Measuring the frequency, f or period, t_p of a signal

Direct measurement of a period: *Count timer clock pulses (tic) for one cycle of the signal*.

Illustration: The microcontroller's counter-timer system has a 1 Hz clock.
A counter is set to run (counting clock cycles) from the rising edge of the input-signal-to-be-measured to the next rising edge. 120 counts are observed.
The input signal has a 120 second period.

In general $t_p = N t_{ck}$ where t_p = period measured, $t_{ck} = 1/f_{ck}$ = counter's clock period, N = the count.

1

Measuring the frequency, f or period, t_p of a signal

Direct measurement of a period: *Count system clock pulses for one cycle of the signal*.

Illustration: The microcontroller's counter-timer system has a 1 Hz clock.
A counter is set to run (counting clock cycles) from the rising edge of the input-signal-to-be-measured to the next rising edge. 120 counts are observed.
The input signal has a 120 second period.

In general $t_p = N t_{ck}$ where t_p = period measured, $t_{ck} = 1/f_{ck}$ = counter's clock period, N = the count.

Direct measurement of a frequency: *Count the number of cycles of the input signal to be measured in a defined interval.*

Illustration: The microcontroller's counter-timer system has a 1 second clock.
A counter is set to run (counting input-signal cycles) for exactly one clock cycle (1 second).
33 counts are observed. The input signal has a frequency of 33 Hz.

In general $f = N f_{ck}$ where f = frequency measured, f_{ck} = system clock frequency, and N = the count.

2

Measuring the frequency, f or period, t_p of a signal

Direct measurement of a period: *Count system clock pulses for one cycle of the signal.*
Illustration: The microcontroller's counter-timer system has a 1 Hz clock.
A counter is set to run (counting clock cycles) from the rising edge of the input-signal-to-be-measured to the next rising edge. 120 counts are observed.
The input signal has a 120 second period.
In general $t_p = Nt_{ck}$ where t_p = period measured, $t_{ck} = 1/f_{ck}$ = counter's clock period, N = the count

Direct measurement of a frequency: *Count the number of cycles of the input signal to be measured in a defined interval.*
Illustration: The microcontroller's counter-timer system has a 1 second clock.
A counter is set to run (counting input-signal cycles) for exactly one clock cycle (1 second).
33 counts are observed. The input signal has a frequency of 33 Hz.
In general $f = Nf_{ck}$ where f = frequency measured, f_{ck} = system clock frequency, and N = the count.

Indirect measurement of a period: Measure the frequency, calculate $t_p = 1/f$.

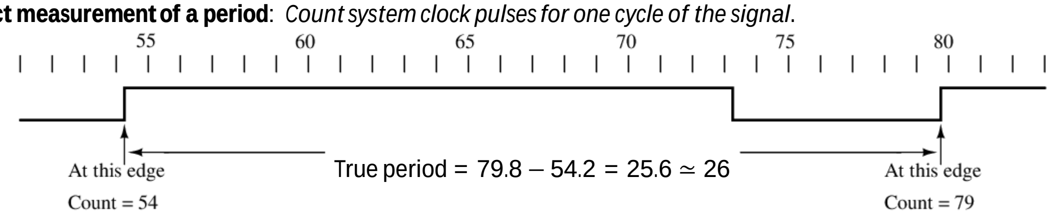
Indirect measurement of a frequency: Measure the period, calculate $f = 1/t_p$.

3

Measuring the frequency, f or period, t_p of a signal

Consider the details of these methods.

Direct measurement of a period: *Count system clock pulses for one cycle of the signal.*



This technique allows an uncertainty of up to ±1 count.
In this case, period becomes 79 − 54 = 25, error = 0.6 of a count.
Thus, uncertainty up to $\pm 1/f_{ck}$. Where f_{ck} is the clock freq. of the counter. Uncertainty is independent of signal—good!
Or, uncertainty up to $\pm 1\ t_{ck}$ where a t_{ck} is the period at which the counter increments, sometimes called the *tic interval*.

The counts might include a rollover event. E.g. an 8-bit counter runs from 0 to 255. The above signal might produce start = 254, end = 23. If result is negative, add 256. In this case, 23 − 254 = −231. That's neg so +256, get 25.
This can be done in one step via tricky manipulation of an ALU on some CPUs.

Potential pitfall: The update rate of the measurement depends on the period of the signal itself—bad!
If the signal stops (or slows too much) while a measurement is in progress the measurement never ends—crash!
One way to prevent the crash is to force a return (with an error code set) if there are two rollovers before the final edge.

4

Measuring the frequency, f or period, t_p of a signal

Consider the details of these methods.

Direct measurement of a Frequency: *Choose a fixed interval like one second (use tic-clock or a counter on the system clock to create this interval) then count the signal's cycles in that interval.*

Example 100 pulses counted in a 1/5 second interval ($f_{ck} = 5$ Hz). $f = 100\ (5) = 500$ Hz.

In general, the interval of counting is t_{ck} seconds long and the count is N

$$f = N f_{ck}$$

Resolution is ± 1 count or $\Delta f = f_{ck}$
In the above example, the resolution is 5 Hz

Uncertainty is independent of the signal—good!
Measurement update rate is independent of the signal—good!
Signal must be fast enough relative to your chosen counting interval—limitation!
Thus there are cases where you would prefer an indirect measurement.

5

Measuring the frequency, f or period, t_p of a signal

Consider the details of these methods.

Indirect measurement of a Frequency: *Count clock tics in a period of the signal, calculate $t_p = N t_{ck}$ then convert to frequency: $f = 1/t_p$.*

We know that the uncertainty of the period is $\Delta t_p = 1/f_{ck}$,
but since we want frequency, we also want to know the uncertainty of the frequency. (Spoiler: Weirdness ensues!)

If the count changes one unit, altered freq. is $f \pm \Delta f = \frac{1}{(t_p \mp \Delta t_p)}$ and now solve for Δf, the resolution or precision.

$$f \pm \Delta f = \frac{1}{\frac{1}{f} \mp \Delta t_p}$$

$$\pm \Delta f = \frac{1}{\frac{1}{f} \mp \Delta t_p} - f = \frac{1}{\frac{1}{f} \mp \Delta t_p} - \frac{f\left(\frac{1}{f} \mp \Delta t_p\right)}{\frac{1}{f} \mp \Delta t_p} = \frac{1}{\frac{1}{f} \mp \Delta t_p} - \frac{1 \mp f\Delta t_p}{\frac{1}{f} \mp \Delta t_p} = \frac{\pm f \Delta t_p}{\frac{1}{f} \mp \Delta t_p}$$

Assume there are many counts in the measurement, i.e. period of signal >> period of counter's clock. $1/f \gg \Delta t_p$
Then Δt_p is insignificant in the denominator.

$$\Delta f \simeq f^2 \Delta t_p$$

Uncertainty is dependent of the signal—bad for high-frequency signals! (Good for low frequency signals!)
Measurement update rate is dependent on the signal—bad! (Signal better not get too low in frequency!)

6

Measuring the frequency, f or period, t_p of a signal

Consider the details of these methods.

Indirect measurement of a Period: *Count cycles of the signal in a defined interval f=* Nf_{ck}
then convert to period: $t_p = 1/f$.

We know that the uncertainty of the frequency is $\Delta f = f_{ck}$,
but since we want frequency, we also want to know the uncertainty of the frequency. (Spoiler: Weirdness ensues!)

If the count changes one unit, altered period is $t_p \pm \Delta t_p = \frac{1}{f \mp \Delta f}$ and now solve for Δt_p, the resolution or precision.

$$t_p \pm \Delta t_p = \frac{1}{1/t_p \mp \Delta f}$$

$$\pm \Delta t_p = \frac{1}{1/t_p \mp \Delta f} - t_p = \frac{1}{\frac{1}{t_p} \mp \Delta f} - \frac{t_p\left(\frac{1}{t_p} \mp \Delta f\right)}{\left(\frac{1}{t_p} \mp \Delta f\right)} = \frac{1}{\frac{1}{t_p} \mp \Delta f} - \frac{1 \mp t_p \Delta f}{\frac{1}{t_p} \mp \Delta f} = \frac{\pm t_p \Delta f}{\frac{1}{t_p} \mp \Delta f}$$

Assume there are many counts in the measurement, i.e. frequency of signal >> freq. of counter's clock. $f \gg \Delta f$
Then Δf is insignificant in the denominator. (Recall that $\Delta f = f_{ck}$)

$$\Delta t_p \simeq (t_p)^2 f_{ck}$$

Uncertainty is dependent of the signal—bad for long-period signals! (Good for short-period signals!)
Measurement update rate is independent on the signal—good!

7

SUMMARY SLIDE

Measuring the frequency, f or period, t_p of a signal

Direct measurement of a period: *Count system clock pulses for one cycle of the signal.*
Illustration: The microcontroller's counter-timer system has a 1 Hz clock.
A counter is set to run (counting clock cycles) from the rising edge of the input-signal-to-be-measured to the next rising edge. 120 counts are observed.
The input signal has a 120 second period.
In general $t_p = Nt_{ck}$ where t_p = period measured, $t_{ck} = 1/f_{ck}$ = counter's clock period, N = the count.
$\Delta t_p = \pm t_{ck}$ Constant resolution. Measurement rate dependent on signal period.
Pitfall: slow or stopped signal must not be allowed to crash system—monitor rollovers.

Direct measurement of a frequency: *Count the number of cycles of the input signal to be measured in a defined interval.*
Illustration: The microcontroller's counter-timer system has a 1 second clock.
A counter is set to run (counting input-signal cycles) for exactly one clock cycle (1 second).
33 counts are observed. The input signal has a frequency of 33 Hz.
In general $f = Nf_{ck}$ where f_{ck} = system clock frequency and N = the count
$\Delta f = \pm f_{ck}$ Constant resolution. Constant measurement rate. Minimum frequency limit exists.

Indirect measurement of a period: Measure the frequency, calculate $t_p = 1/f$.
$\Delta t_p \simeq (t_p)^2 f_{ck}$ Resolution depends on signal. Constant measurement rate.

Indirect measurement of a frequency: Measure the period, calculate $f = 1/t_p$.
$\Delta f \simeq f^2 \Delta t_p$ Resolution depends on signal. Measurement rate depends on signal.
If slow update rate is OK and rollovers monitored, handles slow signals well.

8

Digital Memory—Applications and Technologies

How might memory be used in an embedded system or in a system-on-a-chip?

9

Digital Memory—Applications and Technologies

Cloud/Networked Storage

Github, Dropbox, Google Drive, etc.
MAIN FEATURES:
  Non-volatile, sharing/access control
MAIN ISSUES
  Long access times.
  Security and privacy
  Reliability (if company bankrupt?)

Archival/Mass Storage

Flash memory, a.k.a. "Solid State,"
SD card, USB drive,
Rotating optical stuff: DVD, CD.
Magnetic stuff: hard- floppy-disk, tape
MAIN FEATURES
  Non-volatile
  Can be ported directly to your CPU
  Can be fast
MAIN ISSUES
  Reliability—need for backups
  Sharing is possible, but complex

Main Memory

RAM (Random Access Memory)
  --static, needs no refresh
    usually byte-wide
  --dynamic, needs refreshing
    word-wide modules
    bit or nibble-wide chips
RAM is volatile but fast

ROM (Read only memory)
  --mask programmed
  --Electrically programmed
  --Flash
ROM is non-volatile and fast but read only in these applications.

Virtual Memory

Hold "pages" of mass storage or main memory in main memory to make memory look larger. Especially valuable in wide-word machines.
Requires a memory controller

Cache Memory

Static RAM
--must be very fast
--usually on the CPU chip,
  or very nearby
--requires a memory controller

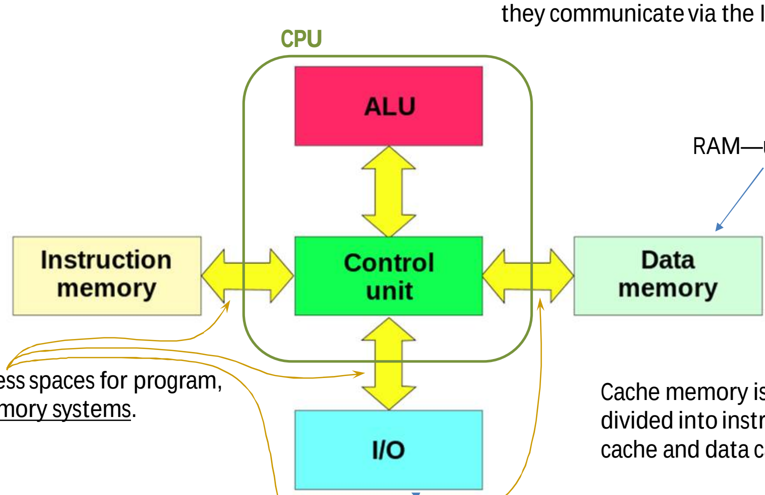
Register Memory

General purpose registers in the CPU
--fastest possible
--usually very limited in quantity,
  e.g. 64 words.
--"everybody" wants to use
  this memory. There is never
  enough of it.

10

Harvard Architecture: One of two popular ways to fit memory into a system

If additional CPU's are added to a system, they communicate via the I/O system.



RAM—usually static

Three entirely separate address spaces for program, data, and I/O ports. Two memory systems.

Cache memory is likewise divided into instruction cache and data cache.

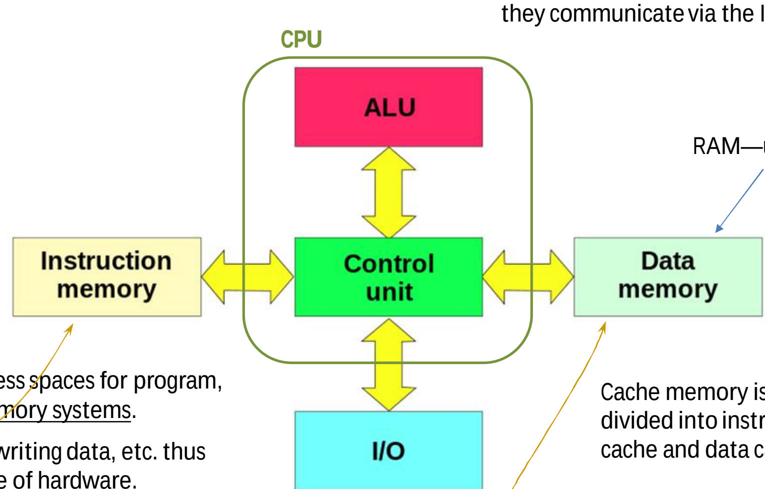
Archival/mass storage
(Also all other types of I/O ports, e.g. keyboard, mouse, display. . .)

https://commons.wikimedia.org/wiki/File:Harvard_architecture.svg

11

Harvard Architecture: One of two popular ways to fit memory into a system

If additional CPU's are added to a system, they communicate via the I/O system.



RAM—usually static

Three entirely separate address spaces for program, data, and I/O ports. Two memory systems.

Can fetch instructions while writing data, etc. thus quite fast, makes efficient use of hardware.

The two memory systems can be very different from each other. E.g. different word widths. Each of the two can be individually optimized to a particular task.

Cache memory is likewise divided into instruction cache and data cache.

Archival/mass storage
(Also all other types of I/O ports, e.g. keyboard, mouse, display. . .)

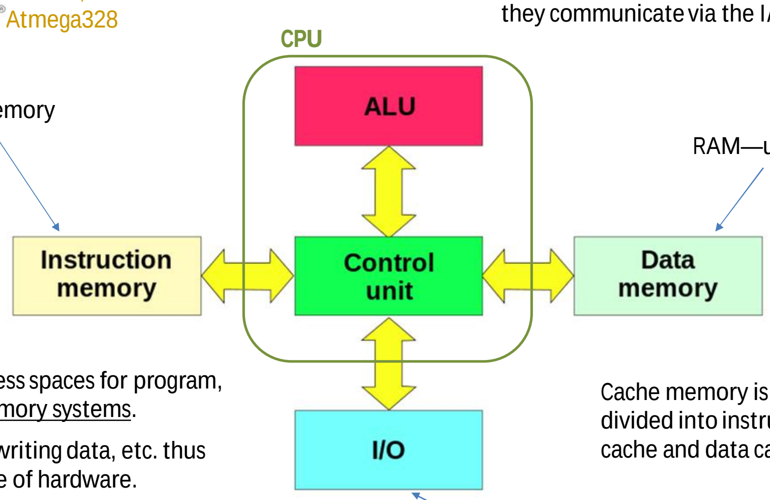
https://commons.wikimedia.org/wiki/File:Harvard_architecture.svg

12

<u>Harvard Architecture</u>: One of two popular ways to fit memory into a system

Popular in many microcontrollers, including Arduino's AVR® Atmega328

If additional CPU's are added to a system, they communicate via the I/O system.



Three entirely separate address spaces for program, data, and I/O ports. <u>Two memory systems</u>.

Can fetch instructions while writing data, etc. thus quite fast, makes efficient use of hardware.

The two memory systems can be very different from each other. E.g. different word widths. Each of the two can be individually optimized to a particular task.

Cache memory is likewise divided into instruction cache and data cache.

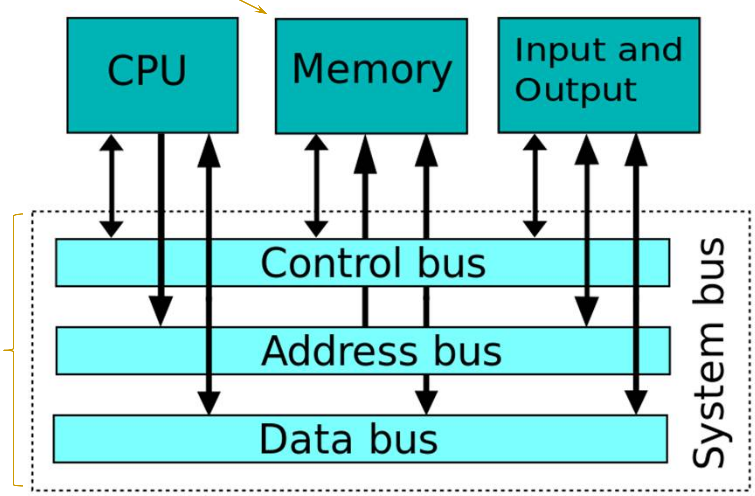
Archival/mass storage
(Also all other types of I/O ports, e.g. keyboard, mouse, display. . .)

https://commons.wikimedia.org/wiki/File:Harvard_architecture.svg

13

<u>Von Neuman Architecture</u> (a.k.a. Princeton Architecture): The other popular way to fit memory into a system

Everything connects via one "system bus." There is only <u>one memory system</u>, shared for all uses.



14

<u>Von Neuman Architecture</u> (a.k.a. Princeton Architecture): The other popular way to fit memory into a system

Everything connects via one "system bus." There is only <u>one memory system</u>, shared for all uses.
("Pure Von Neuman:" one address space. I/O operations are *memory mapped*. Or, "not-so-pure" separate memory and I/O addresses via a "MEM" bit on the control bus.)

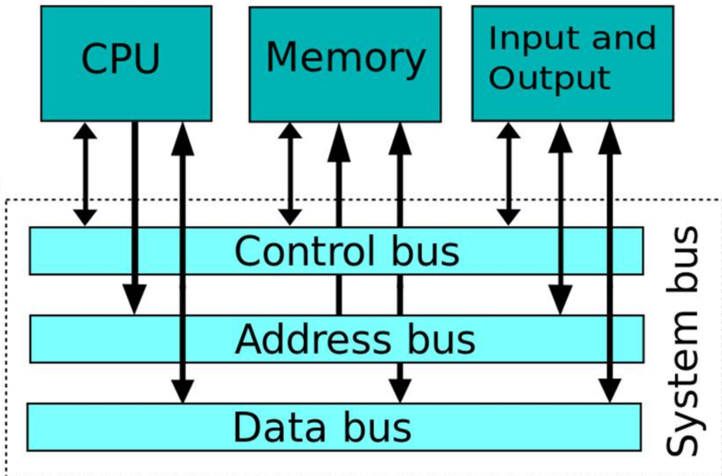
Data and Code boundaries can be moved at will
Code can modify itself. (dangerous!)
Very practical for general purpose computing

If additional CPUs are added (e.g. "quad core"
They communicate over the system bus.

Except for one gotcha: All data and instructions must flow over the single bus in sequence. This could be only half the speed of a Harvard machine.

In a multi-CPU system the system bus quickly Becomes a point of congestion. Each additional CPU brings practically no benefit if the System bus is already fully occupied.

Cache, if used, could be part of the CPU or part of the memory system.



15

<u>Von Neuman Architecture</u> (a.k.a. Princeton Architecture): The other popular way to fit memory into a system
Popular in many general-purpose CPUs including Raspberry Pi's ARM BROADCOM CPU and Windows x86 CPUs, Apple OS x

Everything connects via one "system bus." There is only <u>one memory system</u>, shared for all uses.
(Pure Von Neuman: one address space. I/O operations are *memory mapped*. Or, "not-so-pure" separate memory and I/O addresses via a "MEM" bit on the control bus.)

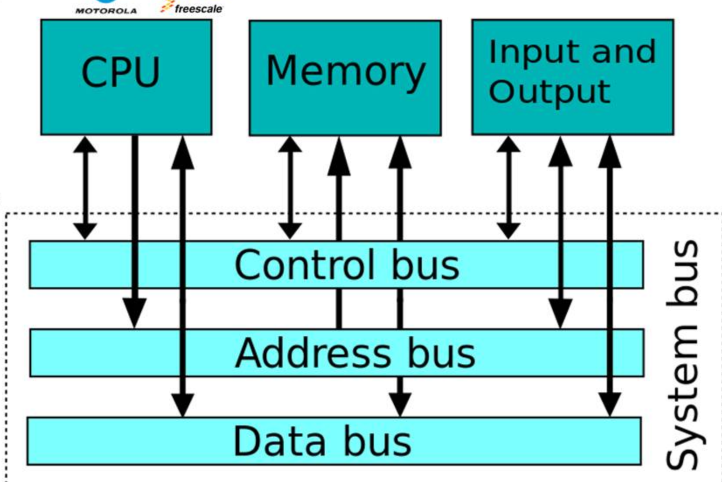
Data and Code boundaries can be moved at will
Code can modify itself. (dangerous!)
Very practical for general purpose computing

If additional CPUs are added (e.g. "quad core"
They communicate over the system bus.

Except for one gotcha: All data and instructions must flow over the single bus in sequence. This could be only half the speed of a Harvard machine.

In a multi-CPU system the system bus quickly Becomes a point of congestion. Each additional CPU brings practically no benefit if the System bus is already fully occupied.

Cache, if used, could be part of the CPU or part of the memory system.

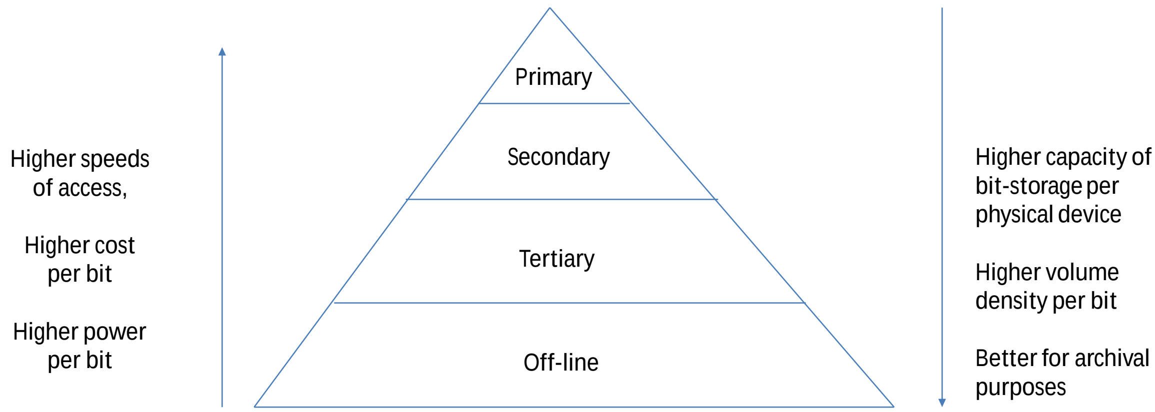


16

Digital Data Storage (from Wikipedia, "Computer Data Storage" Link)

Data storage has a hierarchy

Primary data storage—accessible to the CPU in word-wide addressable chunks via a bus
Secondary data storage—accessible to the CPU via an I/O port, always available (e.g., the hard drive)
Tertiary storage—accessible to the CPU via I/O, not always online (sleeps or jukebox selection, etc.)
Offline storage—requires human intervention, e.g. USB drive, SD Card



17

Data storage has a hierarchy—an example

Raspberry Pi 3 has 1 GB of LPDDR2 RAM" (Pi 4 has up to 4GB)—that is primary storage

Raspberry Pi 3, 4 accommodates a microSD card—that is technically off-line storage, but R-Pi uses it as secondary storage. (If you remove it, the system crashes.)

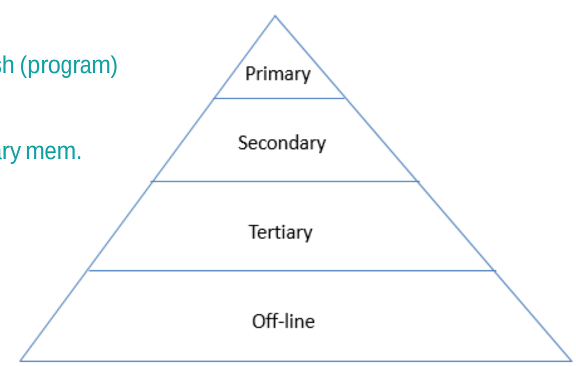
Raspberry Pi 3, 4 has Wi-Fi, one can configure Google-Drive, Dropbox, etc.—tertiary

Raspberry Pi 3, 4 has USB, one can plug in a USB drive—offline storage

Arduino Uno has 2 KB of SRAM (data memory) and 32 KB of Flash (program) storage—primary storage

Arduino Uno has up to 32 GB SD-card storage—used as secondary mem.

Arduino Uno has no built-in tertiary or offline memory.



18

Data storage has various characteristics

Volatile, or dynamic: Volatile memory loses the stored information within milliseconds (or less) of losing power. Dynamic memory retains information for several milliseconds after the removal of power. A refresh operation, which requires power and bus activity, restores the information in a fraction of the time during which power may be turned off. Dynamic memory needs to be treated as volatile memory in most contexts because an interruption of the utility power will cause information loss, but overall dynamic memory saves battery life considerably because it can be turned off most of the time and periodically refreshed on a rapid but low-duty-factor cycle.

Non-volatile: Non-volatile means it retains its stored information for long periods (years) even if the memory has no electric power.

Mutability (Writeability)
Read/Write—allows words of information to be individually overwritten at any time.
(write might take longer than read, but less than 10x the time of a read). E.g. system RAM

Fast read/slow write—Writing is possible but relatively time consuming.
Writing might have to occur in large chunks of data, say 1 kB at a time. E.g. flash memory

Read only—information is encoded into the memory during manufacturing and can never be changed without physically removing and replacing hardware. E.g. ROM BIOS chips in older PC's (ROM is becoming rare)

Accessibility & Granularity
Random access—any address, any time. (Needed for primary storage)
Sequential access—Several addresses (2, 4) or even large "sectors" of addresses (512, more) need to be read or written at once even if only one address is needed. (Common with disk drives, SD cards, etc.)

19