Course wide

Week 2 - Notes

*INTRODUCTION to Floating Point Numbers IS Covered →6→

* Hardware Primitives & Services.
→ introduction to O/S

*Operating System is software generally written in C

* BIOS = Basic Input Output System & IS Stored ON A ROM Chip, ROM = Read only MEMORY

*Drivers, ~~Others~~ O/S's, BIOS' are generally written in C or Assembly Code

* ALC = Arithmetic Logic unit Determins the Appropriate instructions for a given task

* CPU = Central Processing Unit fetches, decodes and then executes instructions that perform
- arithmetic, logic comparisons and other operations

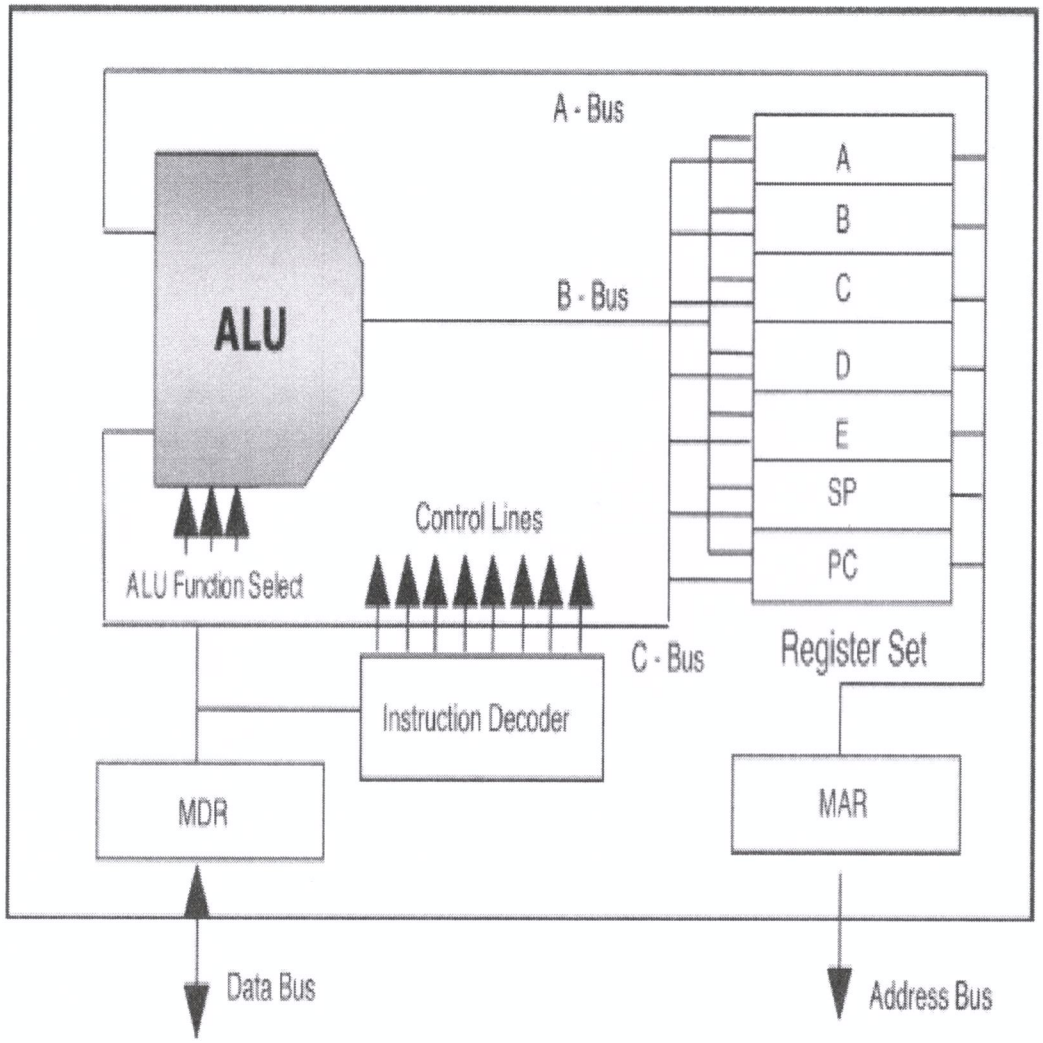
- ~~Other work~~
- logic comparison example
is number1 equal to number2
- other operations example
SKIP Next 50 ~~oper~~ instructions

ECC

* ~~ECC~~ = Error Correction Code is added to Server Ram and Can Correct Bad Bits of ~~Binary~~ Binary that has been Corupted

* MMU = Memory Management UNIT

CPU.

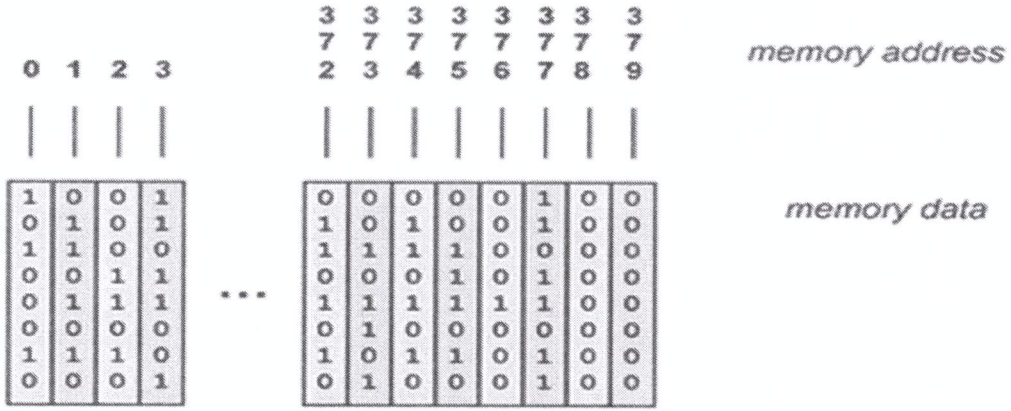


- ***Arithmetic Logic Unit*** **(ALU)** performs arithmetic and logic functions
- ***Registers*** - high speed 'scratch pad' to store data currently being processed
- ***Memory Buffer (Data) Register*** **(MBR, MDR)** – stores data just received from, or about to be written to memory
- ***Memory Access Register*** **(MAR)** – stores address of memory to be accessed next
- ***Program Counter*** **(PC)** – this register stores the address of the next instruction to be processed
- ***Stack Pointer*** **(SP)** – this register stores the address of the top of the "stack" which is used to process instructions

Memory

- Each byte of memory has an address
  - numbered sequentially
  - individual bits not addressable, just (usually) bytes

- An address length of N bits can express 2^N numbers $(0..2^N-1)$
  - so maximum size of memory limited by length of address

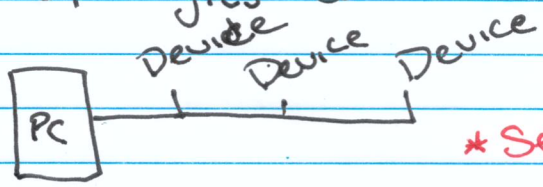


*Data links

Serial: ONE Bit of data sent on a single wire
Parallel: Multiple wires used to transfer multiple bits at the same time.

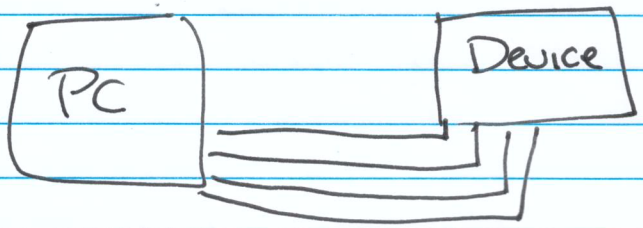
Common Topologies are:

Bus



* Serial

PtP (Point to Point)



* Parallel

*Serial is more common as it can travel further than Parallel and least expensive

* Bus: Shares ~~Bandwidth~~ Bandwidth between all attached devices, get's signal crosstalk on Long distance

* PtP: each device has dedicade data lines and Bandwidth

Hot - Swappable Can be connected or disconned while computer is ON

Warm - Swappable Can be connected/disconnected when Computer is in sleep mode

Cold - Swappable Can be conn/disconn when Comp is off
→ Otherwise you risk data loss
OR computer crash
OR dangerous short circuit
OR ~~Mix~~ Mix of all 3

Examples

Hot - Swap: SAS Harddrives, USB Devices

Warm - Swapp: Batteries? + Legacy connections

Cold - Swap: Hard drive, GPU, Memory

Floating Point Numbers

* Floating Point Representation uses Standard Scientific Notation Ideals

→ Binary Integers in a Computer are of fixed and limited Size and are unable to represent very small or very large ~~Numbers~~ Values

→ Floating Point Numbers or Scientific Notation allows an almost limitless range of small and large numbers to be represented

Examples:

$700.0 = 7.0 \times 10^{2}$ Move decimal Point 2 Places left

Very Large

$4,900,000,000.0 = 4.9 \times 10^{9}$ Move decimal Point 9 Places left

very small

$0.0000006 = 6 \times 10^{-7}$ Move decimal Point 7 Places Right

$$\text{Format} = a \times R^{e}$$

a = Mantissa
R = radix/Base ← Base = 2 for Binary
e = exponent ← can be Positive or Negative

Exponent ↓ Mantissa ↓

$0011\ \ 0000\ 0101 = 5 \times 2^{3} = 5 \times 8 = 40.0_{10}$ (2×2×2)

3 0 5

* Exponent is 4 Bit 2's Complement
* Mantissa 8 Bit UNSigned

* Exponent is 4 Bit 2's Complement
* Mantissa is 8 Bit UNsigned

Exponent ↓ Mantissa ↔

1111 0000 0110 $= 6 \times 2^{-1} = 6 \times 0.5 = 3.0_{10}$

-1 0 6

↑ Binary fraction

Tells us A NEG

take ONE's Compliemen

$= 0000_2 + 1 = 0001_2 = 1_{10}$

Binary Fractions

Binary fractions can also be represented:

Position Value: bp(binary power)	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5} etc
Decimal	0.5	0.25	0.125	0.0625	0.03125
Fractions	1/2	1/4	1/8	1/16	1/32

The following video is useful: Decimal Fractions to Binary Fractions

Conversion from a Decimal Fraction to a Binary Fraction

Multiply by 2 till enough digits are obtained, say 8, or a product is zero.

Example: 0.637_{10} to equivalent binary fraction.

Multiplication by 2	Result
0.637 x 2	**1**.274
0.274 x 2	**0**.548
0.548 x 2	**1**.096
0.096 x 2	**0**.192
0.192 x 2	**0**.384
0.384 x 2	**0**.768
0.768 x 2	**1**.536
0.536 x 2	**1**.072

Reading **down** the result column gives to 8 digits:

$\Rightarrow 0.637_{10}$ is aproximately equal 0.10100011_2

IEEE Standard 754 Floating Point Format

Single Precision - 32 bits

- 1 bit + OR − (1 = NEG 0 = Positive)
- 8 bit exponent - excess 127 format (note Excess or "bias" = 2^{8-1}-1= 128-1 =127)
- 23 bit mantissa

Double Precision- 64 bits

- 1 bit
- 11 bit exponent - excess 127 format Excess or "bias" = 2^{11-1} -1= 1024-1=1023
- 52 bit mantissa

Extended Precision - 80 bits

Only used internally by the computer arithmetic unit, ALU, to reduce rounding errors.

IEEE 754 Floating Point Format Example

Conversion from decimal to IEEE 754 format. Example: Convert 10.5_{10} to Single Precision Format

1. Convert to binary
   1. $10 = 1010_2$
   2. $0.5 = 0.10_2$
   3. $10.5 = 1010.10_2 \times 2^0$

10.5

Reassembled

2. Normalise so that we have 1. in front which gives $1.01010 \times 2^{+3}$. Note the 1. is not stored in the computer to increase precision. The 1. is put back during arithmetic operations.

3. Exponent is +3 in excess (or bias) 127 format
   $= 127_{10} + 3_{10} = 130_{10} = 1000\ 0010_2$

4. Mantissa = 01010000

5. Sign =0, +ve

+3 Exponent

6. IEEE = $0\ 1000\ 0010\ \ 0101\ 0000_2$ and rewrite in nibble
   $= 0100\ 0001\ 0010\ 1000\ 0000\ 0000\ 0000\ 0000_2$
   Then use hexadecimal notation:
   $= 4128\ 0000_{16}$

Padd out Mantissa in this case its 23 Bit

0 + Exponent + Mantissa

Positive Num

Conversion from IEEE 754 to Decimal Example

Convert the IEEE 754 Single Precision Floating Point Format:
$40C4\ 0000_{16}$ to Decimal

$40C4\ 0000_{16}$
$= 0100\ 0000\ 1100\ 0100\ 0000\ 0000\ 0000_2$, in nibbles

Split into sign, exponent, mantissa

$= 0\ 10000001\ 100\ 0100\ 0000\ 0000\ 0000\ 0000$

Sign =0, +ve

Exponent = $1000\ 0001_2$

$= 129_{10}$

Now subtract the "excess" or "bias " 127 from 129 :

$= 129_{10}-127_{10}=2_{10}$

Mantissa or Significand = 1.100 0100 0000 0000 0000 0000 , Note the **1.** is put back in

Result = $1.10001 \times 2^2 = 110.001 \times 2^0 = 6.125_{10}$

Reals - Errors

- Whilst Reals can represent a very wide range of numbers, there is a constant percentage error because of the distance between each number.

- Integers also have an error of 1/2 the least significant bit. This error is constant and becomes a smaller percentage as the number gets larger.

- Take for example a Real number in the decimal system with two digits: one for the exponent and one for the mantissa:

  i.e. Exponent is 0-9 and the mantissa is 0-9.
  Therefore, we can have numbers like:

  1×10^0, 1×10^2, 4×10^6, 9×10^9, etc.

Reals - Errors

Therefore, we can have numbers from:
0 to 9 x 10^9 or 0 to 9,000,000,000
The issue is that there are only 100 different values:

	Error +/-
0 1 2 3 4 5 6 7 8 9	0.5
10 20 30 40 50 60 70 80 90	5
100 200 300 400 500 600 700 800 900	50
1000 2000,3000,4000,5000,6000,7000,8000,9000	500
10,000 ... 90,000	5000
100,000 ... 900,000	50,000
1,000,000 ... 9,000,000	500,000
10,000,000 ... 90,000,000	5,000,000
100,000,000 ... 900,000,000	50,000,000
1000,000,000 ... 9,000,000,000	500,000,000

Note: There is a constant percentage error.

Rounding Off Errors when Floating Point Numbers represent 'Reals'

Arithmetic example:

A decimal example: 543.57 + 1.862 =
Assume the computer handles 4 bit decimal numbers.

Normalised: $0.5436 \times 10^3 + 0.1862 \times 10^1$, rounding 5->6

To add the two numbers we must make the exponents equal:

$0.5436 \times 10^3 \Rightarrow 0.5436 \times 10^3$

$0.1862 \times 10^1 \Rightarrow 0.0019 \times 10^3$, Rounding the error.

Sum $= 0.5455 \times 10^3$

$= 545.5$

By hand, the result is 545.432

The error will lead to further errors when used in further addition operations. This illustrates the concept **"Rounding Error"** when we try to represent "Reals" with a fix number of bits in a computer.

Truncation

In some calculations we use "truncation" instead of "rounding" when dealing with say the mantissa: Using the above example:

To add the two numbers we must make the exponents equal:

$0.5435 \times 10^3 \Rightarrow 0.5435 \times 10^3$, **Truncating the error**

$0.1862 \times 10^1 \Rightarrow 0.0018 \times 10^3$, **Truncating the error**

$$\text{Sum} = 0.5453 \times 10^3$$

$$= 545.3$$

By hand, the result is 545.432

- Note that **"truncation"** has a higher error than **"rounding"** for this example.
- This video is an excellent summary of this section on "floating point numbers"