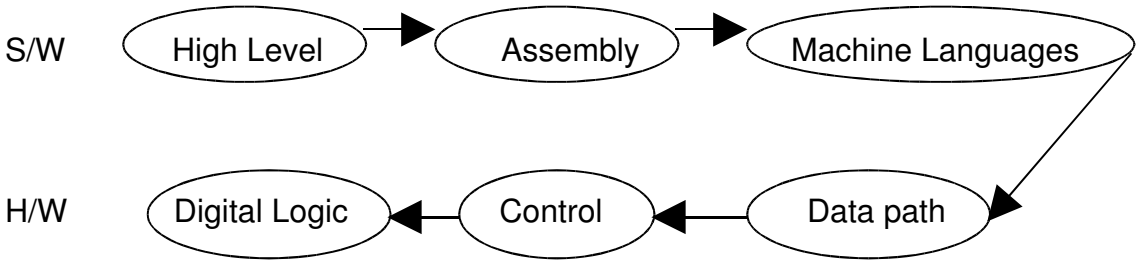
# Introduction…

The journey from software to hardware:




## MIPS

- A company designing abstract processor architectures – not the actual processors.
- Their method is called RISC
- Some processors using MIPS are SPARC (Prism Lab UNIX systems), PowerPC, Nintendo and Sony

We will use SPIM which emulates the MIPS assembly language.

Main computer components (see the motherboard diagram) are

- MEMORY (DRAM)
- CPU
- BUS

## Memory (Also see professor Roumani's notes)

- Memory is always in form of a one dimensional array
- Call number of the cells are not actually written. They are just our way of identifying them.
- Each cell (1 byte) has 8 sub-cells (bits)

  1 B = 8 b
- Bits are numbered from RIGHT → LEFT (See the diagram below)

  Bytes:

  Kilobyte: 1 kB = 1024 B = $2^{10}$ B

  Megabyte: 1 MB = 1024 kB = $2^{10}$ kB = $2^{20}$ B

  Gigabyte: 1 GB = 1024 MB = $2^{10}$ MB = $2^{30}$ B
- In Java:

  byte = 1 byte

short = char = 2 bytes

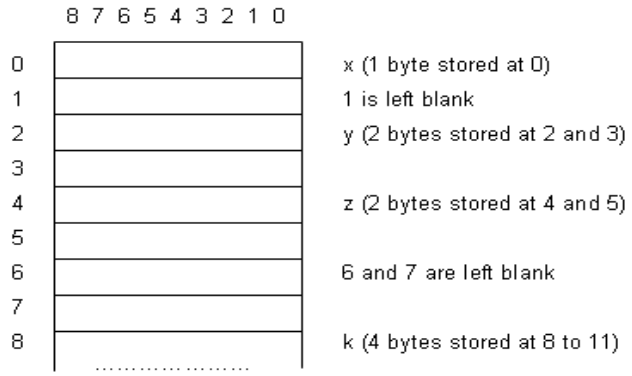
int = 4 bytes

long = 8 bytes

- To store types larger than 1 must find to consecutive spots and store the digits in order:
  If we store most significant digits at lower address (i.e. first) it is called **Big Endian** (Big-End-First) otherwise it is called **Little Endian**.
  e.g. Intel is Little Endian while Motorola (Apple) is Big Endian
- Notion of data types does not exist in memory.
- *Rule:* Store an n-byte object at an address divisible by n:

E.g. This is how we store the following (Java) variables in the memory block below:

byte x; short y; char z; int k;




**RAM** (Random-Access Memory):

- Has the same latency at all addresses.
- DRAM → Dynamic RAM → Latency 10-50 ns (Fast)
- SRAM → Static RAM → Latency 0.5-1 ns (Even Faster)
- So why do we use DRAM if SRAM is faster?
  Because of heat!
  SRAM uses a lot of heat and the material will burn.

  To store bits SRAM uses switches while DRAM uses charged capacitors.

## Disk

Even though it's not RAM we consider it one because the rotation is very fast.

Note: Latency is per byte and this is a BIG problem.

Why do we use Disk if it's slower than RAM?

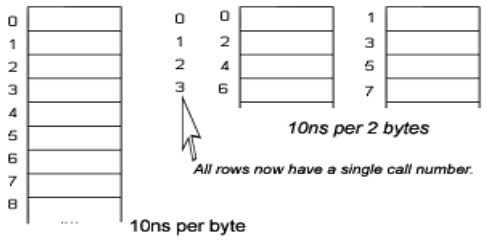
- Because loss of power will erase RAM but will not affect the disk.

## Parallel Processing

If we store cells parallel then we can get more bytes for the same latency (e.g. if we made 2 parallel columns we can get 2 bytes for the same latency as for 1 byte.)

So why don't we make more columns?

- Actually we do. This is how different types of DRAM (e.g. DDR, SDRAM, etc.) are different! The newer ones have more columns.




Different storage types (from top to bottom becomes faster but generates more heat):

| Type | Speed | Use |
|---|---|---|
| Disk | ~ 1ms | Long Term Data Storage |
| DRAM | ~ 10ns | Short Term Data Storage (Current Data) |
| SRAM | ~ 1ns | Cache |
| Register | ~ 0.1ns | CPU |

## CPU

Cycle = Duration of time for 1 operation or between two ticks of the clock. (sec)

1 / Cycle = Clock Rate = Clock Frequency (1/sec = Hertz = Hz)

* Clock Rate is usually measured in MHz = $10^6$ Hz or GHz = $10^9$ Hz

E.g. for a CPU running at 500 MHz:

Cycle = 1 / 500 * $10^6$ sec = 2 ns

→ Each "chunk" of work is done in 1 cycle. (In example above 2 ns)

## Execution of a Program in Computer:

- Load program from Disk to RAM
- Find the chunk of data in RAM and store the address in register PC in CPU.
- Store instruction in register IR. (In MIPS instructions are 32 bits.)

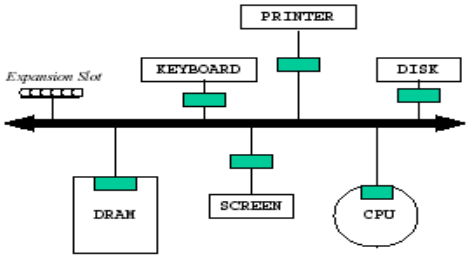
- Fetch Cycle: BIU (Bus Interface Unit) will send the data in PC to IR
  i.e. It'll send the address or data to instructions.
- Decode Cycle: Decode the instructions. At this time the CPU will go to sleep.
- Execute: Finally follow the instructions…
  * Last instruction will bring the O/S back.

## BUS

There are more devices than CPU and the DRAM.
These devices sometimes communicate directly, not just through the CPU.

If we wanted to attach n devices by wire we would need n (n-1)/2 wires. This is a quadratic problem and for every new device we need to add n more wires!
But we don't see that many wires. So how are they connected?

We use the BUS to connect devices. Each device is connected to the BUS and has a unique port number. There is a BIU before each device.




There is a problem comparing to wires: Only two devices can connect at a time.
Historically, even though the devices did not communicate that often sometimes the CPU was waiting forever!

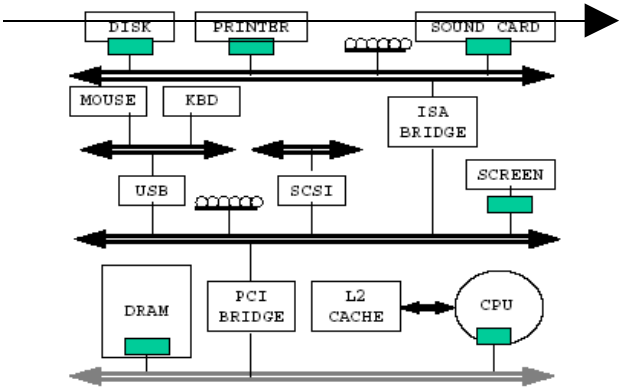
A separate BUS (called the "local BUS" e.g. Intel's "Chipset") was used only between the CPU and the DRAM. This BUS was also connected to the main BUS.

Later with all the graphic programs the screen had to refresh more quickly and thus the CPU needed to communicate with the screen often. That's when they added another BUS called the Video BUS (e.g. "AGP").

Later another BUS was used for slower devices called the USB BUS.

**Local BUS → Video BUS → BUS → USB**

**Slower**




What is limiting the speed of these? - The Distance

Currently the speed is between 133 to 500 MHz. They make them smaller and closer together for faster speed.

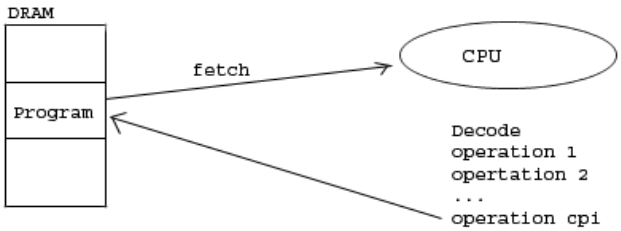
## Performance

The time required to run a program depends on:

DRAM Access Time

and CPU Time:

- CPI (Cycles per Instruction)
- CPU Clock Rate
- n = number of instructions




DRAM Access time depends on too many factors. But we can calculate CPU time:

**CPU Time = ( n x CPI ) / Clock Rate**

But each instruction has its own value for CPI.

**CPU time = $[(CPI)_1 + (CPI)_2 + \ldots (CPI)_n]$ / Clock Rate**

**= n * $(CPI)_{avg}$ / Clock Rate**

E.g. We can find the CPU time for a program with 3 instructions having a CPI of 4,16 and 1 that is running on a 500 MHz CPU:

The average CPI for the program is:

$$(CPI)_{avg} = ( 4 + 16 + 1 ) / 3 = 21 / 3 = 7$$

And the CPU time for the program is:

$$\text{CPU Time} = 3 * 7 / (500 * 10^6) = 42 \text{ ns}$$

**To reduce the CPU Time we can:**

a. Make the CPU faster: This is a hardware/engineering issue
b. Lower the CPI: Use simple instructions
c. Reduce n : Use very simple instruction sets

CISC's Solution (c):

- Let's make instructions complex and reduce n:

  Increase CPI → Reduce n

- Successful solution if done by human (hand-coded).

RISC's Solution (b):

- Let's use more simple instructions.

  Increase n → Reduce CPI

- Good solution for compiler.
- Better solution in general since most programs are written with high-level languages.