INTERFACES

Prepared by Dr Syed Khaldoon Khurshid
Powered by Brilliant Application

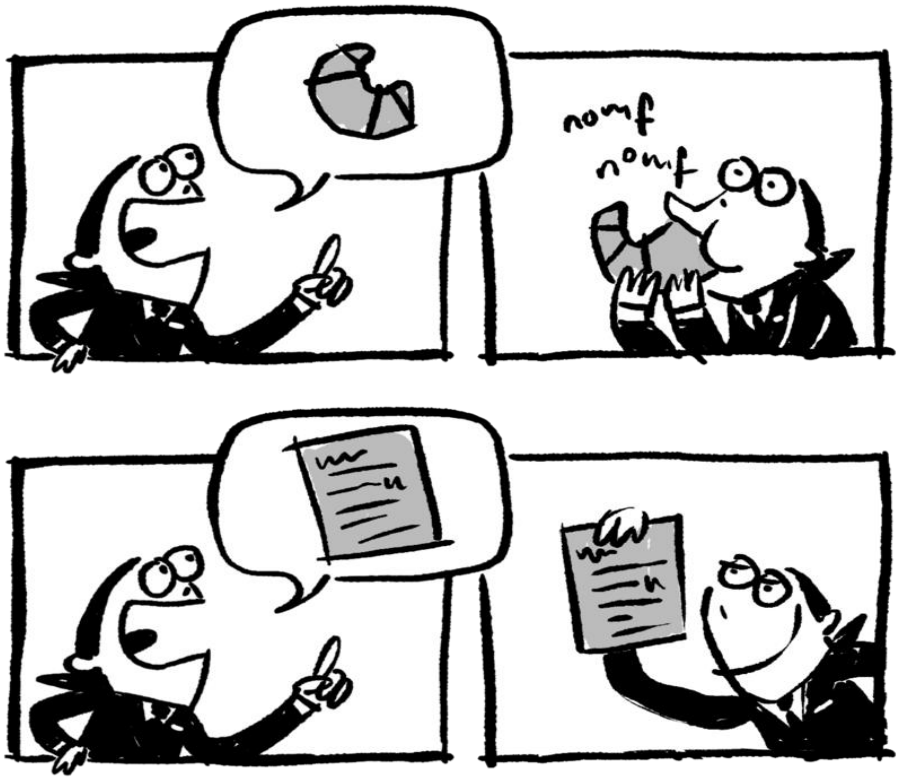
- Mayor Jing is in her office, and she remembers she needs to review all the **Fire Department memos for a presentation to City Council**. She could go and find all the memos herself, but Mayor Jing isn't a micromanager. She has **grouped several employees** as members of the Records Office—an abstraction—and she can ask the Records Office to do things.
- She picks up her phone, hits the button for her Records Office, and says "**get me all the Fire Department memos so I can read them over lunch**."
- Later, when she leaves for lunch, all the memos were in her office.


JING

- Mayor Jing has just used another form of **abstraction**. There was a problem: she needed all the Fire Department memos and she did not have them. **She was able to get a solution to her problem, and she didn't need to know how that solution works!**

- Mayor Jing knows that she can ask her Records Office to **do a certain set of things**: for example, she can ask them to search for a group of records on a particular topic or all the records from a certain date. But she'd never ask the Records Office to fight a fire! The things that Mayor Jing can ask form **a sort of *menu*. When you have an abstraction, the menu tells you what you can ask for and gives you some idea of what you will get back.**
- The menu of the Records Office isn't so different from the **menu at the restaurant** Mayor Jing eats lunch at. There are a limited set of requests you can make, and when Mayor Jing makes one of those requests, she expects to get something relatively specific back.

- The burrito restaurant and the Records Office are two kinds of abstractions. **In computer science, we call the menu that an abstraction offers its *interface***.
- In computer science, **the interface is often called an API**. That officially stands for ***Application Programming Interface***. However, the only important letter to remember is the "I," standing for "interface."



- **One of the important properties of an interface is that it's possible to use it *without knowing how it works***. Mayor Jing may not know how to cook the thing she is eating for lunch, but she's still able to order it!
- It is the case that *somebody* needs to know how to prepare Mayor Jing's restaurant order. **Someone also needs to know how to get all the Fire Department memos**. But this is a (potentially complicated) detail that gets hidden from Mayor Jing, allowing her to think about other things. This is one of the benefits of abstraction!
- **It is also important for computer scientists that when there is an interface, there may be *more than one implementation***.
- Let's look back at Mayor Jing's request for all the Fire Department memos, and think about how that request could be implemented.

- Maybe Mayor Jing was employing the **extremely patient**, Farhad, who fulfilled the request by looking through every memo one by one to collect all the ones whose topic was the Fire Department.
- What's the **biggest downside of this approach**?

- This approach is slow.
- This approach requires hiring lots of people.
- This approach takes a lot of space.

Correct answer: This approach is slow. Farhad's method will get the job done, but it takes a lot of time!

- **As an alternative**, maybe Tiye the librarian has taken a leave from her library job to work for Mayor Jing.
- Inspired by her library's old **card catalog**, Tiye has filled many rooms with **copies of every memo stored in every conceivable order:** one copy of all memos ordered by **date**, one ordered by **author**, one ordered by **recipient**, one ordered by **title**, and one ordered by **topic**.

- This means she can go to the **topic-ordered file** and very quickly find all the memos about the Fire Department.
- What's the biggest downside of this approach to the problem?

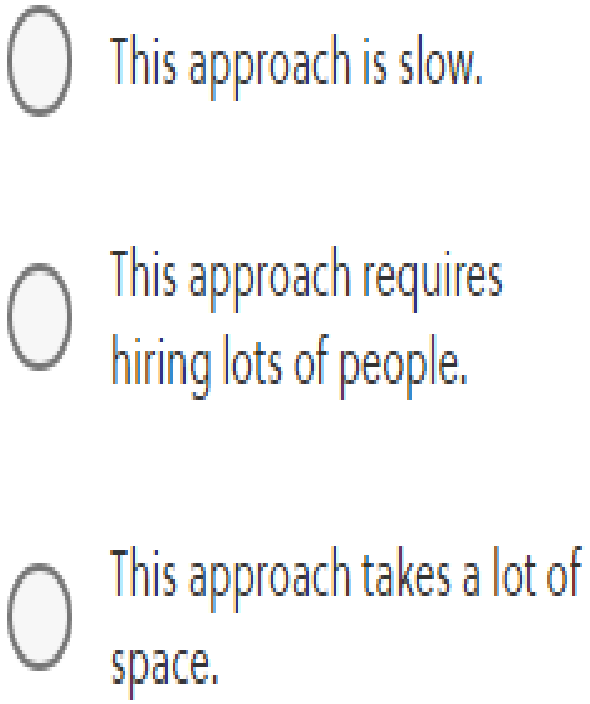
- ○ This approach is slow.
- ○ This approach requires hiring lots of people.
- ○ This approach takes a lot of space.

Correct answer: **This approach takes a lot of space.**

- Tiye's approach works, but it will fill a lot of space with filing cabinets!
- **It would be more reasonable if Tiye tried a more modern approach of storing the separate indexes in a computer database**. But remember that, in computer science, "**space**" is used to describe the use of **computer storage**. Storing those extra copies of information in another order to help with faster search uses lots of computer storage, even if it wouldn't take nearly as much physical space.

- What if the Mayor's Office has gotten help by hiring Pierre and his **embarrassingly parallel** baking assistants? **They can easily divide all the memos among them and find the ones that are about the Fire Department.**

- **What's the biggest downside of this approach to the problem?**



Correct answer: This approach requires hiring lots of people. Pierre and his assistants can get the job done quickly, but there are lots of them!

- We've seen **three very different ways of implementing** Mayor Jing's request for all the Fire Department memos. **But from Mayor Jing's perspective, all of these had the same *interface*: she asked the Records Office for all the Fire Department memos, and she got all the memos back.**
- This will probably work fine for Mayor Jing… **until it stops being fine**.

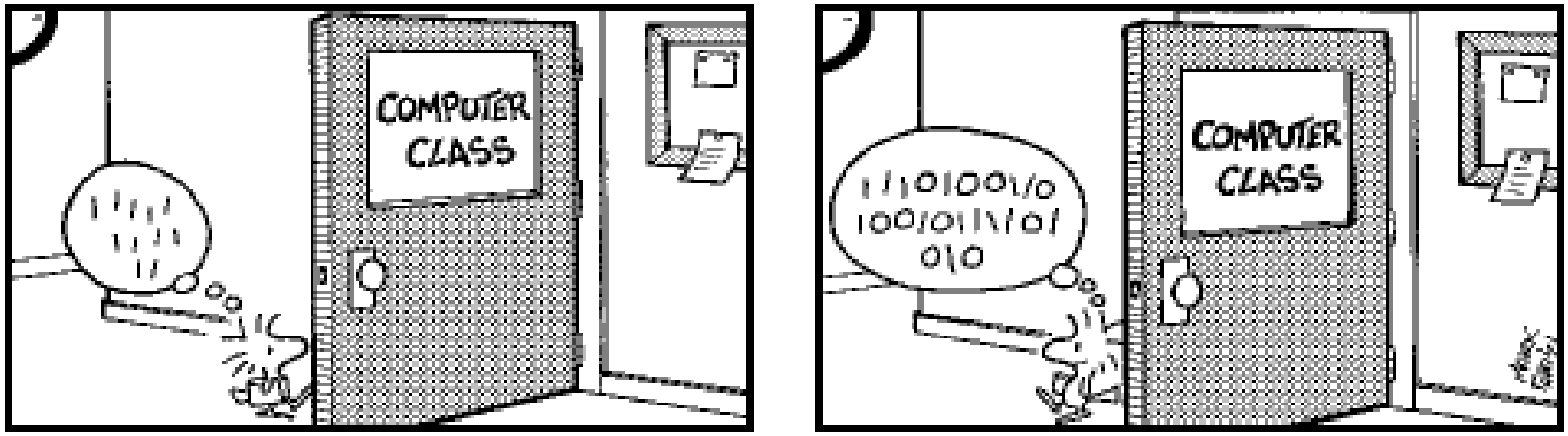
Maybe the department's personnel costs are too high, and Hans needs to downsize to a less embarrassing number of assistants.

- Maybe the City Council is angry that their chambers are full of Tiye's filing cabinets, and she needs to back off a bit.
- Maybe Mayor Jing isn't able to get answers to her questions in a timely fashion, and Farhad needs to get more organized.
- Figuring out the right solution always requires Mayor Jing to ***look through the interface*** **for a bit to fix the problem**. But once she's fixed the problem in a way that meets her priorities and needs, she can go back to treating her Records Office as an abstraction.

- In this example, we've seen how all of our computational problem-solving strategies can approach a simple problem: finding a small set of Mayor Jing's Fire Department memos.
- **In computer science, we're frequently using *abstraction* to allow ourselves to focus on a small part of a very complex system. Once we've concentrated on a small part of the big system, we can think about using other problem-solving strategies to attack the more manageable problem.**

CHAPTER 1

Data Storage (& Representation)



Prepared by Dr Syed Khaldoon Khurshid

1.1 Bits and Their Storage

- Information represented as patterns of *bits* (binary digits)
- **A bit is either 0 or 1 (*true* or *false*)**
- **Meaning of bit(-stream)s varies**
  - **numeric values, characters, images, sounds…**
- **Requires a device that can be in one of two states (& remain in that state as long as needed)**
  - ***Flip-flop* circuits**

1.1 The Boolean Operations AND, OR, and XOR

The AND operation

```
  0     0     1     1
AND 0 AND 1 AND 0 AND 1
  0     0     0     1
```

The OR operation

```
  0     0     1     1
OR  0 OR  1 OR  0 OR  1
  0     1     1     1
```

The XOR operation

```
  0     0     1     1
XOR 0 XOR 1 XOR 0 XOR 1
  0     1     1     0
```

- Note: AND and OR exist in **natural language**!

Example:
Boolean Operations and Natural Language

1. Khalid has gone to cafe.
2. **AND** Umer has gone to School.
3. **THEN** their mother can do house chores.

Result:

- Both has to go to school for their mother to do chores of the house.
- Check with OR operation.

1.1 AND and OR Gates

- **‘0’ and ‘1’ Digits are representing “Voltage levels” ?**

AND

Inputs	Output
0 0	0
0 1	0
1 0	0
1 1	1

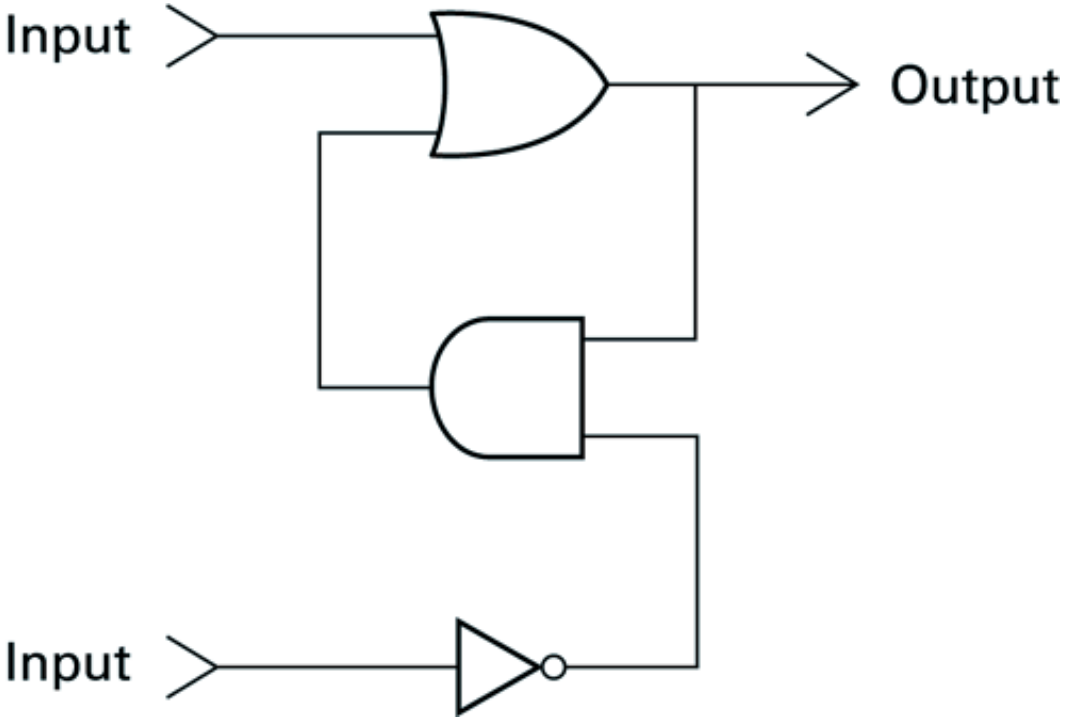
OR

Inputs	Output
0 0	0
0 1	1
1 0	1
1 1	1

Flip Flop circuit:

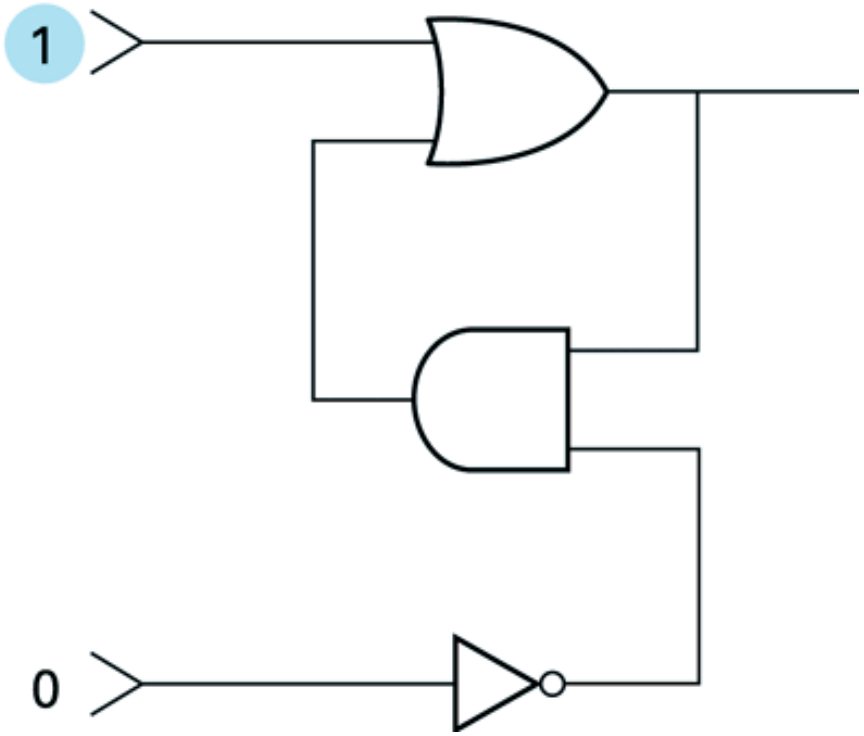
- Flip-flops and latches are **fundamental building blocks of digital electronics systems** used in computers, communications, and many other types of systems.
- **A flip flop circuit that produces an output value of ‘0’ or ‘1’ that remains constant until a temporary pulse from another circuit causes it to shift to the other value.**

1.1 A Simple Flip-flop Circuit

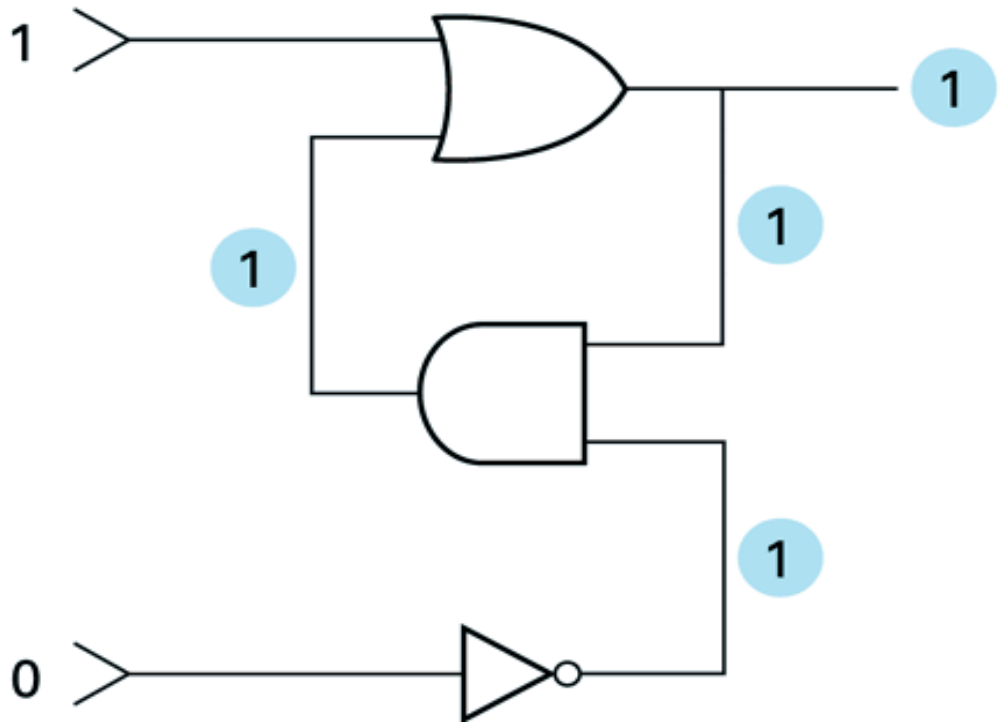


- As long as both inputs remain 0: output does not change
- Temporarily placing **1** on upper input => output = **1**
- Temporarily placing **1** on lower input => output = **0**
- So: output flip-flops between 2 values under external control

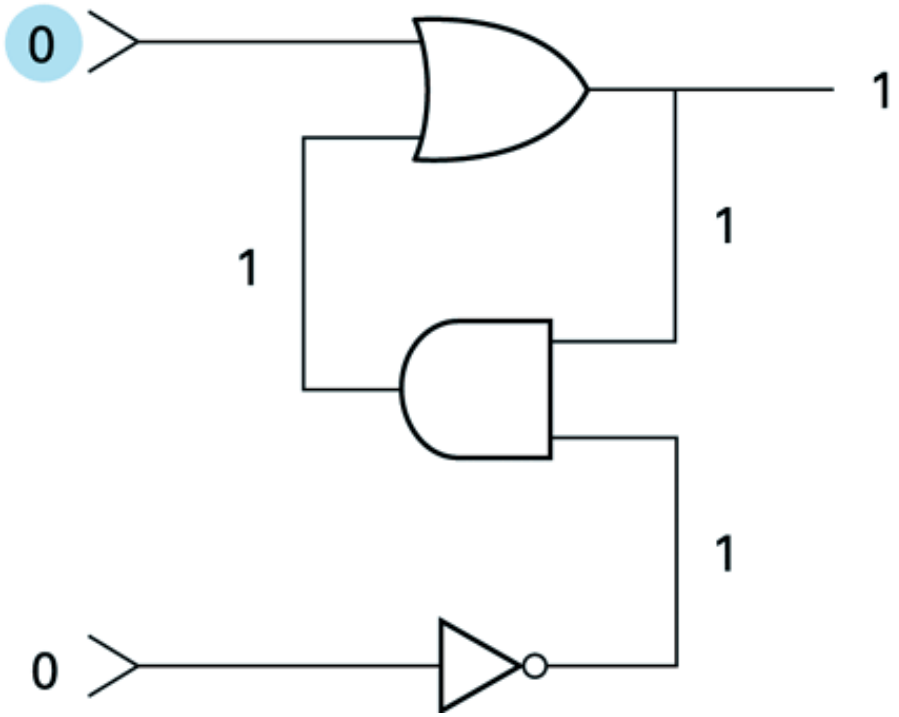
1.1 Setting the Output of a Flip-flop to 1



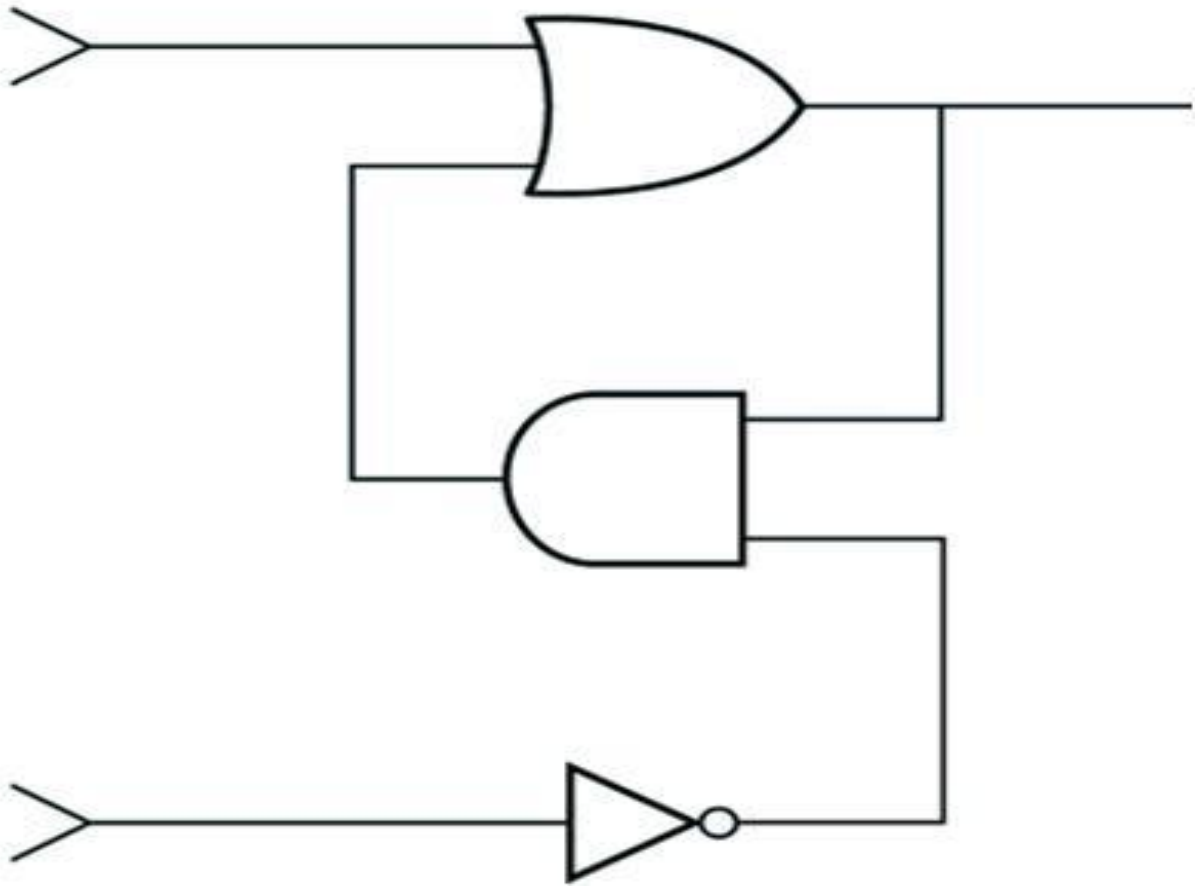
1.1 Setting the Output of a Flip-flop to 1 (cont'd)



1.1 Setting the Output of a Flip-flop to 1 (cont'd)



Temporarily placing ‘1’ on the lower end of the flip flop (retaining previous output):



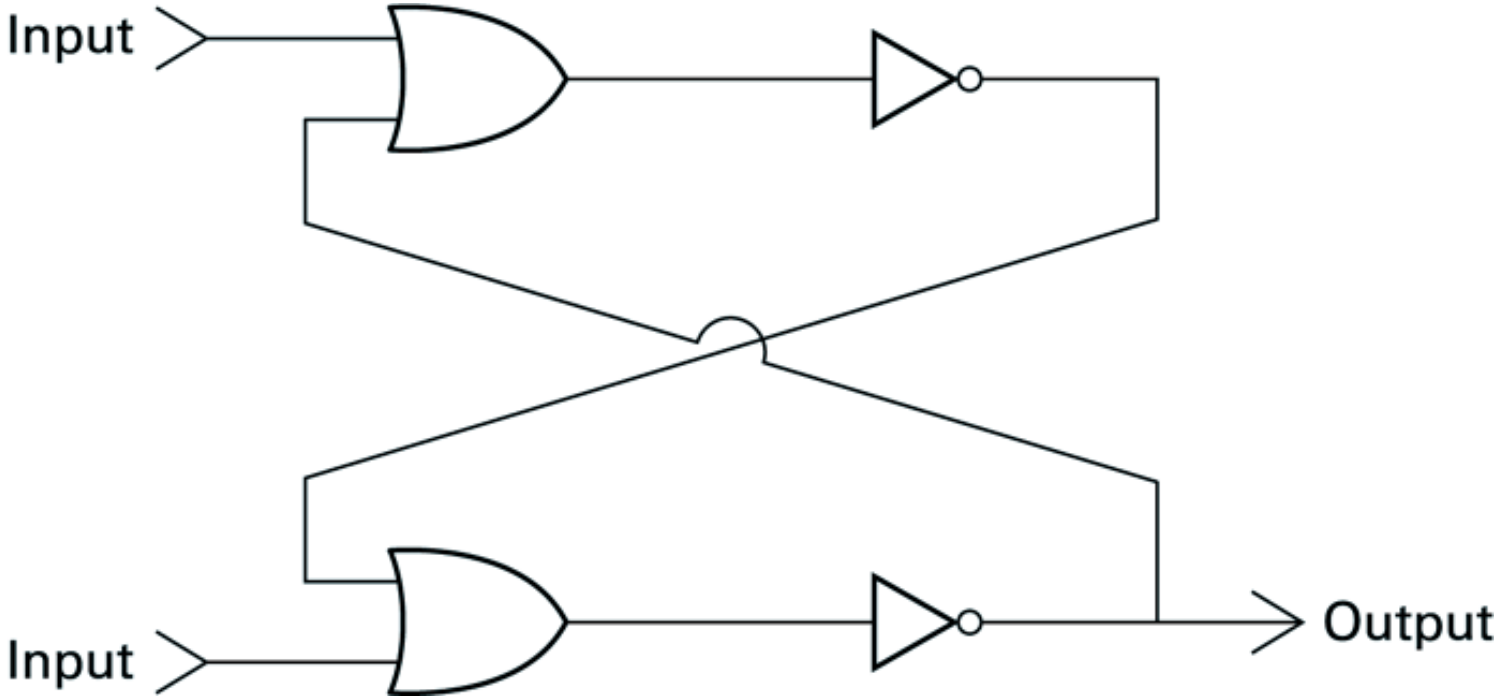
Another way of constructing Flip-Flop:

SR latch (Set-Reset) Latch

Write outputs & sequence of steps on the basis of following inputs?

- **upper input = 1 and lower input = 0**
- **upper input = 0 and lower input = 0**

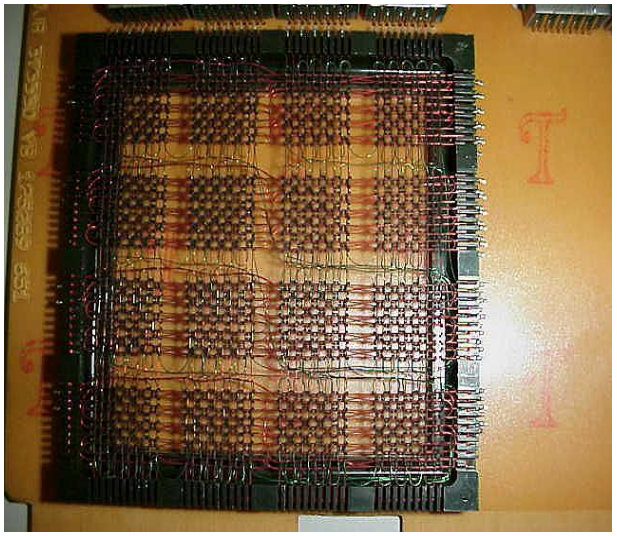
Solve it...its solution in The next class.



Other Storage Techniques

- **Cores**

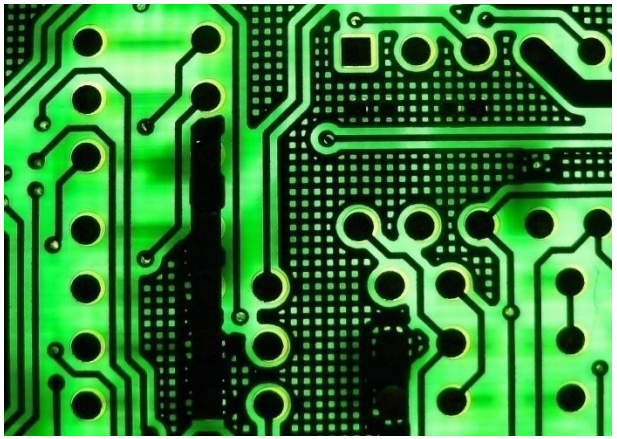
- Magnetic-core memory was the predominant form of random-access computer memory for 20 years between about 1955 and 1975. Such memory is often just called core memory, or, informally, core.
- Core will be Magnetized in one of the two directions
- Retain Data after Machine is switched off
- Obsolete



Core Memory from an IBM 2821

- **Capacitors**->Millions-> Chip

- A **capacitor** can store electric energy when it is connected to its charging circuit. And when it is disconnected from its charging circuit, it can dissipate that stored energy, so it can be used like a temporary battery. ... In car audio systems, large **capacitors** store energy for the amplifier to **use** on demand.
  - Charge or Discharge Plates
  - Charges on capacitors dissipate
  - Refresh Circuit

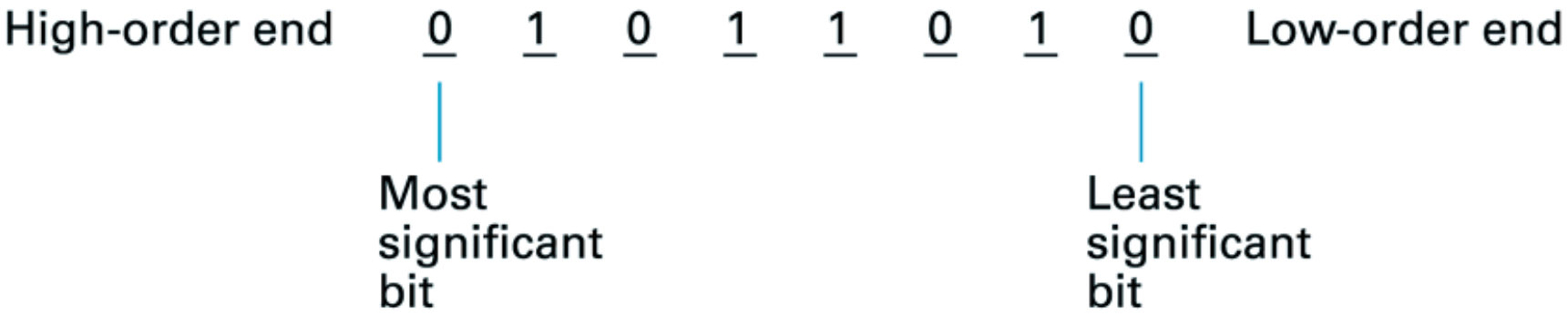


Circuit Board with capacitors

- **Dynamic Memory**

1.2 Main Memory

- Large collection of circuits, each capable of storing a single bit
- Arranged in small cells, typically of 8 bits each (a.k.a.: *byte*)



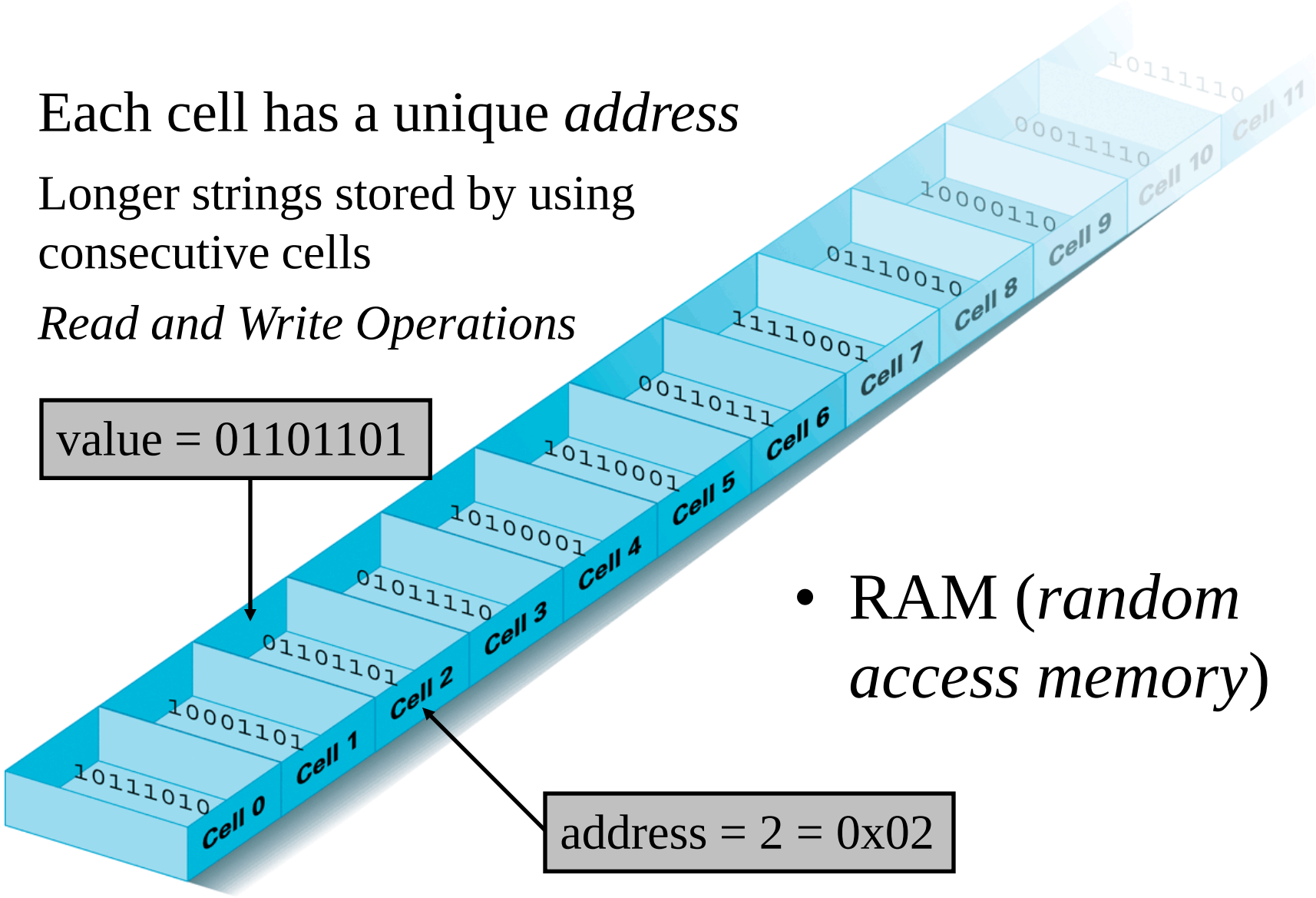
1.1 The Hexadecimal Coding System

- Bit-streams often very long
- For simplicity of notation:
  - Hexadecimal system
- Reduces 4 bits to 1 symbol
- Especially important in assembly language programming.

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

1.2 Arrangement of Memory Cells

- Each cell has a unique *address*
- Longer strings stored by using consecutive cells
- *Read and Write Operations*



- RAM (*random access memory*)

Chapter 1: Problem 6

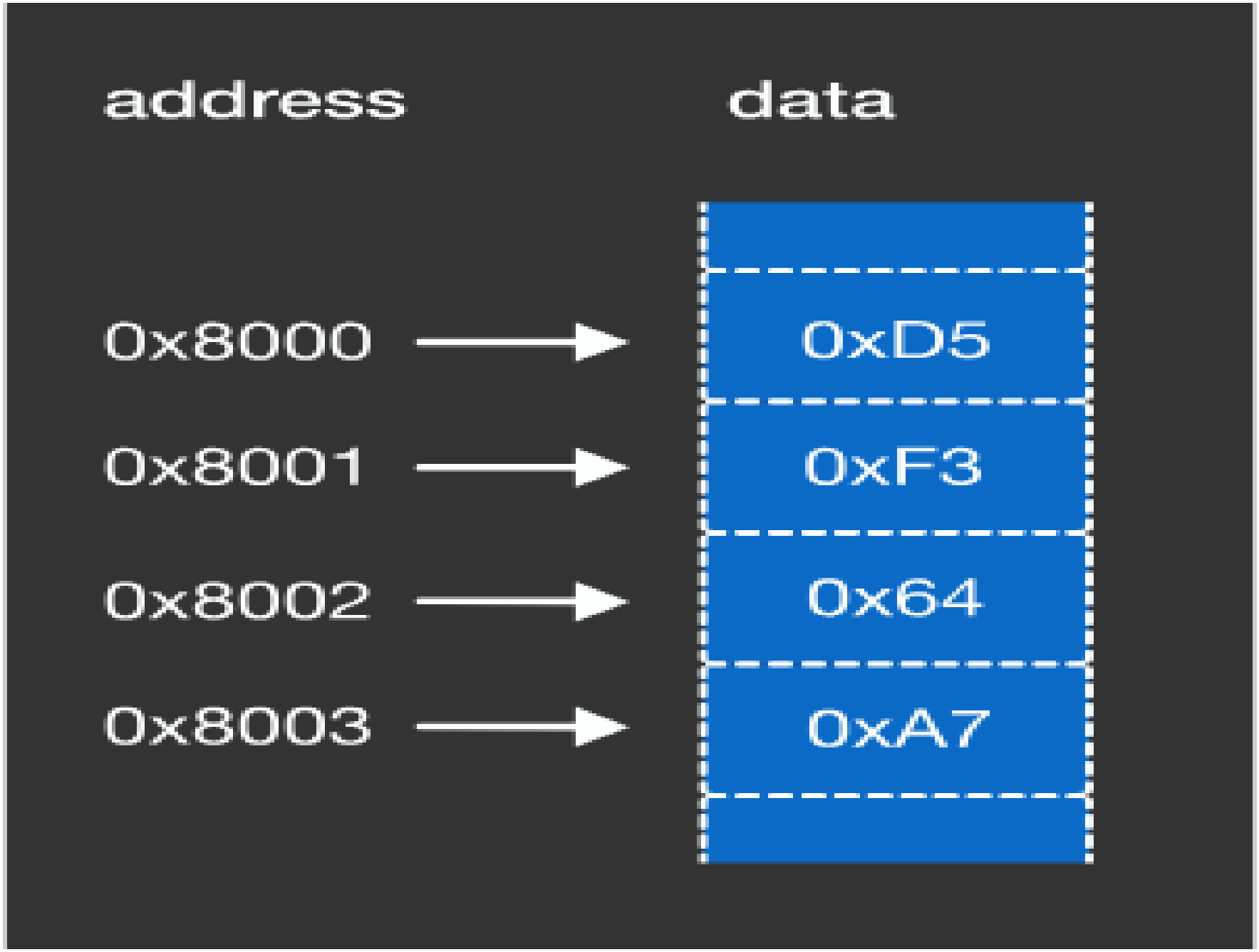
How many cells can be in a computer's main memory if each cell's address can be represented by 3 hexadecimal digits?

- Three digits:
  - 3 positions, each of which can be one of 16 values (from the range: 0, 1, …, 9, A, B, C, D, E, F)
  - smallest: $000 = 0\times16^2 + 0\times16^1 + 0\times16^0 = 0$
  - largest: $FFF = 15\times16^2 + 15\times16^1 + 15\times16^0 = 4095$
  - So, total number of unique addresses = $16^3 = 4096$

Introduction to Memory

@ brilliant Application

- From the point of view of a running program, memory **can be understood as a long tape of bytes**, and the program can read from or write to certain places on the tape. There are two concepts that we need to know:
- **address, an integer that specifies a place on the tape**
- **data, the byte stored in each place on the tape**


address
data
0x8000
0xD5
0x8001
0xF3
0x8002
0x64
0x8003
0xA7

- We often use hexadecimal for data. The beginning address of the tape (or memory) is determined by the OS. **For this example, the byte at the address 0x8000 contains the data 0xD5.**
- **In modern computer architectures, one memory address points to one byte.** Therefore, the address of the next byte is 0x8001, which contains the data 0xF3.
- Programs access memory by running CPU instructions and specifying the target address. **For example:** read 1 byte from address 0x8000 write 2 bytes to address 0x8002 (this means 0x8002 and 0x8003).

- When working with memory, we often use sizes that are $\mathbf{2^N}$ for convention.
- We use units like:

  MB

  (2^{20} =1024×1024 bytes) or

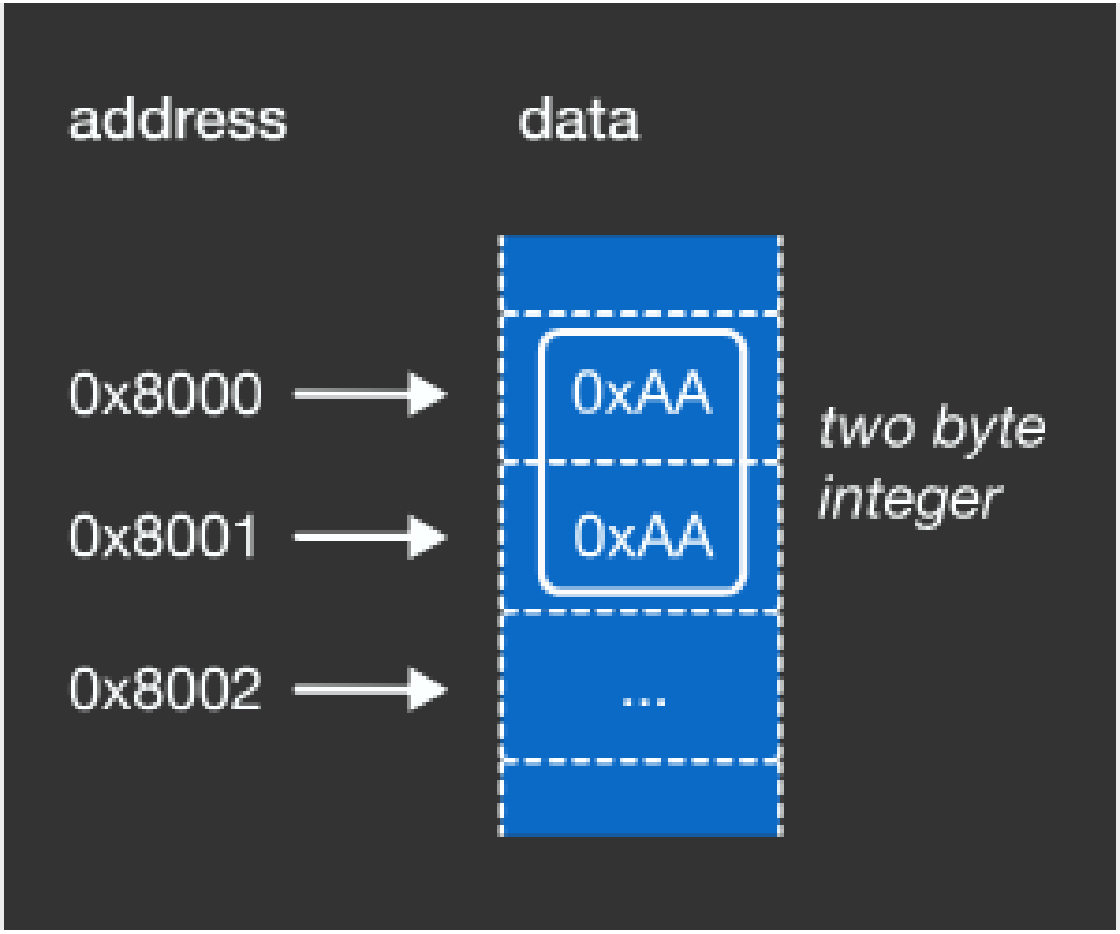
  GB

  (2^{30} =1024×1024×1024 bytes),

  instead of the regular units like

  MB (10^6 bytes) or GB (10^9 bytes).

- Since **one byte can hold values no larger than 255**, we often work with integers that span multiple bytes to hold larger numbers.
- **For example**, we need at least 2 bytes of memory to represent the number **43690**, which is 0xAAAA in hexadecimal.



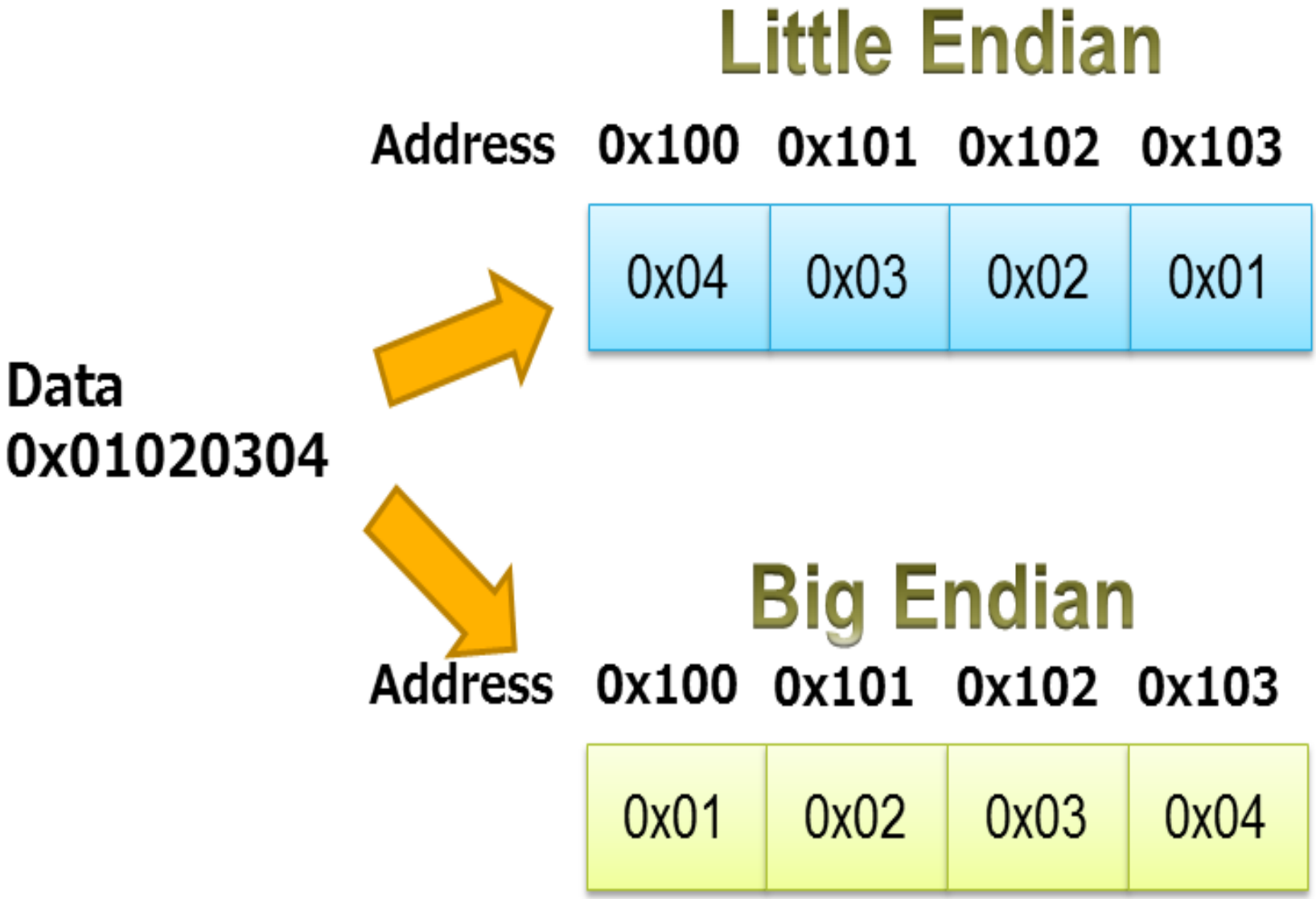
At least how many bytes of memory do we need to represent the decimal number **1,000,000,000**?

Explanation

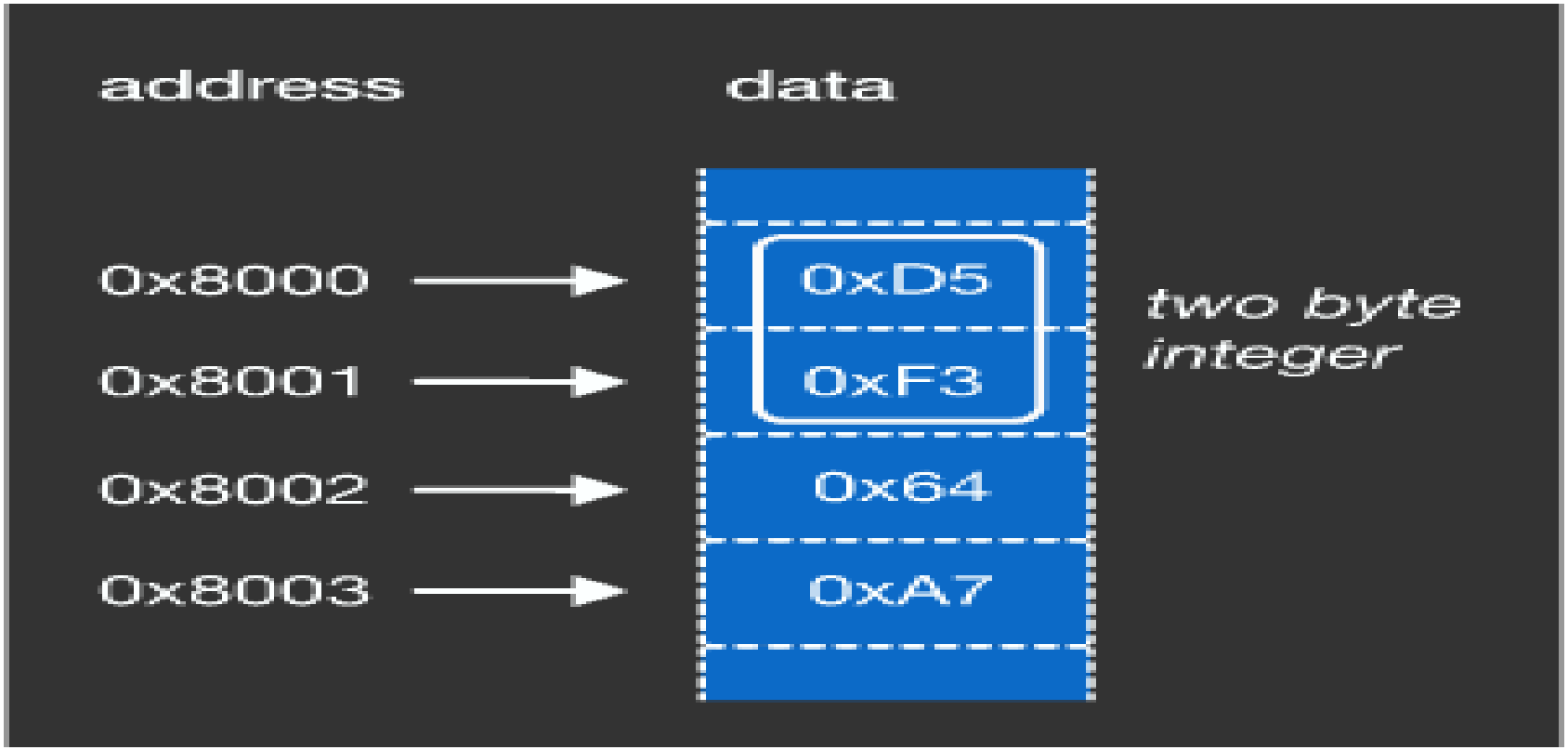
- **N bytes can represent values up to $2^{8N}-1$.** Thus,
- 1 byte can represent values up to 255.
- 2 bytes can represent values up to 65,535.
- 3 bytes can represent values up to 16,777,215.
- **4 bytes can represent values up to 4,294,967,295**.
- Since 3 bytes are not enough but 4 bytes are, we need at least 4 bytes.
- **Correct Answer:4 Bytes**

Endianness:

- Since each byte has a memory address, it's worth asking: **in what order do we store the bytes that compose a multi-byte integer**?
- Do we store the least-significant (rightmost) byte in the first memory address, or do we store the most-significant (leftmost) byte in the first memory address?
- The answer is that it depends on the **endianness** of your computer's architecture.
- Storing the most-significant byte first is called **big-endian**. Storing the least-significant byte first is called **little-endian.**



https://www.iar.com/support/resources/articles/migration-techniques-for-different-endianness/



Given a **little-endian computer architecture**, and memory that looks like the above diagram, the address **0x8000 and 0x8001** together are used to describe a 2-byte integer. What is the integer in decimal (base 10)?

Correct answer: **62421**

Explanation

- The address 0x8000 contains 0xD5, and the address 0x8001 contains 0xF3.
- Since the computer architecture is **little-endian**, the two-byte integer is **0xF3D5**, which is 62421 in decimal.

- The entire memory can be considered as a giant 1-dimensional tape of bytes. However, to make it easy to work with, we'll often **wrap it across multiple lines such that there are many bytes per line (often 16),** and the memory address of the first byte on each line will be specified in a column to the left of the resulting table. To get the memory address of an arbitrary byte, look up the memory address of the first byte on that line, count the number of bytes to the left of the target byte, and add that number to the memory address of the first byte.
- This method of displaying memory is called a **hex dump** and is quite common in debuggers, packet sniffers and hex editors etc.

```
0xff340: 80 f3 bf 5f ff 7f 00 00 de d1 0b 00 01 00 00 00
0xff350: 00 e0 07 00 01 00 00 00 00 00 00 00 00 00 00 00
0xff360: 00 00 00 00 00 00 00 00 aa 94 1d 00 00 00 00 00
0xff370: 00 00 00 00 00 00 00 00 e0 b0 0b 00 01 00 00 00
0xff380: a0 f3 bf 5f ff 7f 00 00 48 d2 0b 00 01 00 00 00
0xff390: c0 b1 0b 00 01 00 00 00 c0 b1 0b 00 01 00 00 00
0xff3a0: 50 f4 bf 5f ff 7f 00 00 80 0f 00 00 01 00 00 00
0xff3b0: 0f 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
0xff3c0: 68 00 00 00 01 00 00 00 5f 7c c2 5f ff 7f 00 00
```

What is the memory address of the **red bolded** byte?

Correct answer: **0xff389**

Explanation

- The beginning address of the row is **0xff380**.
- The offset of the bolded byte is **9**, since the position of a byte in a row is zero-indexed.
- Therefore, the address of the target byte is **0xff380 + 9 = 0xff389**.

The x86-64 architecture is **little-endian**. Below is a memory dump from a computer running on this architecture:

```
0xa72de0: 4f 01 64 05 7a e9 ee 84 99 f6 bb e2 d5 a7 fe e1
0xa72df0: bf 36 94 97 d5 f3 64 a7 08 ff 82 01 76 b4 1f 5b
0xa72e00: 18 2a bb 88 a0 d3 7b db ab 18 45 db a0 cc 63 e7
0xa72e10: 8e b0 60 6b c0 98 a9 a8 12 82 9a 82 79 ed 87 45
0xa72e20: ac 67 36 ef 46 d9 6b c1 79 01 bd 2d 0d c6 29 dd
0xa72e30: 8f ee 3e 9a b4 b3 de 5e 17 cb 74 86 71 10 49 f2
0xa72e40: 29 1f 72 0b f1 f2 dd e7 e9 1a a4 f8 98 c7 24 e9
0xa72e50: db 81 12 8e d1 9d f7 9f a8 25 40 10 62 57 f2 c7
```

What is the value of the 32-bit integer stored at memory address 0xa72e29?

Answer in decimal (base 10).

Correct answer: **221101313**

Explanation

- A 32-bit integer takes four bytes. The four bytes starting at address 0xa72e29 are **01, bd, 2d, 0d**, each in hexadecimal.
- Due to the architecture being little-endian, the integer is read as 0x0d2dbd01.
- **0x0d2dbd01** is 221101313 in decimal.

MEMORY LAYOUT

Powered by Brilliant App

- we are going to see how variables in programs get placed in memory. We will use the programming language C for explanation. There are a few data types that represent integers in C, i.e. char, short, int, and long. **The size of each data type depends on the compiler and computer architecture**, but we will assume the following sizes here:
- **char … 1 byte**
  short … 2 bytes
  int … 4 bytes
  long … 8 bytes

- When programming, we need to decide on the size of the integer to use. We often use int, the default type, which has **4 bytes** and can hold integers in the range about **-2 billion to +2 billion**, and this is enough for most cases. We use the other data types for situations where we **want larger or smaller integers**. For example, we use long when we need a variable that can hold numbers that are larger than 2 billion. **We may use char or short when we want to save memory**.

Question

What is the total amount of memory, in bytes, needed when allocating the following four variables?

- **char a;**
- **short b;**
- **int c;**
- **long d;**

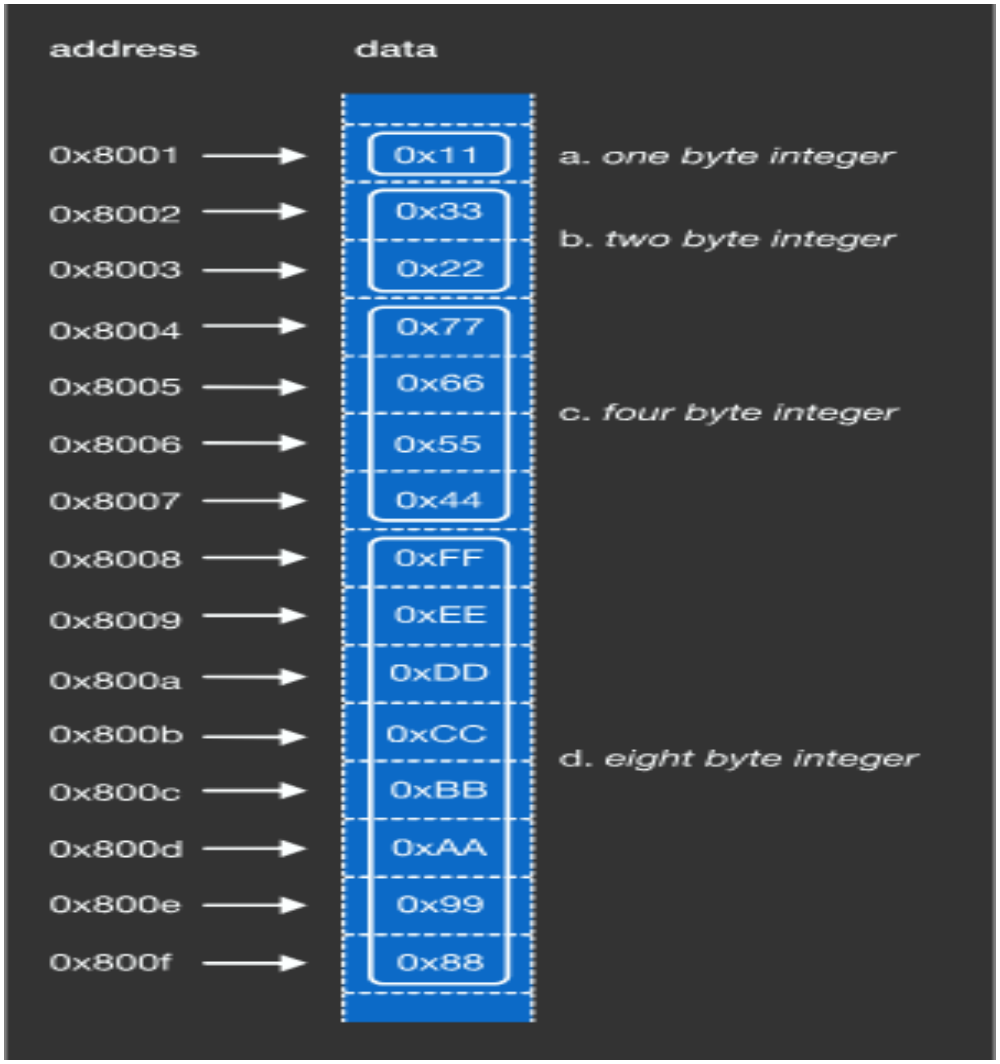
Correct answer: **15**

Explanation

- 1 + 2 + 4 + 8 =15.
- The total memory needed is 15 bytes.

- Let’s consider the memory of the following variables:
- **char a = 0x11;**
- **short b = 0x2233;**
- **int c = 0x44556677;**
- **long d = 0x8899AABBCCDDEEFF;**

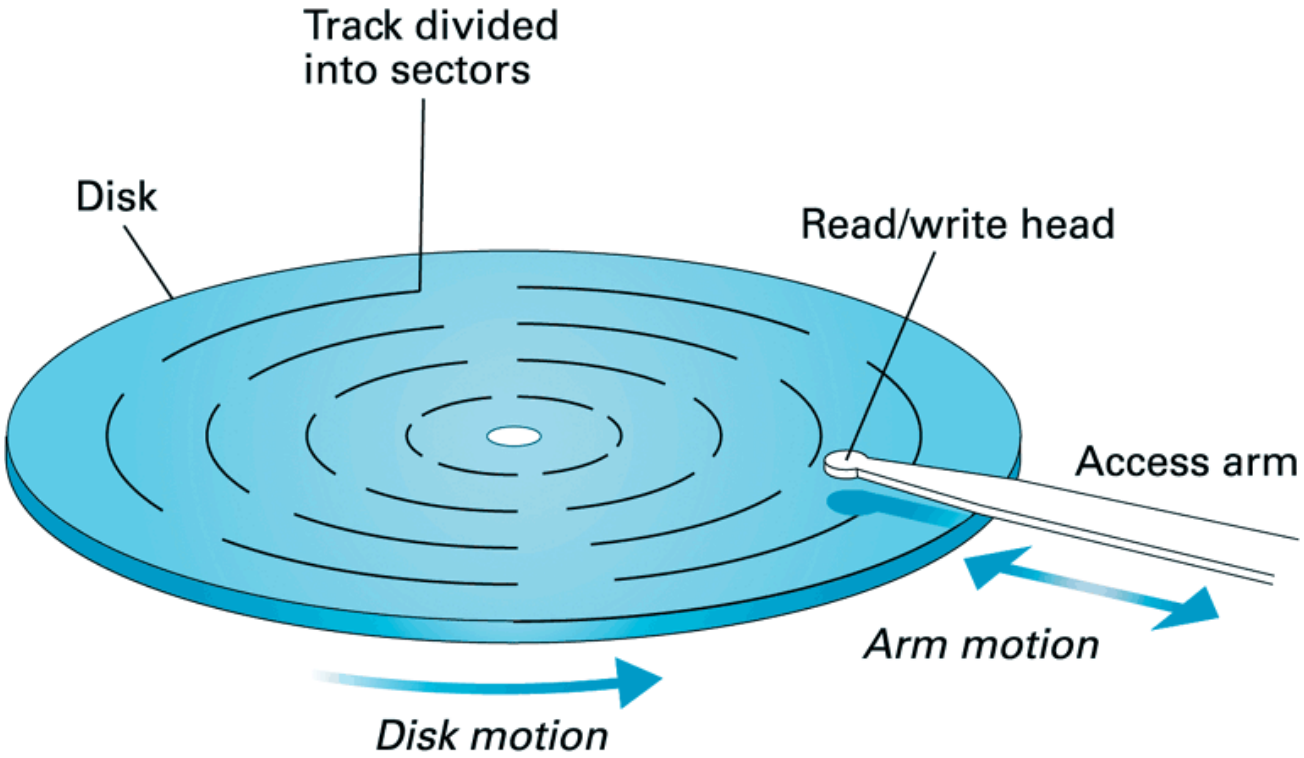
On an example environment, the memory layout may look like the following diagram:



- In this example, **the order of variables defined in the source code and the order of the data stored in memory matched each other.**
- Furthermore, the variables were placed next to each other in memory without any padding in between. **However, in actual situations, the compiler may reorder variable allocation and add padding for optimization.**

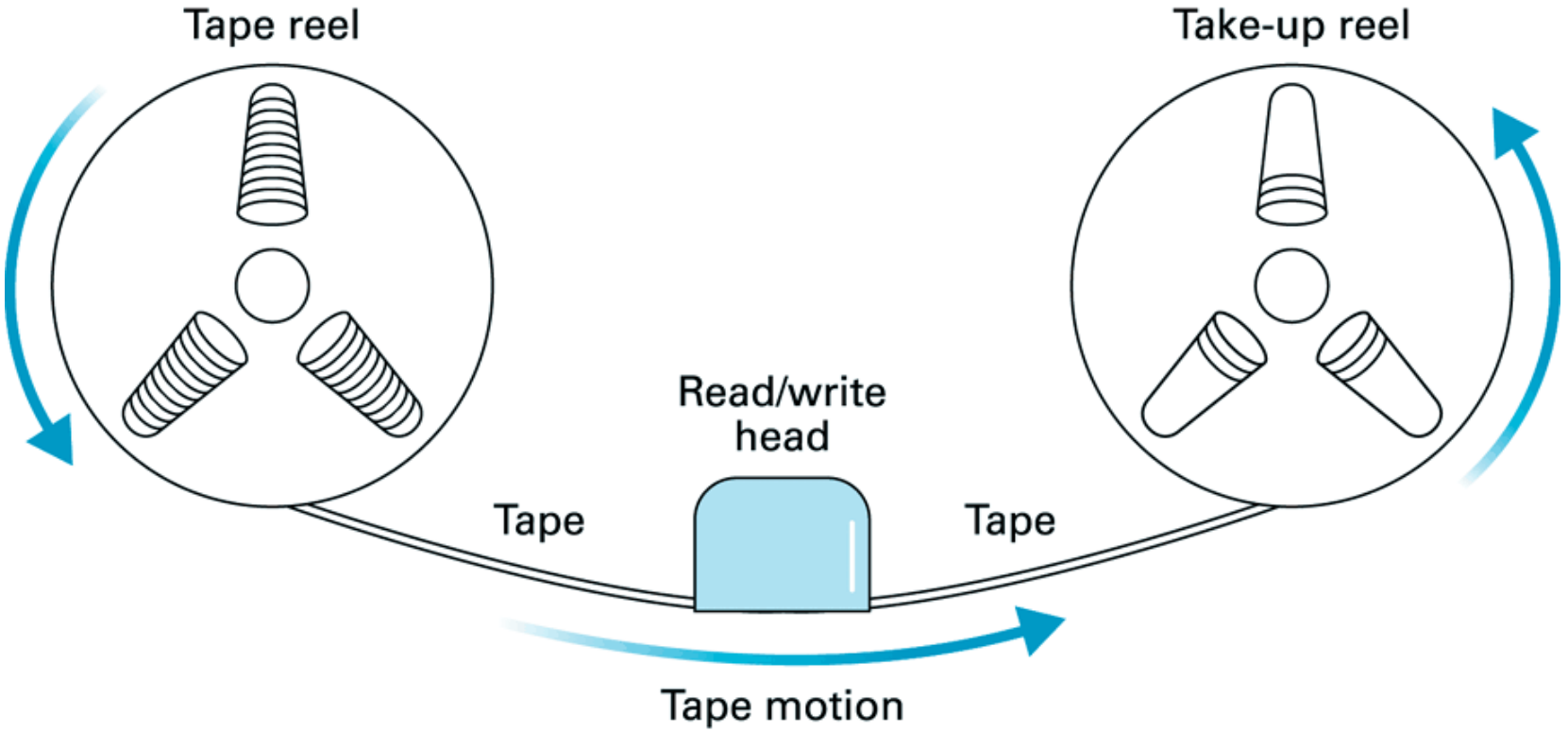
Mass Storage Devices

1.3 A Magnetic Disk Storage System



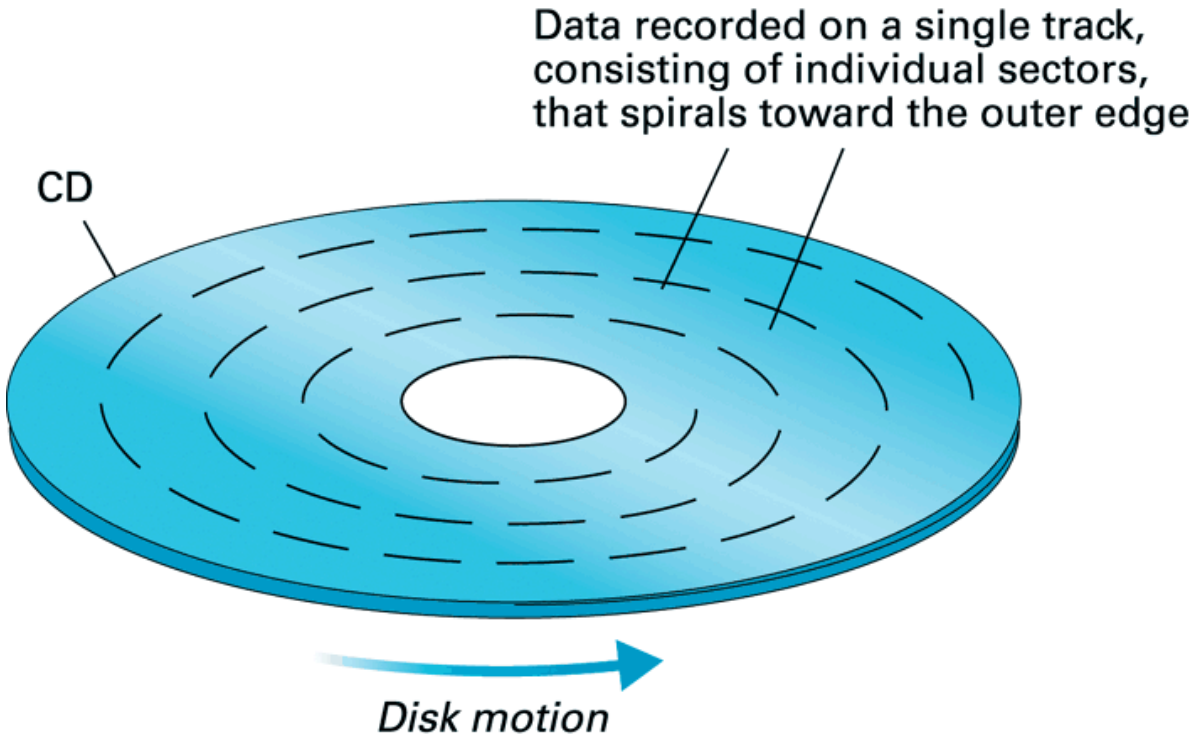
- Each track contains same number of sectors
  - **Sector Size -> 512 or 1024 KB**
- Location of tracks and sectors not permanent (*formatting*)
- Examples: hard disks, floppy disks, Zip disks, ...
- **Evaluating disk's performance:** seek, latency and access times
  - transfer rate

1.3 Old, but still commonly used: Magnetic Tape



- **Offers little or no random access (slow!)**
- Good choice for off-line data storage (archives)

1.3 CD/DVD Storage Format



- Data stored by creating variations in the reflective surface
- Data retrieved by means of a laser beam
- Data stored uniformly (so CD rotation speed varies)
- Random access much slower than for magnetic disks

Media type
High-density optical disc

Capacity
25 GB (single-layer)
50 GB (dual-layer)
100/128 GB (BDXL)
Block size 64 kb

Usage
Data storage
High-definition video (1080p)
High-definition audio
Stereoscopic 3D
PlayStation 3 games

The name Blu-ray Disc refers to the blue laser used to read the disc, which allows information to be stored at a greater density than is possible with the longer-wavelength red laser used for DVDs. The major application of Blu-ray Discs is as a medium for video material such as feature films.

1.3 Mass Storage

- **Main memory is *volatile* and limited in size**
- **Additional memory devices for mass storage:**
  - a.o.: magnetic disks, optical disks, magnetic tapes
  - Online and Offline Devices
- **Advantages over main memory:**
  - less volatile, large capacity, capability of removal, generally much cheaper
- **Disadvantages over main memory**
  - mechanical motion for data access/retrieval (Response time slow!)
  - in general: lesser degree of random access

File Storage and Retrieval

- Storage in mass storage medium->Files
- A block of data conforming to the physical characteristics of storing device is called **Physical records.**
- A file usually has natural divisions determined by the information represented. It describes **Logical records.**
- Problem: scattered data
- Solution: **Buffer**
- **Degree of random access of the data**


Any
Questions