Memory

Chapter 2

Primary Memory

Memory is the part of the computer where programs and data are stored

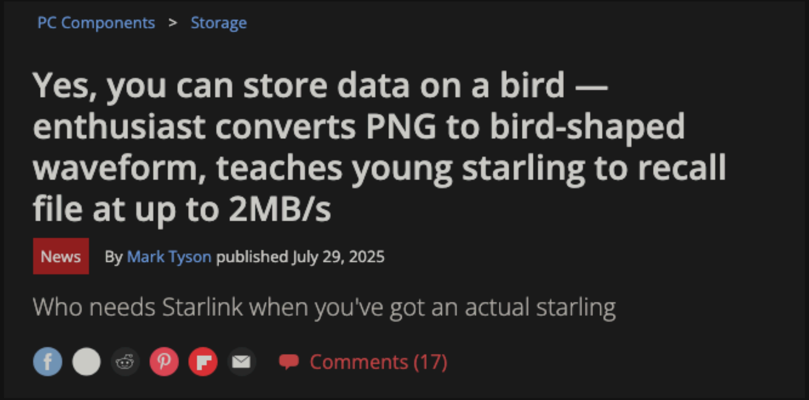
Often called *store* or *storage*, but the term *storage* tends to mean disk storage

Bits

A bit is the basic unit of memory, it can either be a `0` or a `1`, and it's the simplest possible unit

Binary arithmetic is used because it's "efficient" but what that means is that it's easier to distinguish between two states than it is to distinguish between many states

You can store any value in any form of memory, but the way you store it will affect how you access it, and how efficiently and reliable you can access it



PC Components > Storage

Yes, you can store data on a bird — enthusiast converts PNG to bird-shaped waveform, teaches young starling to recall file at up to 2MB/s

News By Mark Tyson published July 29, 2025

Who needs Starlink when you've got an actual starling

Comments (17)

BCD

Some computers like old IBM mainframes are advertised to have decimal as well as binary arithmetic.

They do this by using 4 bits to store one decimal digit using a code called BCD (Binary Coded Decimal)

Four bits provide 16 combinations, used for the 10 digits 0-9 plus 6 unused combinations

The number 1944 in BCD would be

$0001\ 1001\ 0100\ 0100$ and in binary it would be 0000011110011000_2

Sixteen bits each, but the BCD version can store up to `9999` while the 16 bit binary can store up to `65,536`

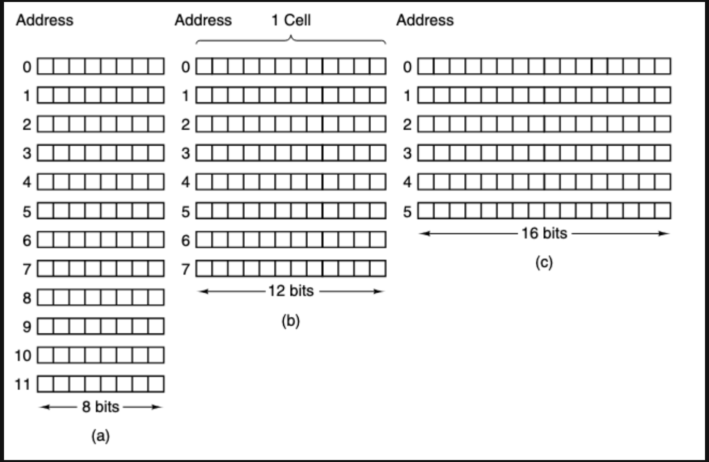
Memory Addresses

A memory system consists of *cells* or *locations* which can store a piece of information, and has a unique *address*.

And if a memory has n cells, they will have addresses from 0 to $n - 1$

If a cell consists of k bists, then it can hold any one of 2^k bit combinations

Assuming you have 96-bit memory, you can make a storage system:



Memory Addresses

Computers that use the binary number system express memory addresses as binary numbers

And if an address has m bits, the maximum number of addressable cells is 2^m

For example to represent 11 addresses, you need at least 4 bits, because $2^4 = 16$ and $2^3 = 8$

And it's independent of the size of the cells

A memory system with 2^{12} cells, with 8 bits each, and a memory of 2^{12} cells with 64 bits each would both need 12-bit addresses

Bytes and Words

An 8 bit cell is usually called a *byte*, sometimes called an *octet*

Bytes are grouped into `words`, and a computer with a 32-bit word, has 4 bytes per word while a computer with a 64-bit word has 8 bytes per word

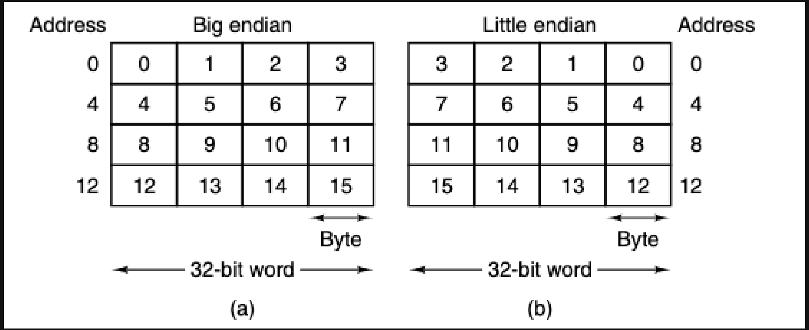
This is important because most instructions operate on entire words, like adding two words together

So a 32 bit machine will have 32 bit registers and instructions for manipulating 32 bit words

But a 64 bit machine will have 64 bit registers and instructions for manipulating 64 bit words

Byte Ordering

The bytes in a word can be numbered from left to right, or from right to left, this has major implications



The left image, where the numbering starts at the "big" end is called *big-endian*

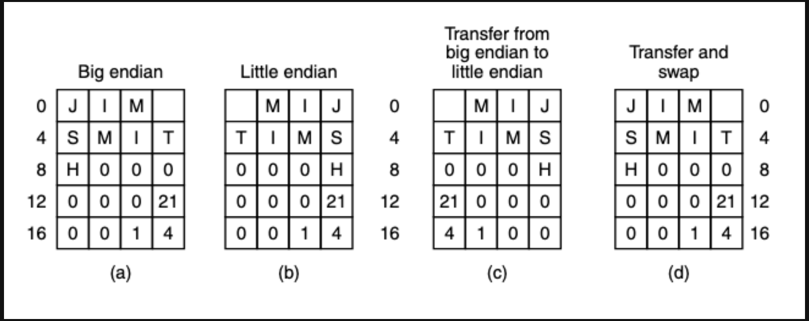
And the right image, where the numbering starts at the "little" end is called *little-endian*

If we only ever represented numbers, it would be easy to convert between the two

00000000 00000000 00000000 00000101

Byte Ordering

But if we have mixes of numbers and characters, like in a string, then the ordering matters



Big endian

0	J	I	M	
4	S	M	I	T
8	H	0	0	0
12	0	0	0	21
16	0	0	1	4

(a)

Little endian

	M	I	J
T	I	M	S
0	0	0	H
0	0	0	21
0	0	1	4

(b)

Transfer from big endian to little endian

0		M	I	J
4	T	I	M	S
8	0	0	0	H
12	21	0	0	0
16	4	1	0	0

(c)

Transfer and swap

J	I	M		0
S	M	I	T	4
H	0	0	0	8
0	0	0	21	12
0	0	1	4	16

(d)

For a single machine, it doesn’t matter, but when one machine sends data to another, it can cause problems

Assuming that the big endian system sends the record one byte at a time

The big endians 0 goes to the little endians 0, the big endians 1 goes to the little endians 1, and so on

Error correcting codes

Computer memories aren't perfect, voltage spikes, cosmic rays, and other factors can cause bits to flip

So we need to be able to detect and correct errors, using error-detecting and error-correcting codes

What is an error?

Imagine a memory word that has m data bits with an added r data bits called redundant/check bits. With the total number of bits being $n = m + r$. This is called a `codeword`

And given any two `codewords` like 100001001 and 10110001 it's possible to determine how many corresponding bits differ by using an XOR bitwise operation and counting the ones

This is called the *Hamming distance* between two `codewords`

Hamming Distance

This hamming distance means that if two objects are d distance apart, then they need d amount of errors to happen to convert one into the other

With an m-bit memory word, all 2^m possible combinations are valid, but only 2^m of the 2^n `codewords` are actually valid

And if the read turns up an invalid `codeword`, the computer knows that a memory error has occurred, because given the algorithm for computing the check bits, it's possible to construct a complete list of the legal `codewords`.

Then find the `codeword` that is closest to the read codeword, and the distance between those two is the Hamming distance of the complete code

Error Detection and Correction

To do error detecting and correction, we rely on the Hamming distance

To detect d single-bit errors, you need a distance of $d + 1$

And to correct d single-bit errors, you need a distance of $2d + 1$

A simple example of an error-detecting code is the *parity bit*, which is a single bit that is added to a word to make the number of ones even or odd

Cache Memory

Historically, CPUs have always been faster than memories

There are two main ways to deal with this imbalance. The simplest way is to just start memory reads when they are encountered but continue executing and stall the CPU if an instruction tries to read the memory word before it arrives

- the slower the memory, the longer the stall

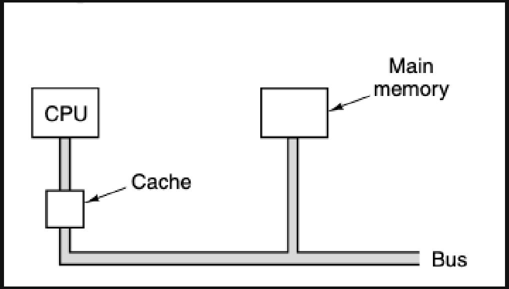
The other solution is to have machines that do not stall but instead require the compilers not to generate code to use words before they have arrived

- the problem with this approach is that it’s harder and inconsistent
- after a `load` operation, there tends to be nothing to do until it loads so you just `nop`

Cache Memory

The problem is economics, we can build faster memory, but for it to run as fast as the CPU, it would need to be on the CPU itself. Putting memory on a CPU chip makes it bigger, and that makes it more expensive, so we have to keep that memory small

So we have a compromise, we have a small amount of fast memory on the CPU chip, and a larger amount of slower memory off the chip



The small but fast memory is called *cache*

The cache is given the most heavily used memory words,

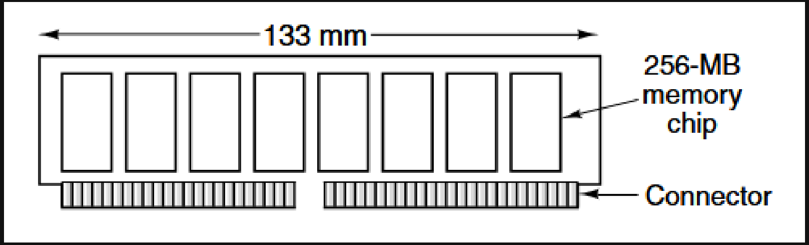
- and if the CPU needs a word, it checks the cache first,
- if the word is in the cache,
- it can be read quickly,
- if not, it has to be read from the slower memory

The way the computer chooses which items to put in cache is called the *locality principle* which is the basis for all forms of caching algorithms. This principle is from the observation that memory references tend to only use a small fraction of the total memory at a time

Where data like data in loops, data next to other data, instructions next to other instructions, are sent to cache memory ahead of time

Memory Packaging and Types

SIMM (Single Inline Memory Module) and DIMM (Dual Inline Memory Module) are the two main types of memory packaging



Then there's SO-DIMM(Small Outline DIMM) for laptops and other small devices

Also ECC memory

Secondary Memory

The main memory is always too small

Memory Hierarchies

To store a large amount of data, a memory hierarchy is used, which is a set of storage devices with different speeds and sizes

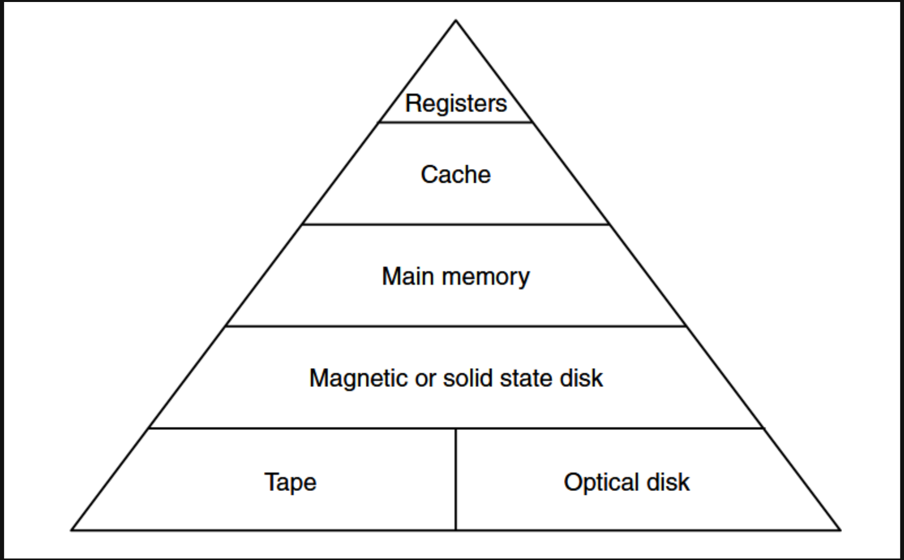
First is registers, second is cache, etc

As you go lower down the hierarchy, three key parameters increase.

- Access time gets longer
- The storage capacity increases
- The number of bits per dollar increase

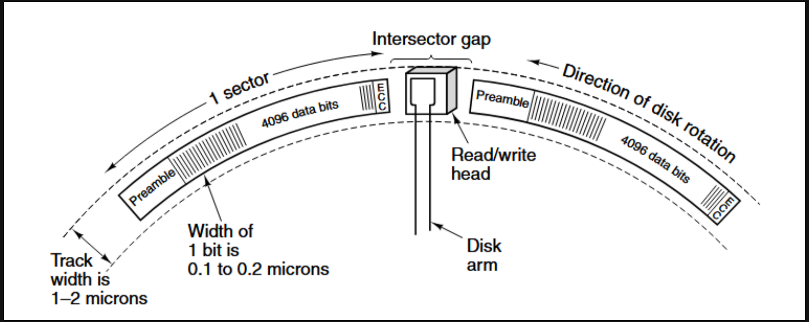
Demo

Demo2



Magnetic Disks

Aluminum platters with a magnetizable coating, which is read by a head that floats above the surface



If a positive or negative current passes through the head, it magnetizes the surface just beneath it. Aligning the magnetic particles on that area.

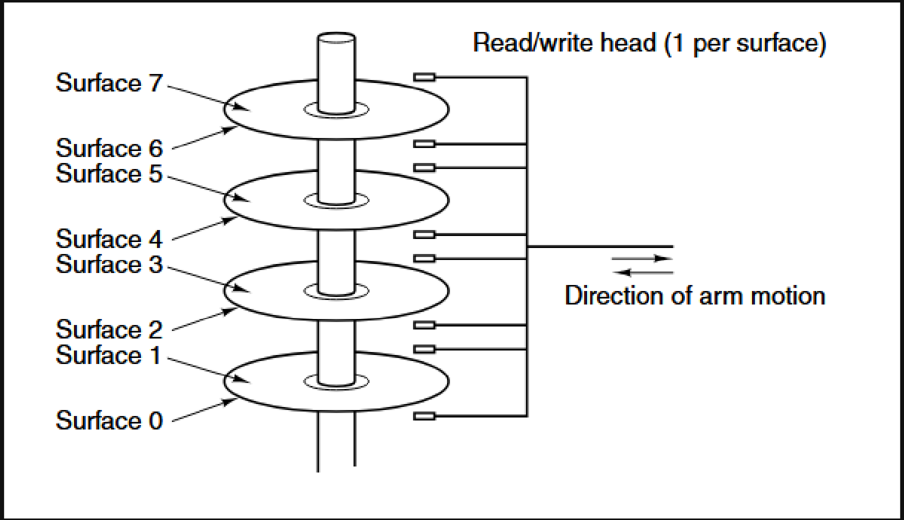
And when the head passes over a magnetized area, it induces a current on the head, which is converted into data.

Magnetic Disks

- A circular sequence of bits that complete a rotation is called a `track`
- each track is divided up into `sectors`
- each sector starts with a `preamble` to help sync the head and the data
- at the end of the sector is some error correcting code

It's usually called a *hard disk* to differentiate it from a *floppy disk*

And most disks have multiple platters, each with its own head glued together



RAID

Redundant Array of Inexpensive/Independent Disks

CPU performance has been increasing, but not as fast as the speed of secondary memory.

Like computers, to increase the speed of disks, people realized you could just have more of them

In 1988, Patterson and others suggesting six ways to organize disks to improve performance, which was then called RAID

The idea was to get multiple disks and replace the controller with a single RAID controller. And it acts like a single disk to the computer

The way to do this is to organize them in different ways, Level 0 to 6

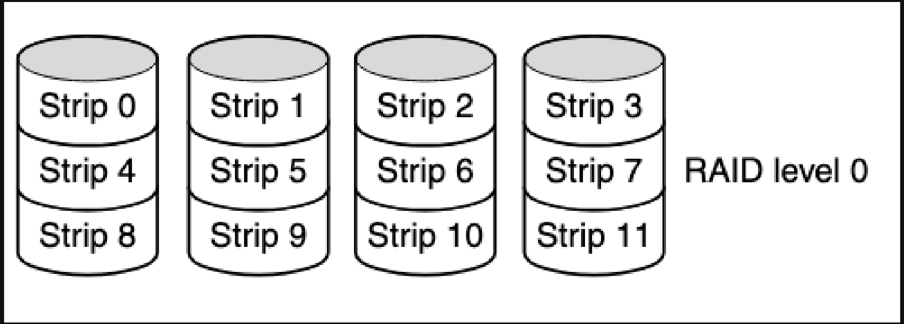
note that levels don't mean better, they just mean different

RAID 0

This is called *striping*

Where you distribute data over multiple disks, so that when a call for a large amount of data with a single read happens, we can pull that data in parallel

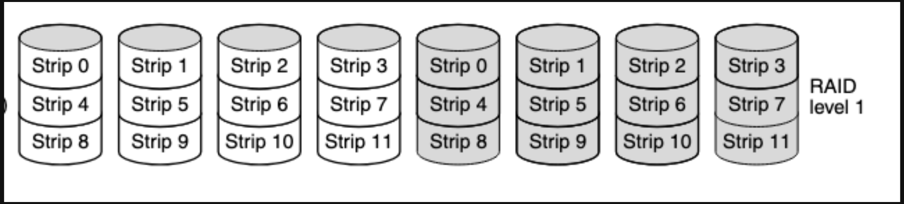
This is the simplest with the clearest disadvantages. It’s also less reliable



RAID 1

Is considered a *true* RAID, and it duplicates all the disks

In any read, either copy can be used, and it has significantly better reliability where if a drive crashes, the other drive can be used, and then recovered

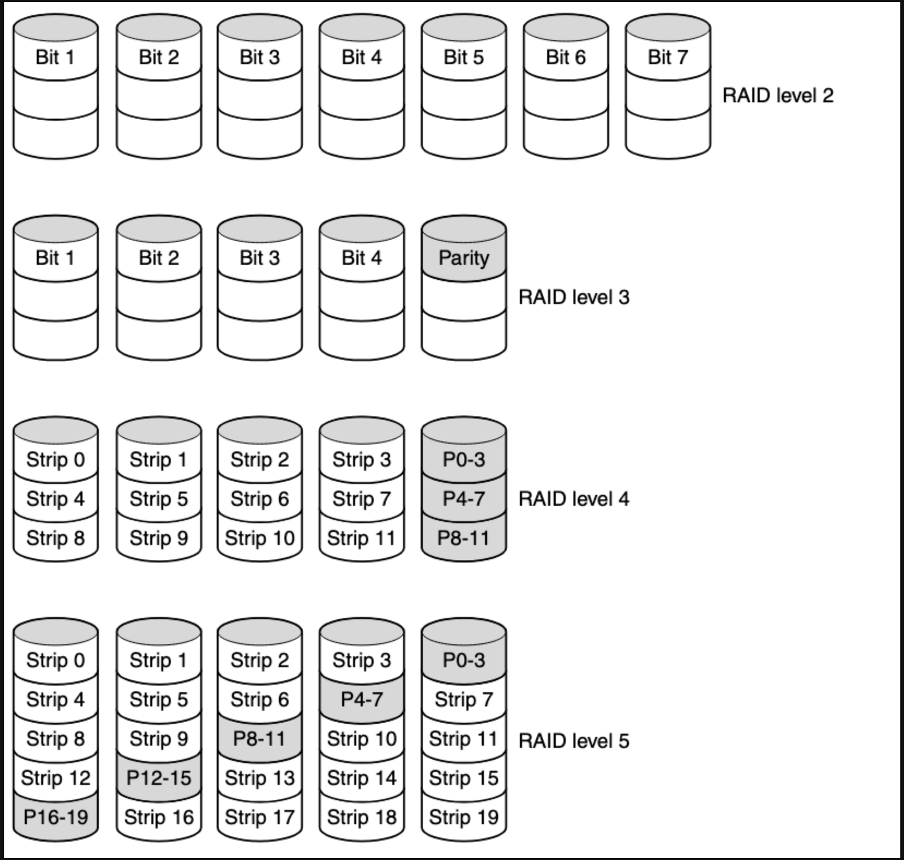


Other RAIDS

Levels 2-5 aren't as common, but they are used in some extreme systems

2. RAID 2 is a bit-level striping with Hamming code error correction
3. RAID 3 is byte-level striping with a dedicated parity disk
4. RAID 4 is block-level striping with a dedicated parity disk
5. RAID 5 is block-level striping with distributed parity

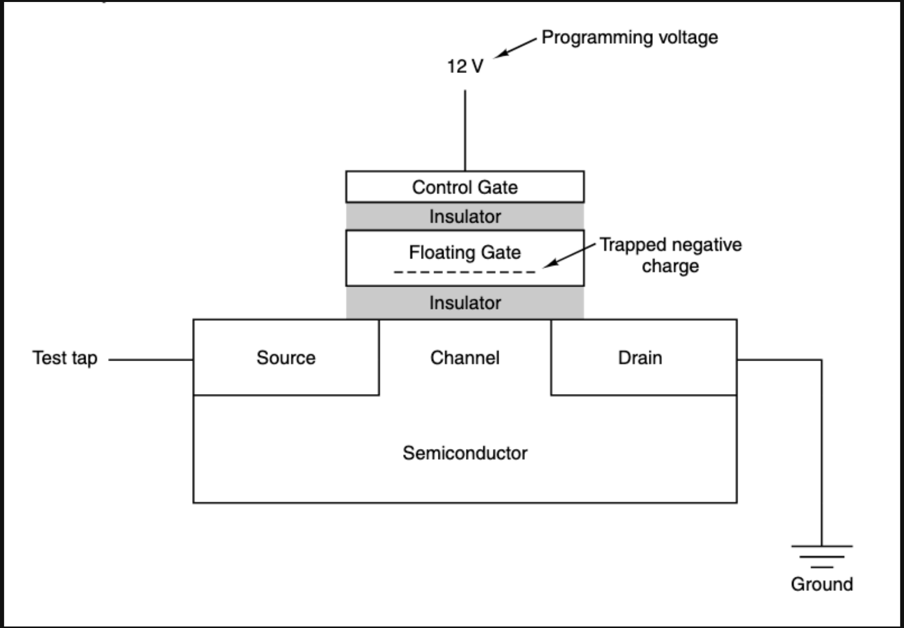
All these attempt to increase performance while maintaining reliability, at the cost of complexity and loss of raw storage space



Solid State Disks

Transistors can fail over time as it’s used. One way it can fail is called "hot carrier injection" where an electron charge gets embedded inside a once-working transistor, where it’s permanently on or off

Flash disks are made of many solid-state flash memory cells made of a special type of transistor.



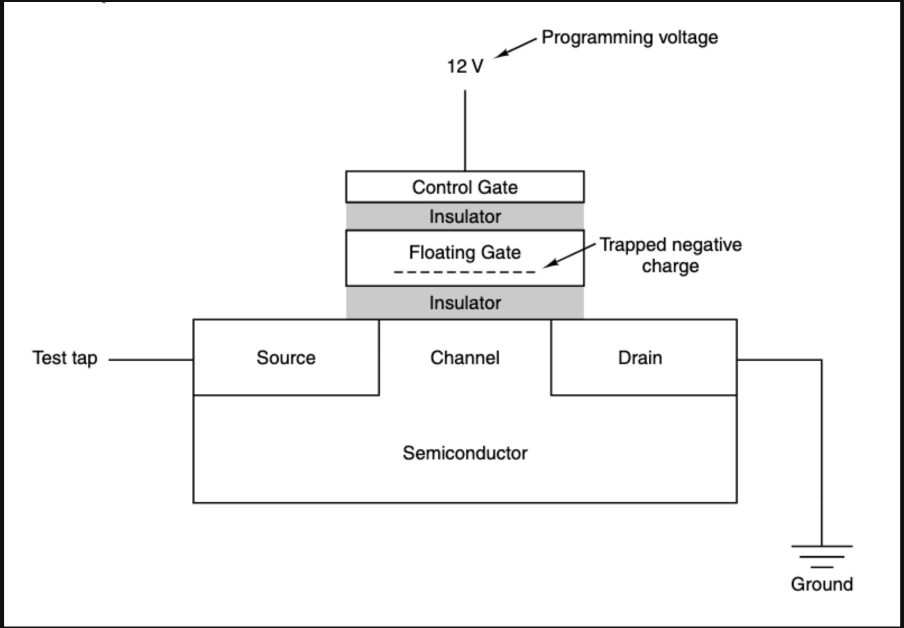
Solid State Disks

By default, this acts like a transistor, but by pushing a high voltage to the control gate, we can trap electrons in the floating gate, which changes the threshold voltage of the transistor

This means the internals get a negative charge, which means it takes more voltage to turn the transistor on

And by testing how much voltage it takes to turn the transistor on, we can determine if it's a `0` or a `1`

This also means that an SSD is *non-volatile*, because even if it loses power, the electrons are still trapped in the floating gate



Solid State Disks

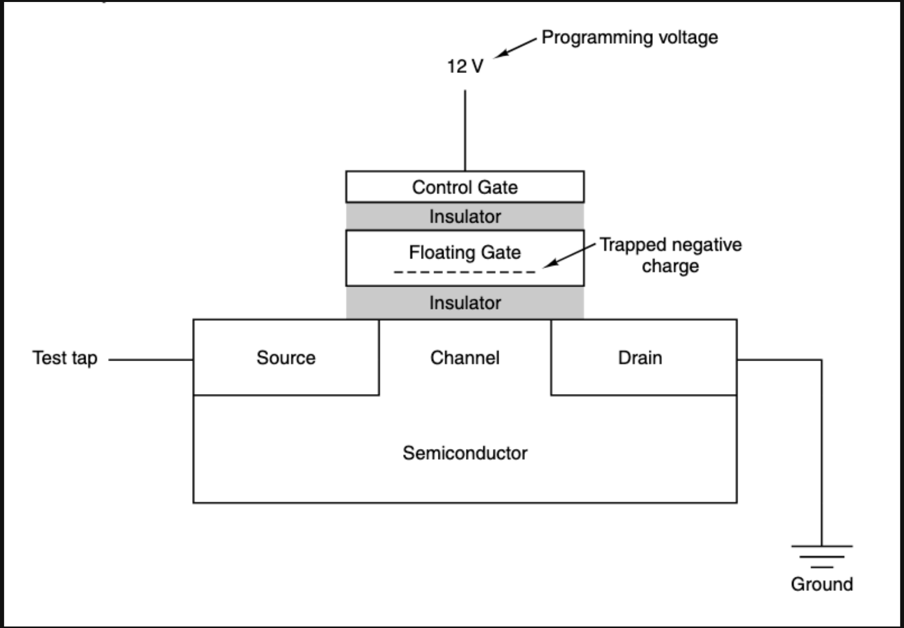
Because SSDs are essentially memory chips, they have significantly faster access times than magnetic disks since they don't have to *seek* or *spin*

It's also more reliable since there are no moving parts

The main problem is cost per bit, and their failure rate

Typical flash cells can only be written to about 100,00 times before it loses functions

This is counteracted by *wear leveling* which distributes writes evenly across the disk



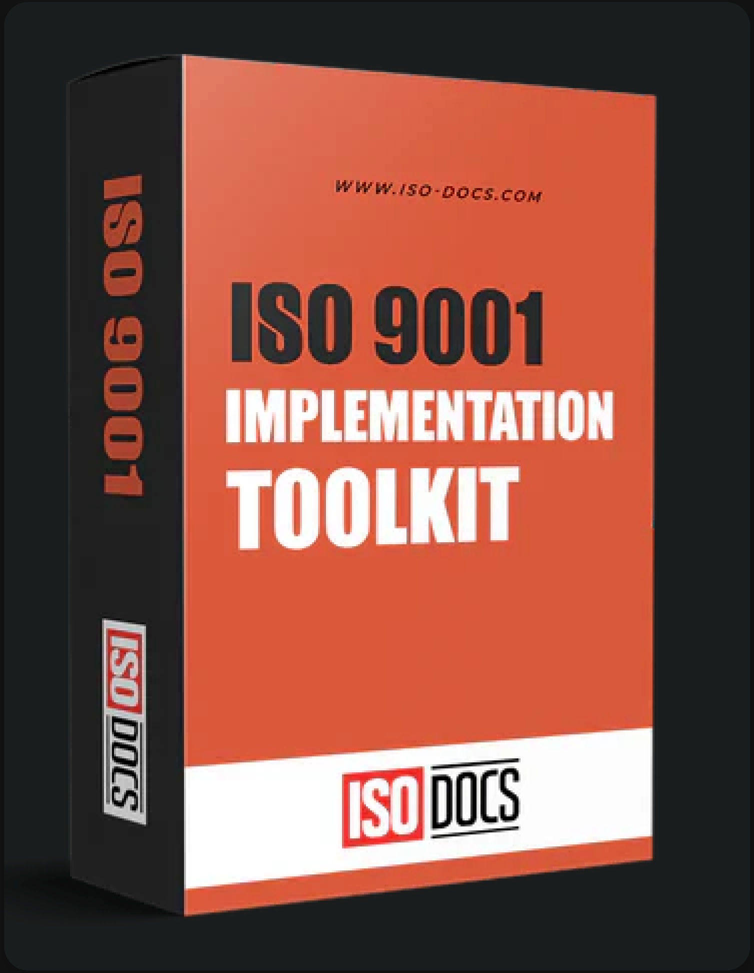
CD-ROMs

CDS were used primary for distribution of software and media, as well as backups because of their low cost and large capacity

At around the 1980s, Sony developed the CD (Compact Disc) which quickly replaced vinyl records for music.

Then the precise technical details for the CD were published in an official International Standard (ISO 10149) popularly known as the Red Book

This allowed other manufacturers of both CDs and CD players to be made in a way that was compatible

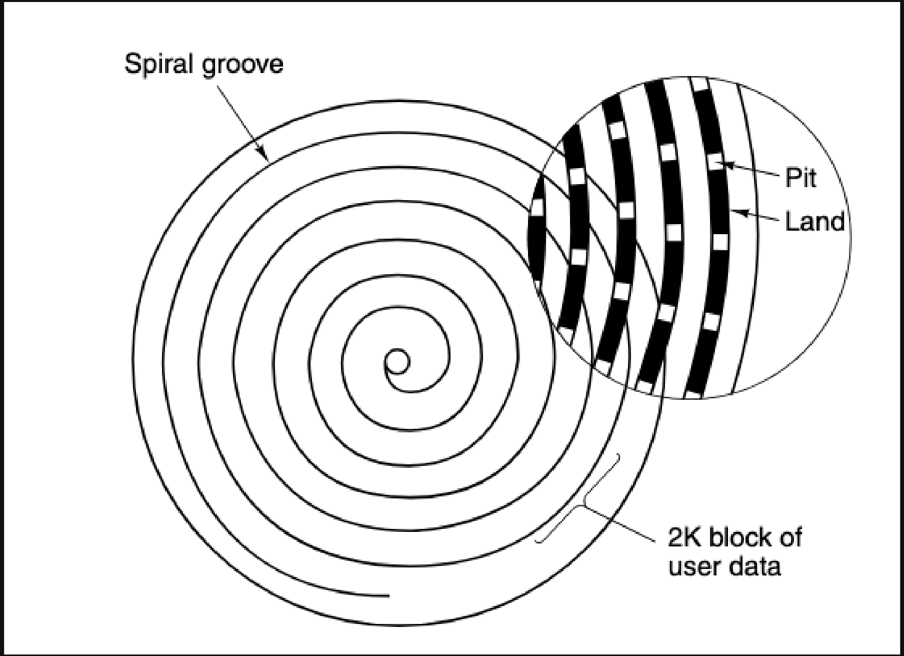


CD-ROMs

The way a CD is made by using a high power infrared laser to burn small hols in the glass master disk. Then a mold is made from that master disk, with bumps instead of holes.

The depressions in the disk are called *pits* and unburned areas in between are called *lands*

And the CD is played by pointing a laser at the surface, and depending on whether the laser is reflected or not, it reads a `0` or a `1`



Developments of the CD

1. `CD-R (Recordable)` were developed to allow users to write data to a CD, but only once
2. `CD-RW (Rewritable)` were developed to allow users to write and erase data multiple times, but with a limited number of write cycles
3. `DVD (Digital Versatile Disc)` was developed to store larger amounts of data, up to 4.7 GB for single-layer discs and 8.5 GB for dual-layer discs
4. `Blu-ray Disc` was developed to store even larger amounts of data, up to 25 GB for single-layer discs and 50 GB for dual-layer discs, using a blue laser instead of a red one

Quiz

https://ishortn.ink/ComOrgQ4