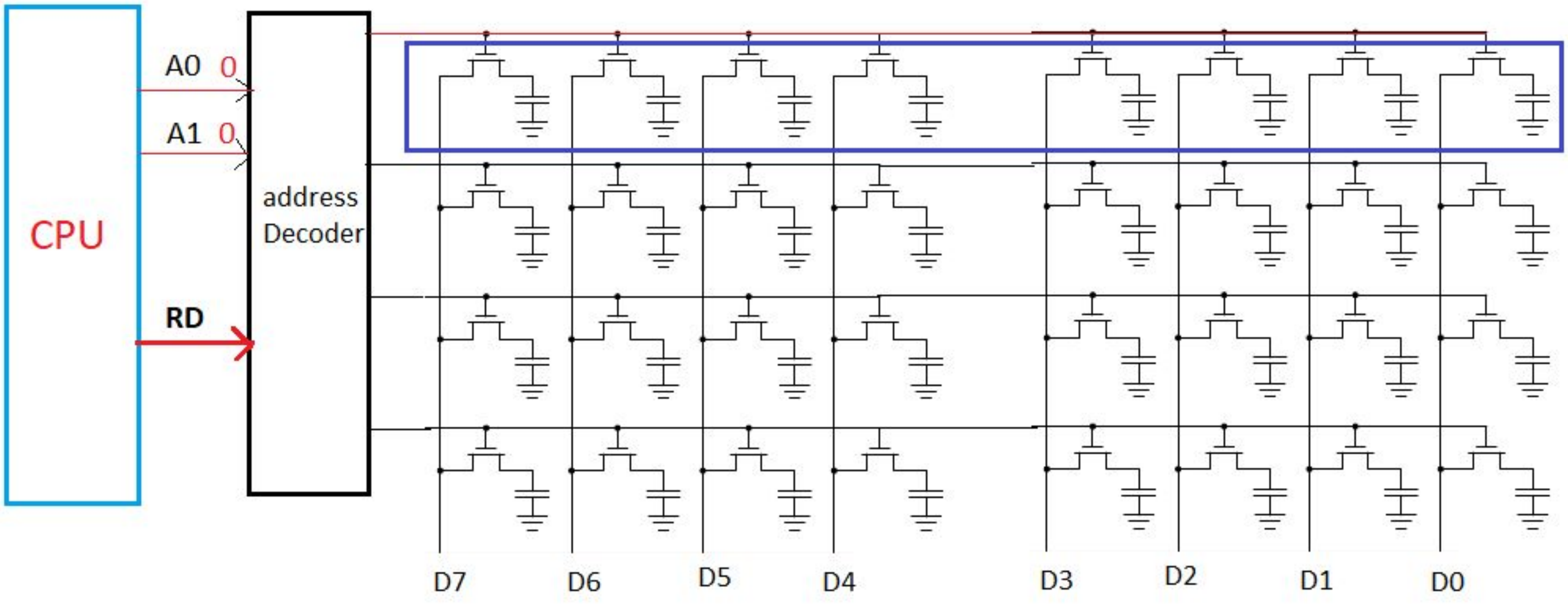
Interfacing CPU-RAM



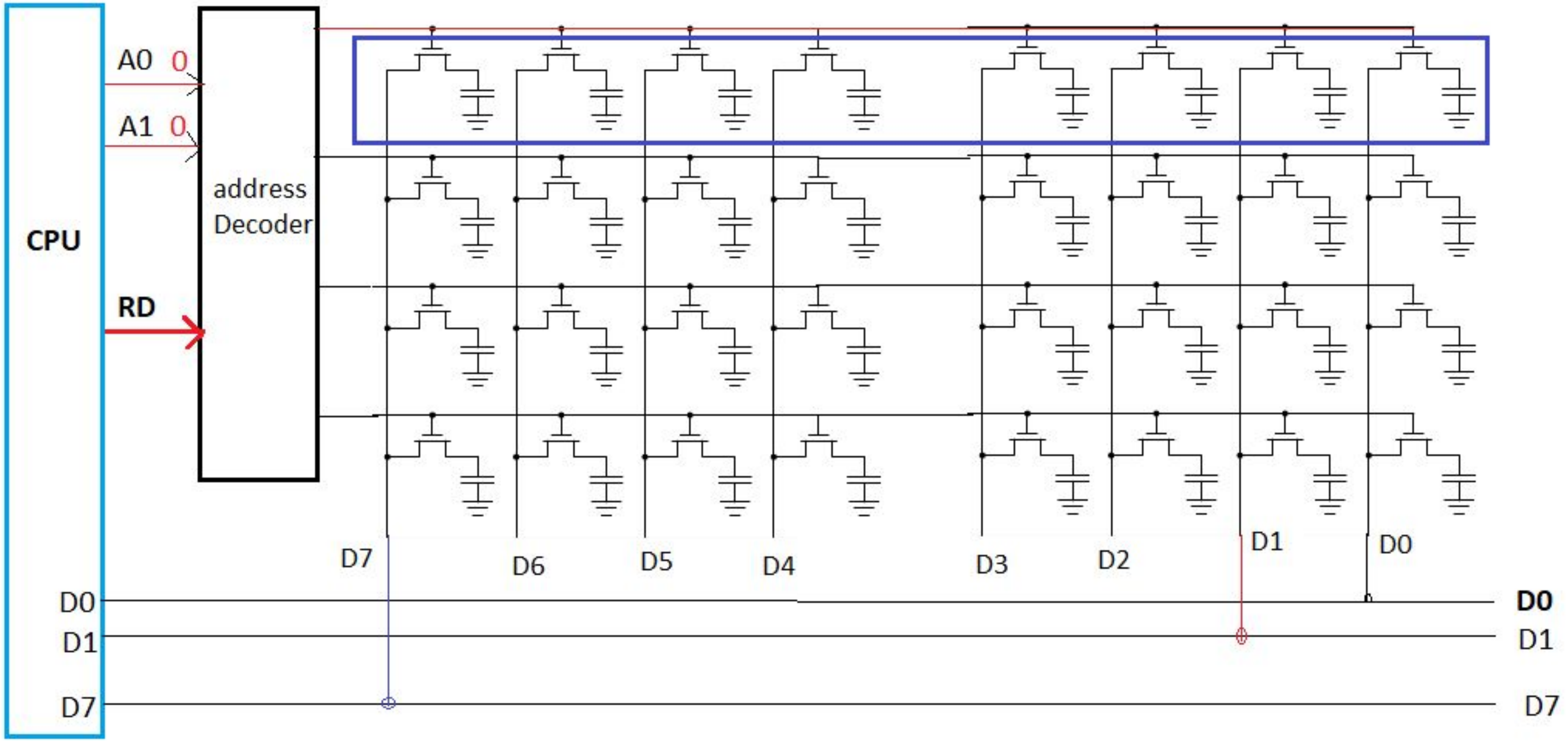
Interfacing CPU-RAM



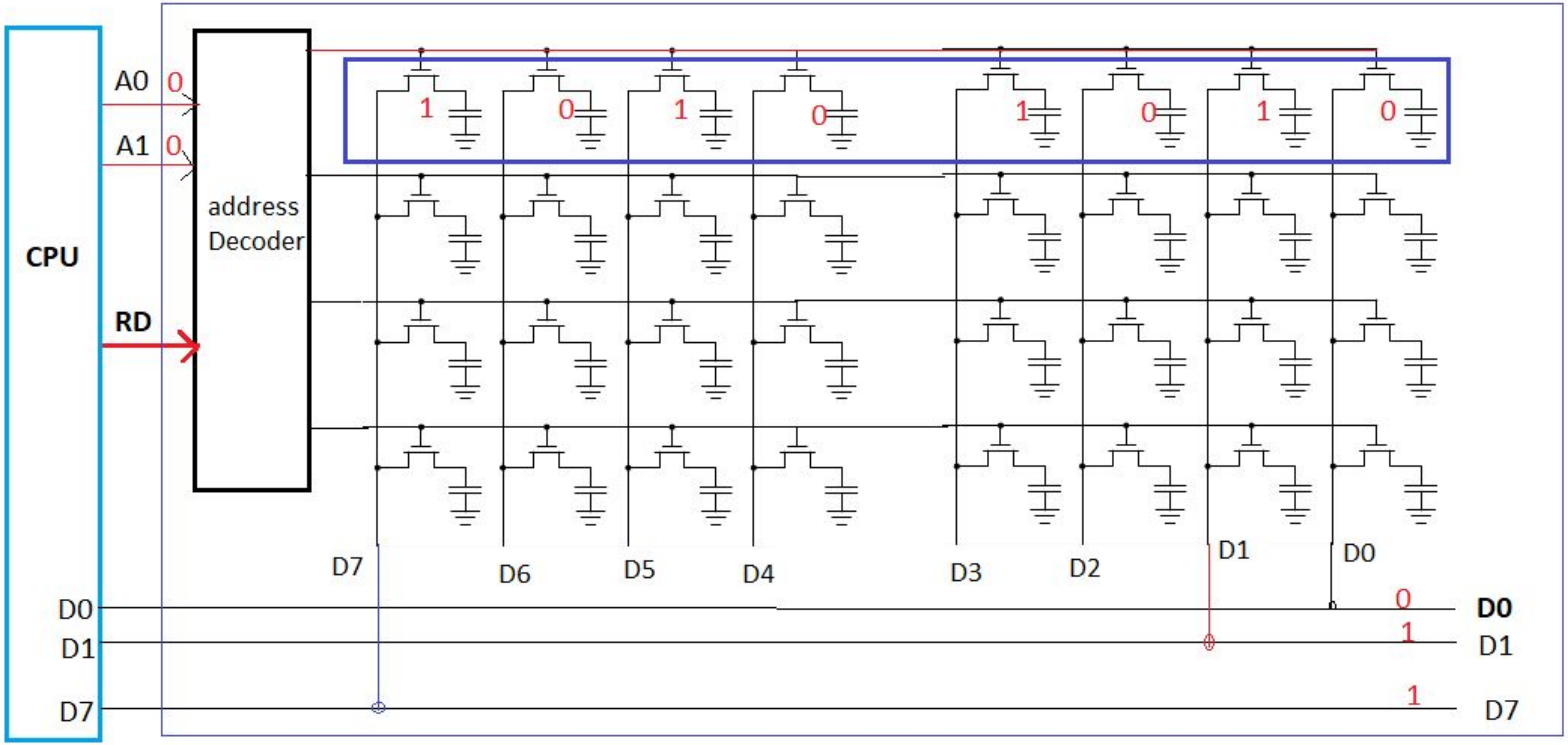
Interfacing CPU-RAM

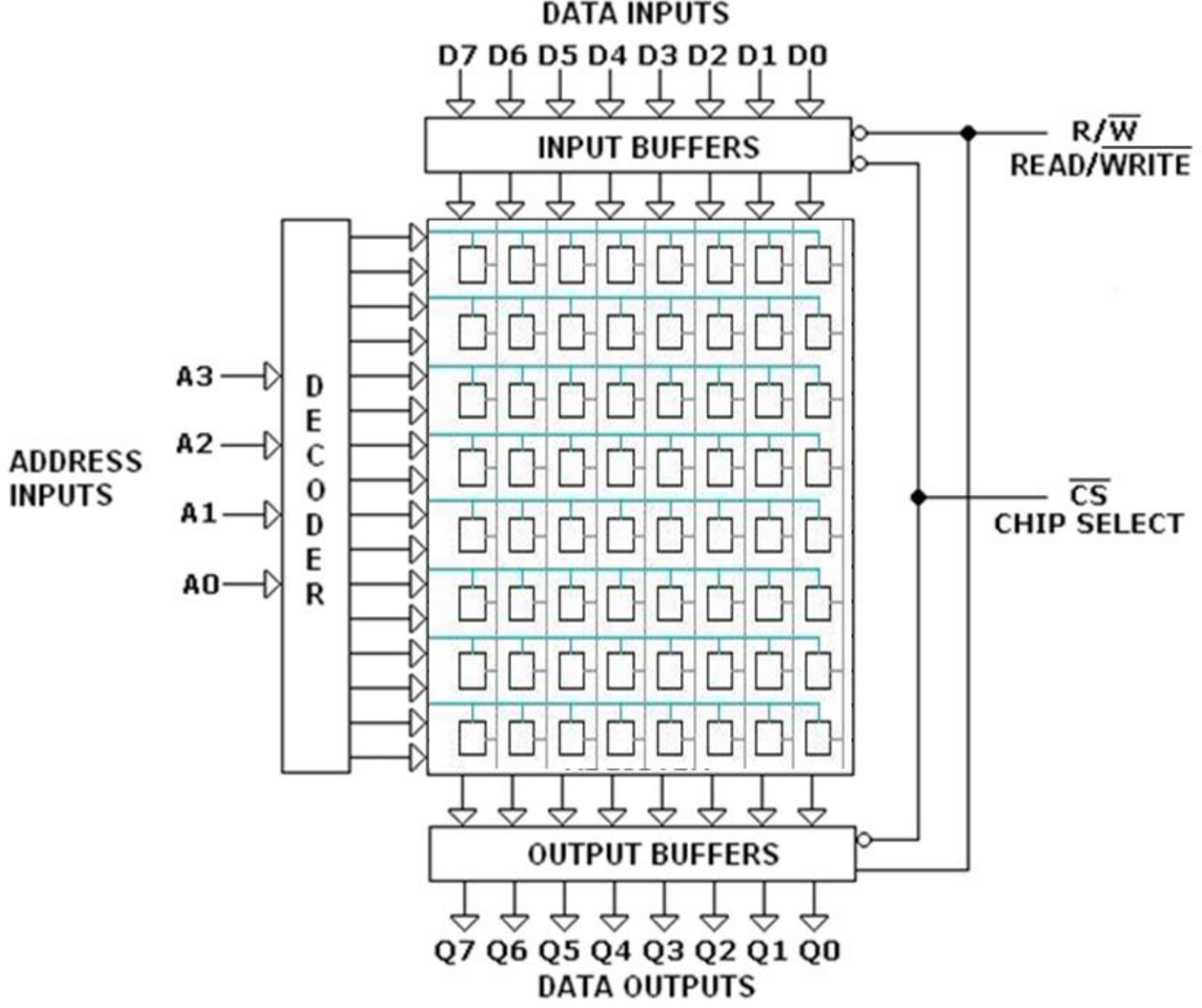


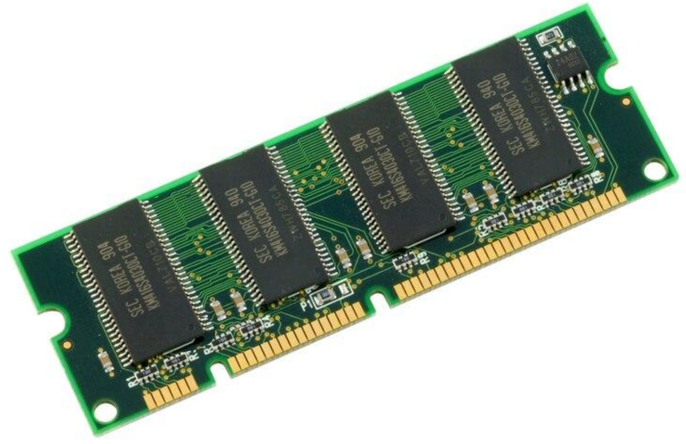
Interfacing CPU-RAM



Interfacing CPU-RAM




DATA INPUTS
D7 D6 D5 D4 D3 D2 D1 D0
INPUT BUFFERS
R/$\overline{W}$
READ/$\overline{WRITE}$
ADDRESS INPUTS
A3
A2
A1
A0
DECODER
$\overline{CS}$
CHIP SELECT
OUTPUT BUFFERS
Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
DATA OUTPUTS

M5M44256BP
723QR01 -8


Overview of Memory Organization

- Memory is one of the most important sub-systems of a computer that determines the overall performance.
- Conceptual view of memory:
  - Array of storage locations, with each storage location having a unique address.
  - Each storage location can hold a fixed amount of information (multiple of bits, which is the basic unit of data storage).
- A memory system with *M* locations and *N* bits per location, is referred to as an *M x N* memory.
  - Both *M* and *N* are typically some powers of 2.
  - Example: 1024 x 8, 65536 x 32, etc.

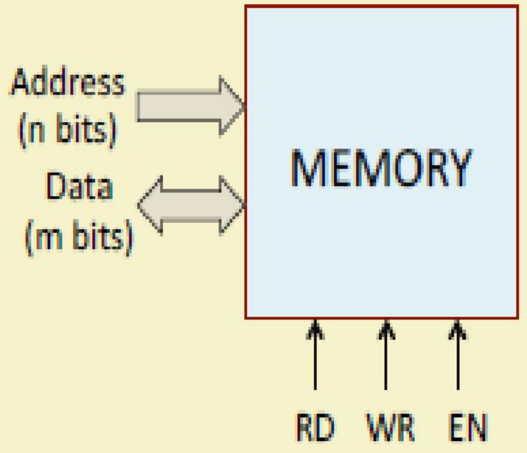
Some Terminologies

- Bit: A single binary digit (0 or 1).
- Nibble: Collection of 4 bits.
- Byte: Collection of 8 bits.
- Word: Does not have a unique definition.
  - Varies from one computer to another; typically 32 or 64 bits.

How do we Specify Memory Sizes?

Unit	Bytes	In Decimal
8 bits (B)	1 or 2^0	10^0
Kilobyte (KB)	1024 or 2^{10}	10^3
Megabyte (MB)	1,048,576 or 2^{20}	10^6
Gigabyte (GB)	1,073,741,824 or 2^{30}	10^9
Terabyte (TB)	1,099,511,627,776 or 2^{40}	10^{12}
Petabyte (PB)	2^{50}	10^{15}
Exabyte (EB)	2^{60}	10^{18}
Zettabyte (ZB)	2^{70}	10^{21}

- If there are *n* bits in the address, the maximum number of storage locations can be 2^n.
  - For n=8, 256 locations.
  - For n=16, 64K locations.
  - For n=20, 1M locations.
  - For n=32, 4G locations.
- Modern-day memory chips can store several Gigabits of data.
  - Dynamic RAM (DRAM).



Example: 1KB IC (D2708)

- 1KB: Approximately 1 Thousand (exact value 1024) locations each having capacity of 8 bits/1Byte
- Address starts at 0 and ends at 1 less than 1 Thousand, actually encoded in BINARY
- In Binary, first address requires 1 bit (0) and final addressable location requires 10 bits (all 1's: 11…11), since $\mathbf{2^{10} = 1K}$
- For ease of Decoder design, uniform address format is used for all the locations; Maximum number of bits!
- For convenience/ease of representation/programming/discussion, Hexadecimal number system is used to represent Memory address

Physical Address of Memory (for 1KB)

Address (10 bits in binary)	Content (8 bits)
1111111111B	11001100 (machine code/data)
…	
0000000000B	00110101(machine code/data)

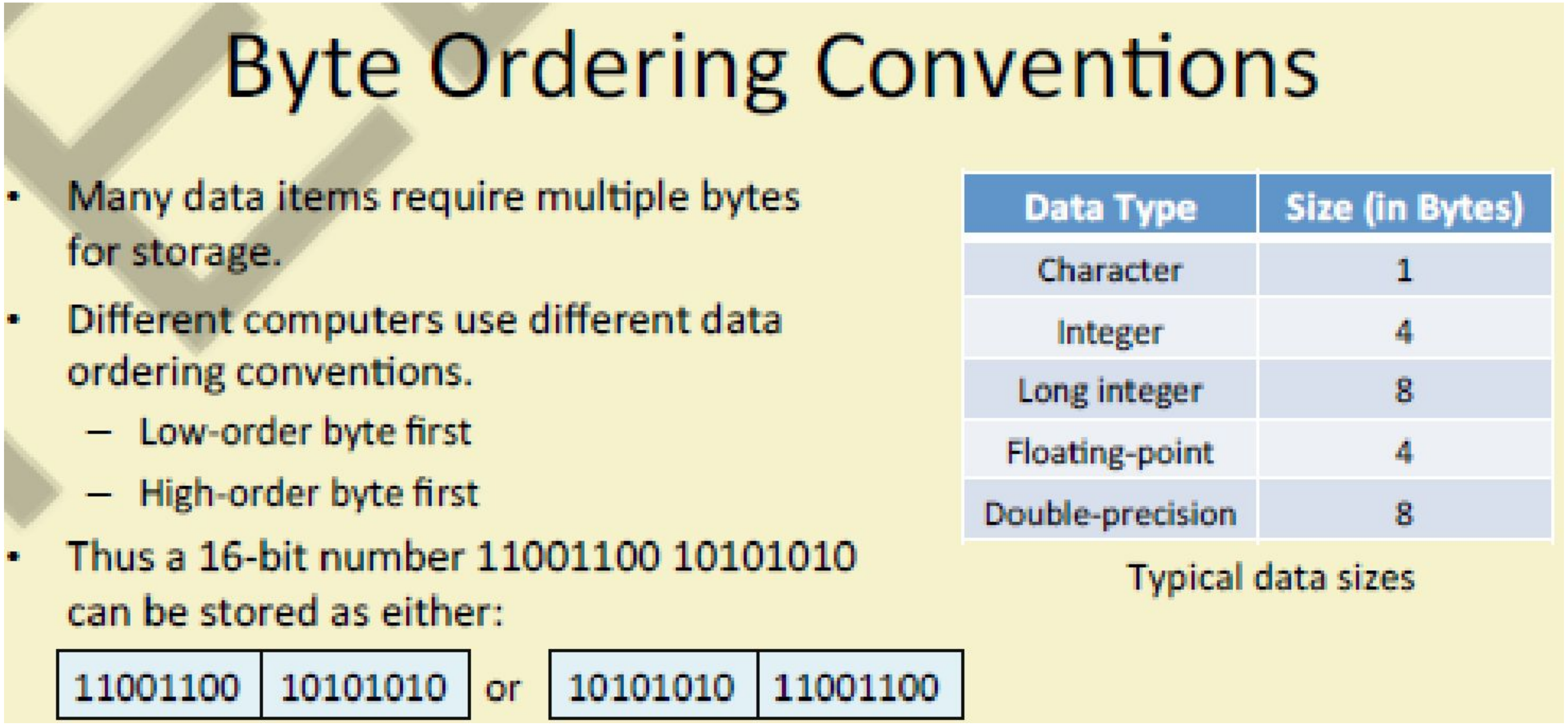
Address (3 digits in hexadecimal)	Content(8 bits)
3FFH	11001100 (machine code/data)
000H	00110101(machine code/data)

Capacity of Memory

- Example: 1MB: Approximately 1 Million (exact value 1024 x 1024) locations each having capacity of 1Byte
- Address starts at 0 and ends at 1 less than 1 Million, actually encoded in BINARY
- In Binary, first address requires 1 bit (0) and final addressable location requires 20 bits (all 1's: 11…11), since $\mathbf{2^{20} = 1M}$
- For ease of Decoder design, uniform address format is used for all the locations; Maximum number of bits!
- For convenience/ease of representation/programming/discussion, Hexadecimal number system is used to represent Memory address

Some Examples

1. A computer has 64 MB (megabytes) of byte-addressable memory. How many bits are needed in the memory address?
   - Address Space = 64 MB = 2^6 X 2^{20} B = 2^{26} B
   - If the memory is byte addressable, we need 26 bits of address.
2. A computer has 1 GB of memory. Each word in this computer is 32 bits. How many bits are needed to address any single word in memory?
   - Address Space = 1 GB = 2^{30} B
   - 1 word = 32 bits = 4 B
   - We have 2^{30} / 4 = 2^{28} words
   - Thus, we require 28 bits to address each word.



Byte Ordering Conventions

- Many data items require multiple bytes for storage.
- Different computers use different data ordering conventions.
  - Low-order byte first
  - High-order byte first
- Thus a 16-bit number 11001100 10101010 can be stored as either:

11001100	10101010	or	10101010	11001100

Data Type	Size (in Bytes)
Character	1
Integer	4
Long integer	8
Floating-point	4
Double-precision	8

Typical data sizes

Byte Ordering

- The ordering of bytes within a **multi-byte** data item defines the endian-ness of the architecture.
- In BIG-ENDIAN systems the most significant byte of a multi-byte data item always has the lowest address, while the least significant byte has the highest address.

  In LITTLE-ENDIAN systems, the least significant byte of a multi-byte data item always has the lowest address, while the most significant byte has the highest address.
- In the following example, table cells represent bytes, and the cell numbers indicate the address of that byte in main memory. Note: by convention we draw the bytes within a memory word left-to-right for big-endian systems, and right-to-left for little-endian systems.

- The two conventions have been named as:

  a) Little Endian

  - The least significant byte is stored at lower address followed by the most significant byte. Examples: Intel processors, DEC alpha, etc.
  - Same concept followed for arbitrary multi-byte data.

  b) Big Endian

  - The most significant byte is stored at lower address followed by the least significant byte. Examples: IBM's 370 mainframes, Motorola microprocessors, TCP/IP, etc.
  - Same concept followed for arbitrary multi-byte data.

An Example

- Represent the following 32-bit number in both Little-Endian and Big-Endian in memory from address 2000 onwards:

  01010101 00110011 00001111 11000011

Little Endian	
Address	Data
2000	11000011
2001	00001111
2002	00110011
2003	01010101

Big Endian	
Address	Data
2000	01010101
2001	00110011
2002	00001111
2003	11000011

Expanding word size and capacity