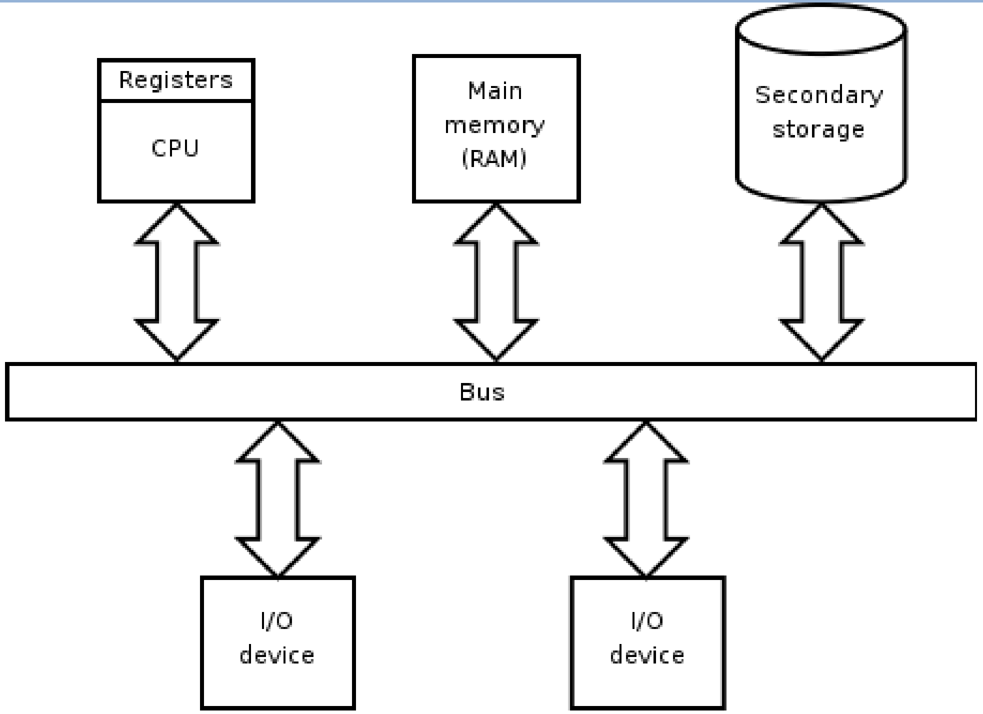
Operating Systems

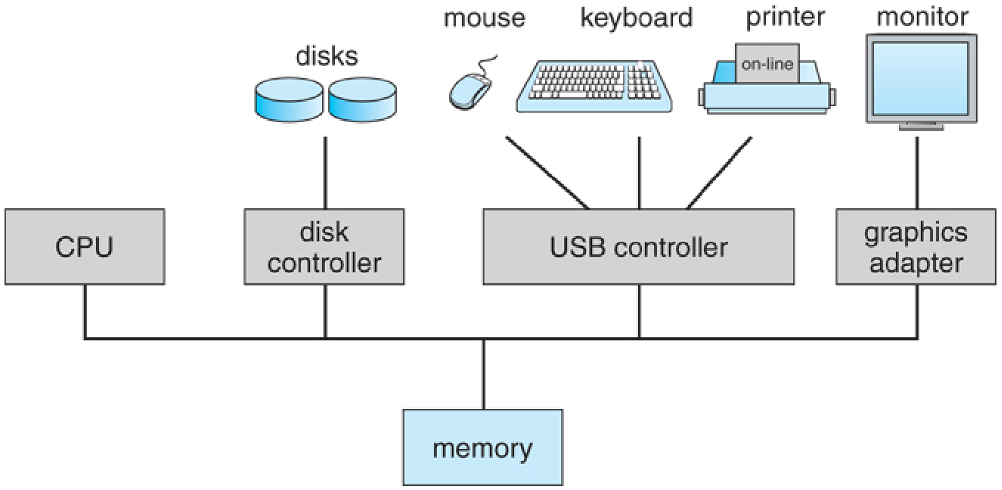
Eric Lo

2 Modern Computer Architecture

Overview, Glossary, and Revision

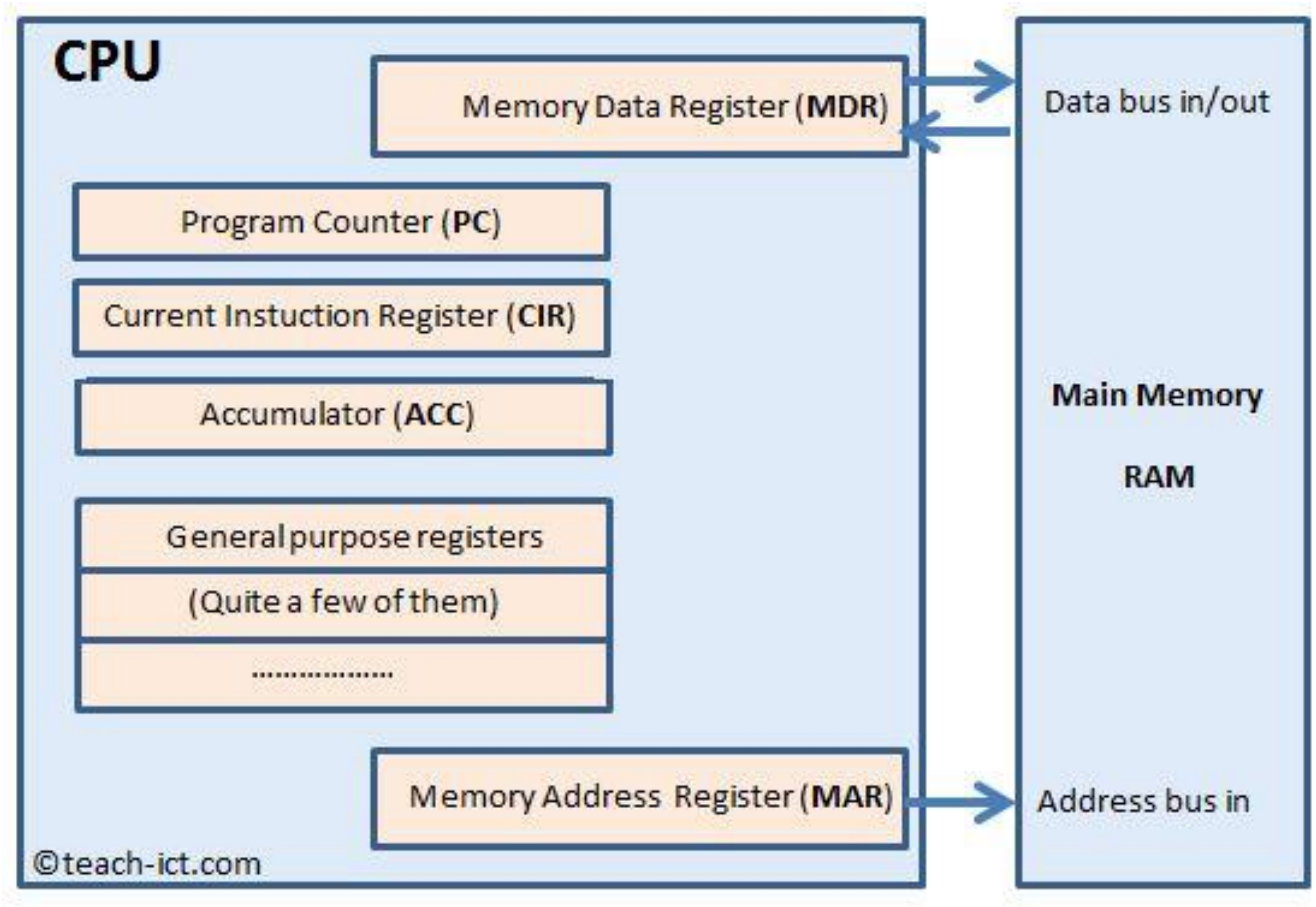
Basic Computer Architecture





http://faculty.ycp.edu/~dhovemey/spring2006/cs101/lecture/lecture1.html
https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/1_Introduction.html

Inside a CPU -- Registers

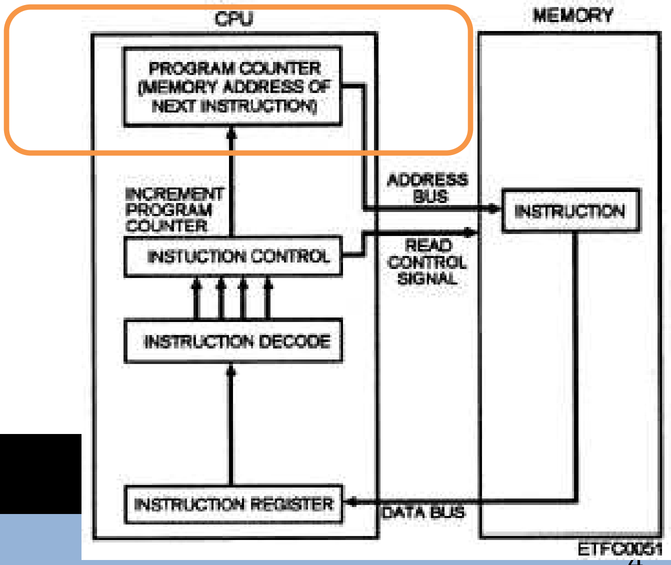


Ref: Teach ICT

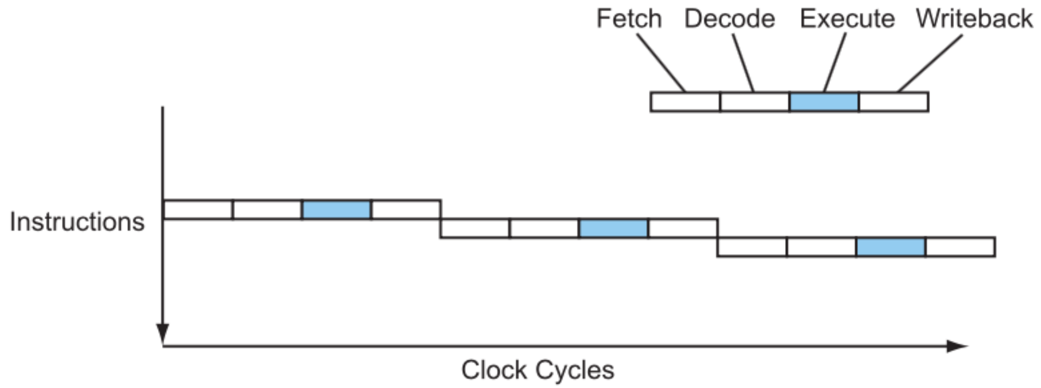
Inside a CPU

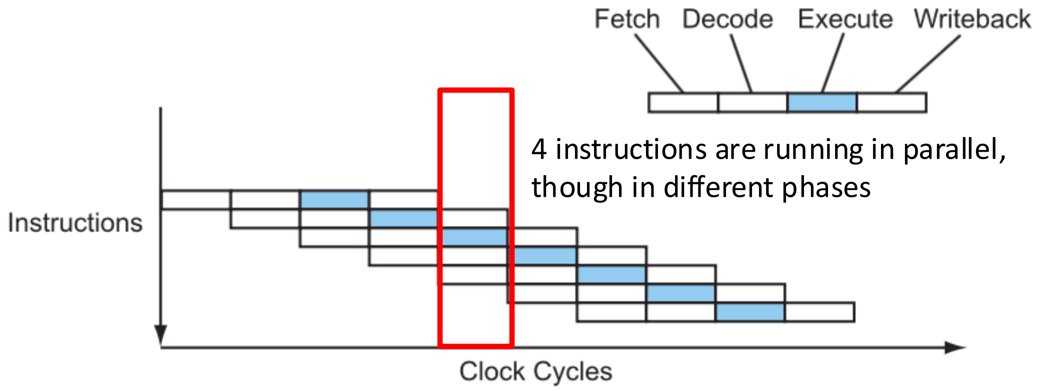
- **Program Counter**:
  - A **register** that contains the memory address of the next **instruction**
- **Instruction** Cycle (Fetch-Decode-Execute-MemoryAccess-WriteBack)
  - "Fetch": fetch the next **instruction** from the memory
  - "Decode": prepares various registers in readiness of the next step.
    - E.g., instruction: ADD `a, b` will
      - » Load content of a to register 1
      - » Load content of b to register 2
  - "Execute": carry out the computation / compute the memory address
  - "MemoryAccess": read/write from/to memory
    - For LOAD/STORE instruction
  - "WriteBack": write back results to the register

CPU speed = CPU clock rate
E.g., 1.8 GHz can perform 1,800,000,000 **clock** cycles per second



Sequential vs Scalar (Pipeline)





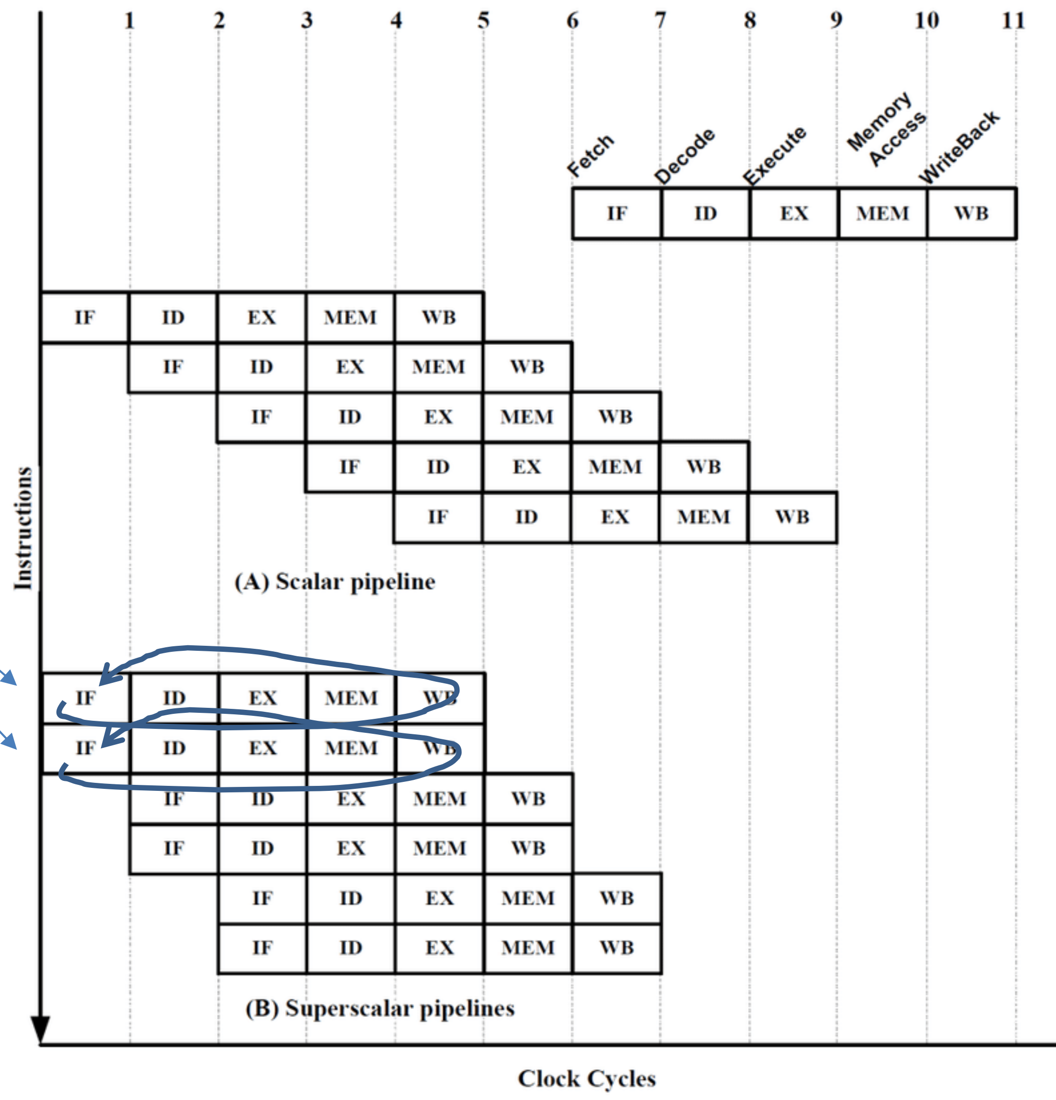
4 instructions are running in parallel, though in different phases

Ref.lighterra.com; the Memory Access Phase is missing in the figure because sequential processors are too old, no MA phase by that time

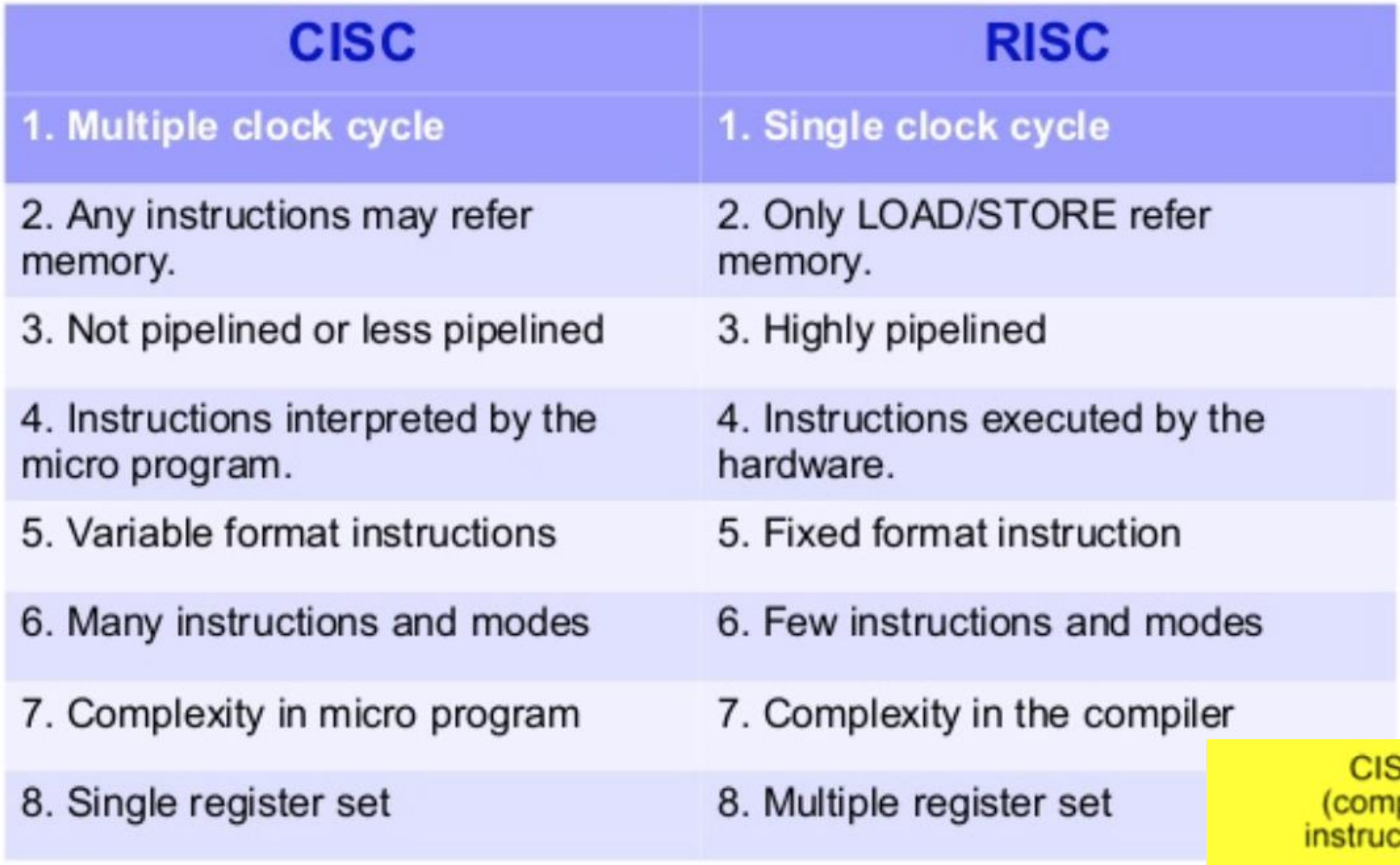
Scalar vs Superscalar

- Scalar = Pipelining
- Super-scalar = instruction-level parallelism

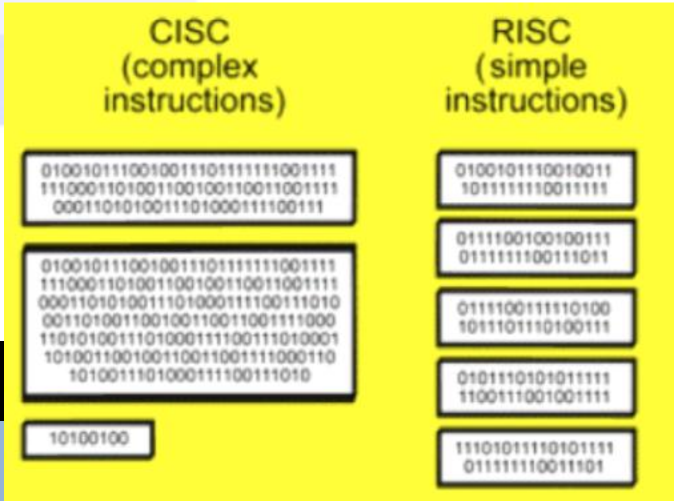
  = multiple cycles in parallel



Processor Design



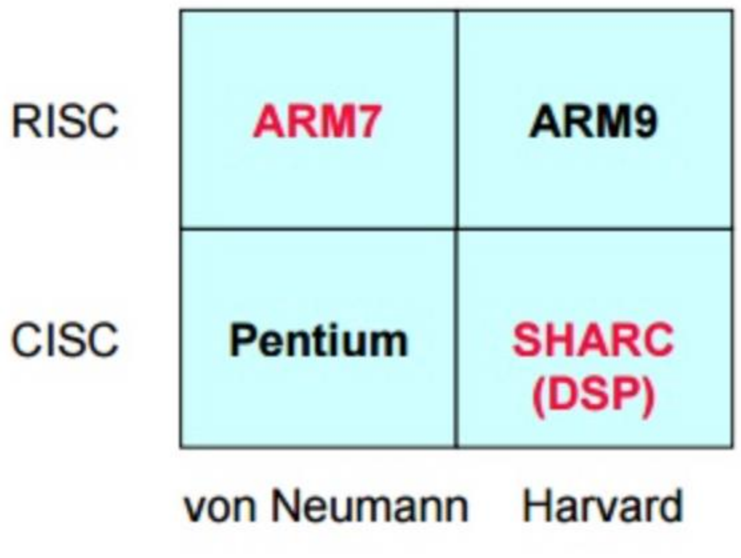
CISC	RISC
1. Multiple clock cycle	1. Single clock cycle
2. Any instructions may refer memory.	2. Only LOAD/STORE refer memory.
3. Not pipelined or less pipelined	3. Highly pipelined
4. Instructions interpreted by the micro program.	4. Instructions executed by the hardware.
5. Variable format instructions	5. Fixed format instruction
6. Many instructions and modes	6. Few instructions and modes
7. Complexity in micro program	7. Complexity in the compiler
8. Single register set	8. Multiple register set



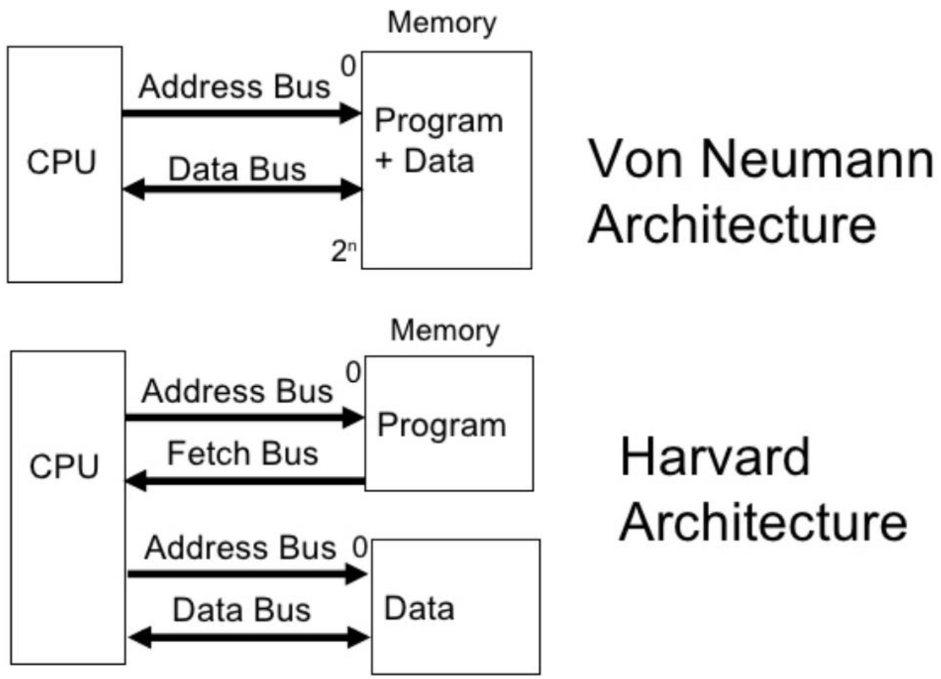
RISC vs CISC. Which one is better? Why?
Which one dominates the market? Why?

http://www.markedbyteachers.com

Intel



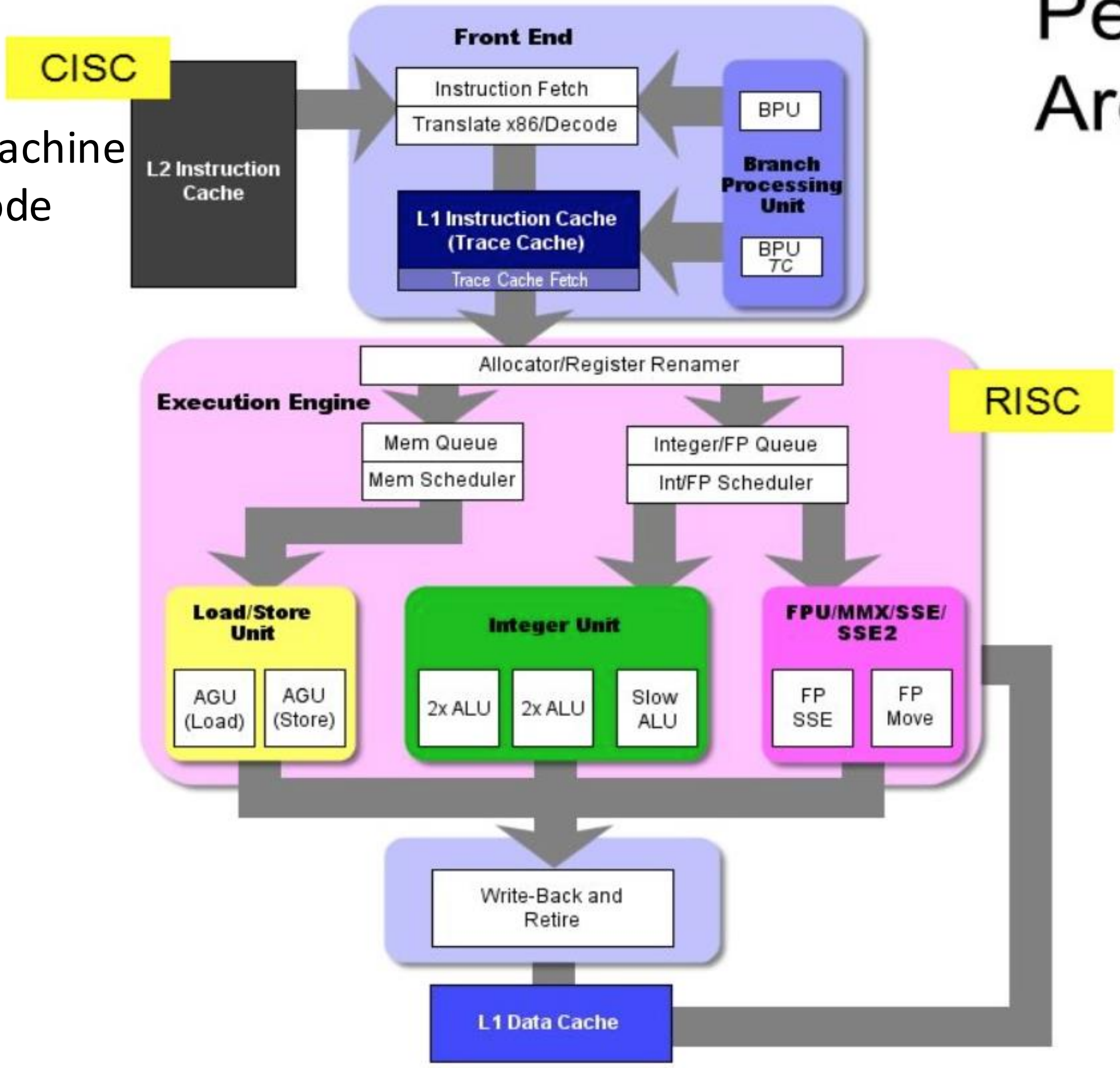
	von Neumann	Harvard
RISC	ARM7	ARM9
CISC	Pentium	SHARC (DSP)



Intel’s Dilemma

- Now almost all compliers generate CISC ISA code

machine code

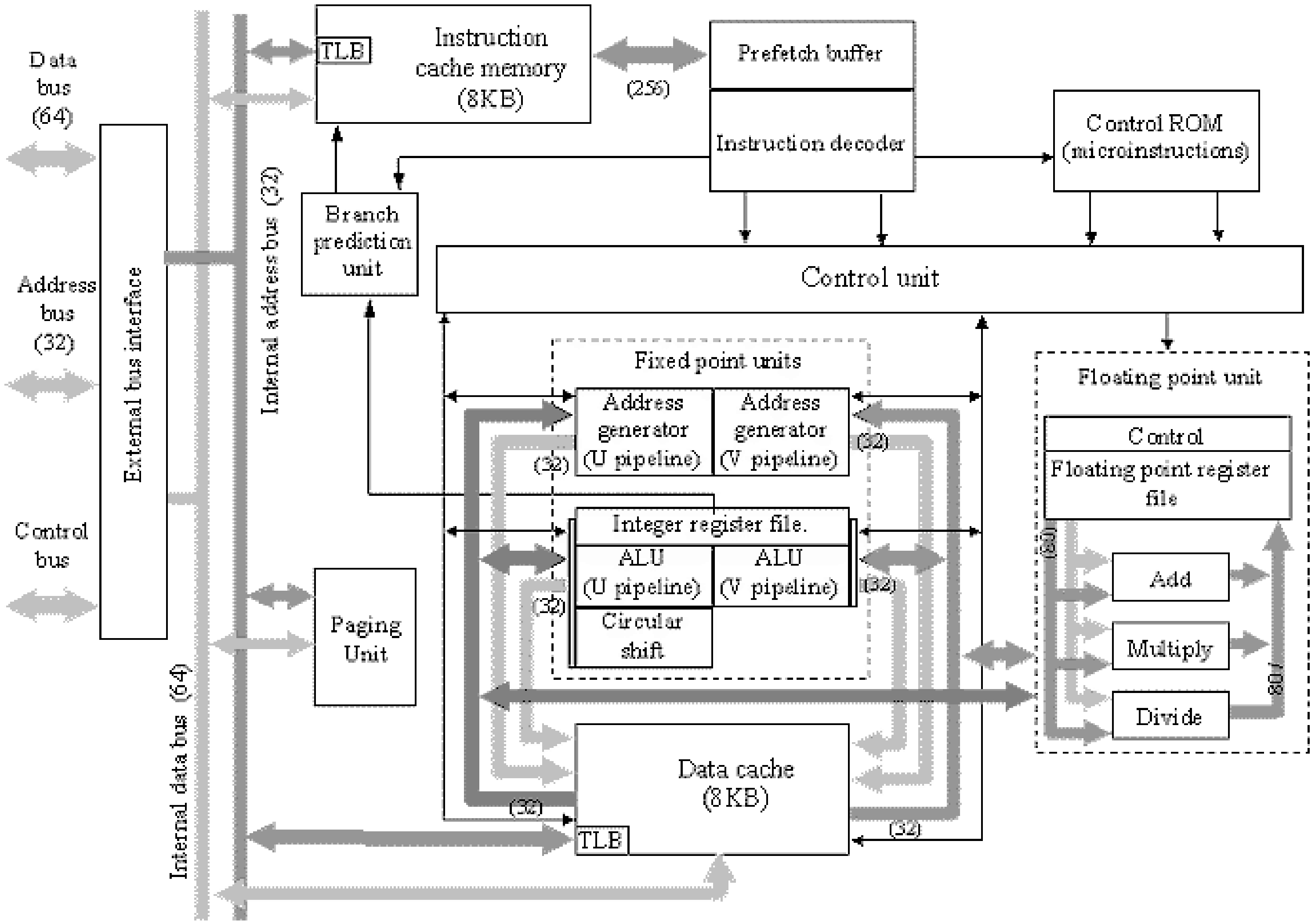


Pentium Architecture

Decode: CSIC -> RISC

Execute RISC code

More inside a modern CPU



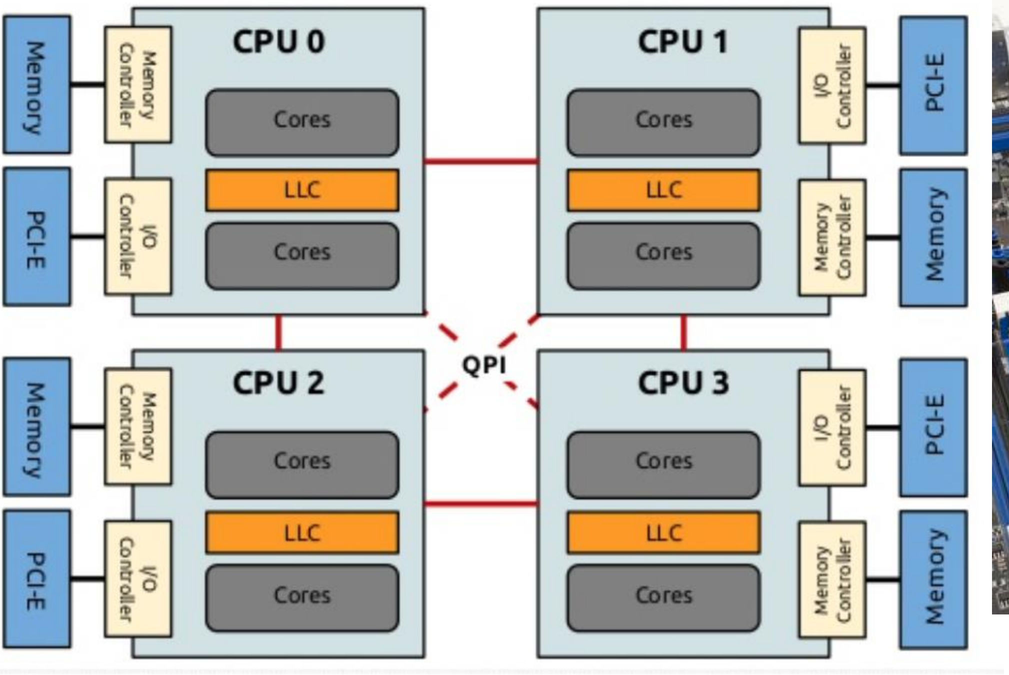
edux.pjwstk.edu.pl

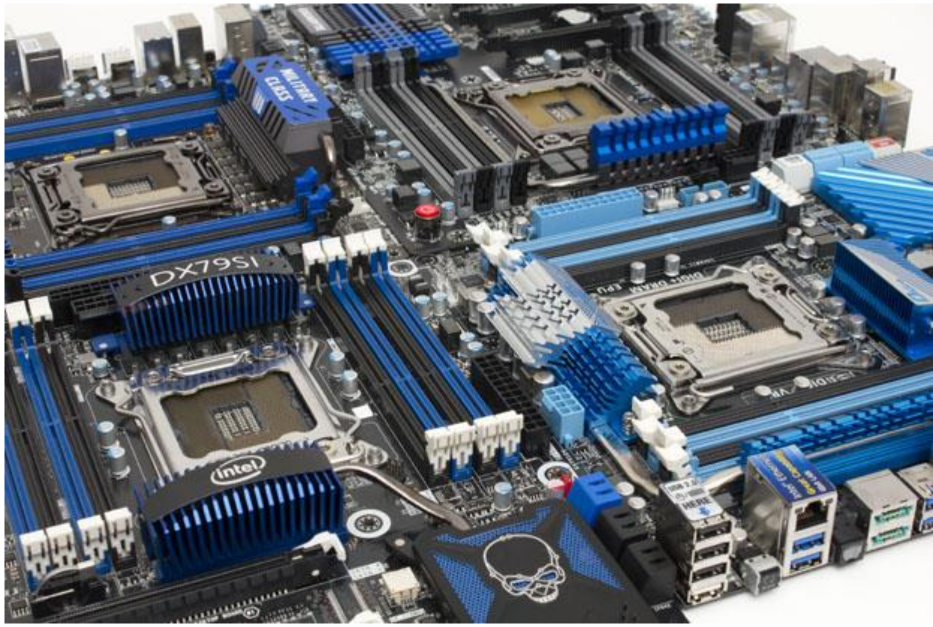
More inside a modern CPU

- With speculative execution
  - Instructions executed != Instructions retired
- OoO execution (More instruction-level parallelism)
  - Examines a sliding window of consecutive instructions
    - The “instruction window”
  - ”Ready” instructions that don’t (or no longer) depend on any former instructions could be executed first
  - https://en.wikipedia.org/wiki/Out-of-order_execution
- Can the compiler do the job?
  - Complier doesn’t have the runtime information
    - E.g., it can’t pick ahead if the next instruction is a branch

Multi-core + NUMA

- Even more cycles in parallel

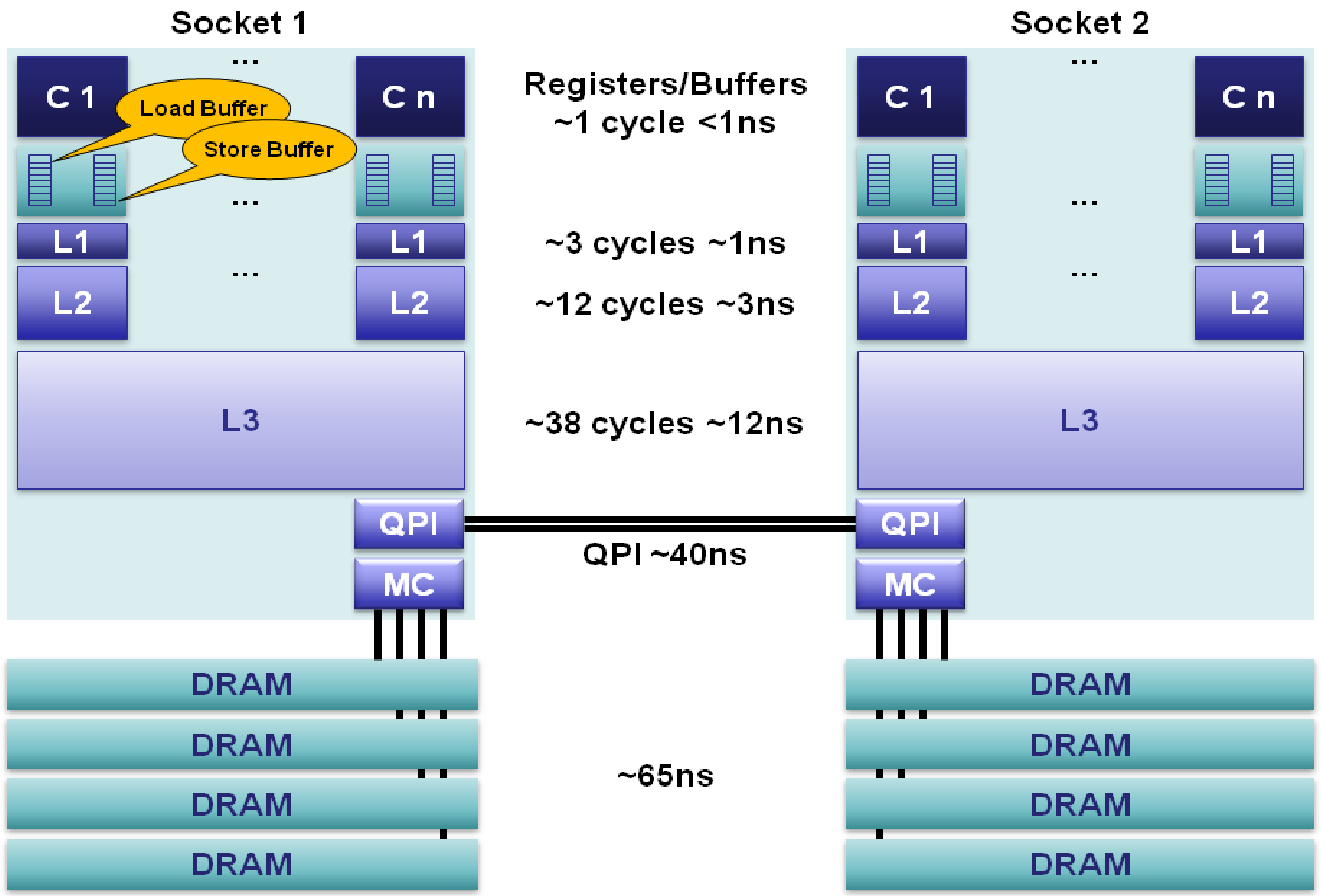




Intel Sandy Bridge

Ref: Paul Sim

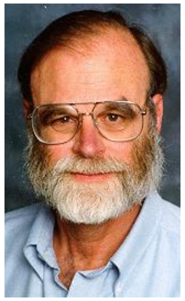
Memory Hierarchy



Ref. DZone

Why we have to know all these?

- Random access vs. sequential access time on HDD
- OS and all other systems **had** been optimizing to reduce random access on HDD
- "**Tape** is Dead, **Disk is Tape**, Flash is **Disk**, **RAM Locality is King**."



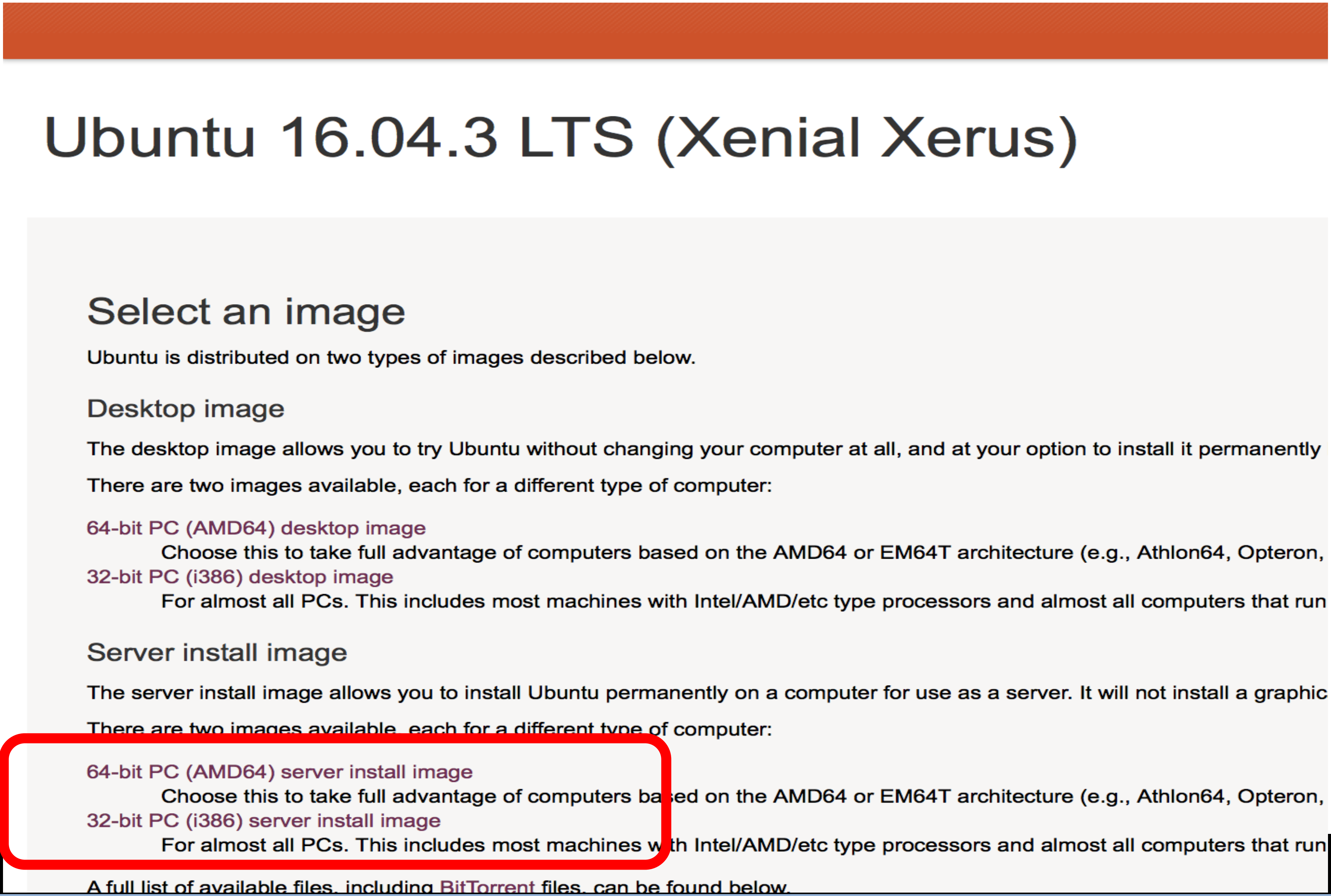
- What about
  - **Random access vs. sequential access time on RAM**?
  - Random access hurts CPU's speculative execution
    - => branch mis-prediction => expensive!
    - How to achieve "if" without using "if"?

Wait… how are these hardware things related to OS?

- Kernel is the piece of of software that directly deals with the hardware
- OS/Kernel programmers should arguably be writing the most efficient programs on earth
  - Efficient on memory usage
    - Remember PDP-7 only had 9kB RAM
  - Efficient on speed
    - If kernel programs don't care about speed, who cares more?
  - Different implementations for different architectures!
    - E.g., Intel Xeon's cache is **inclusive** whereas AMD Opteron's cache is **non-inclusive**
      - When reading X from memory,
        - » Inclusive: X will have a copy from L1 to L3
        - » Non-inclusive: X will read X into L1 directly (L2 and L3 have no copies)
    - Kernel programmers need to know this when designing cache replacement policies
    - That also explains why…

Different implementations are needed for different architectures

•



Ubuntu 16.04.3 LTS (Xenial Xerus)

Select an image

Ubuntu is distributed on two types of images described below.

Desktop image

The desktop image allows you to try Ubuntu without changing your computer at all, and at your option to install it permanently

There are two images available, each for a different type of computer:

64-bit PC (AMD64) desktop image
Choose this to take full advantage of computers based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron,
32-bit PC (i386) desktop image
For almost all PCs. This includes most machines with Intel/AMD/etc type processors and almost all computers that run

Server install image

The server install image allows you to install Ubuntu permanently on a computer for use as a server. It will not install a graphic

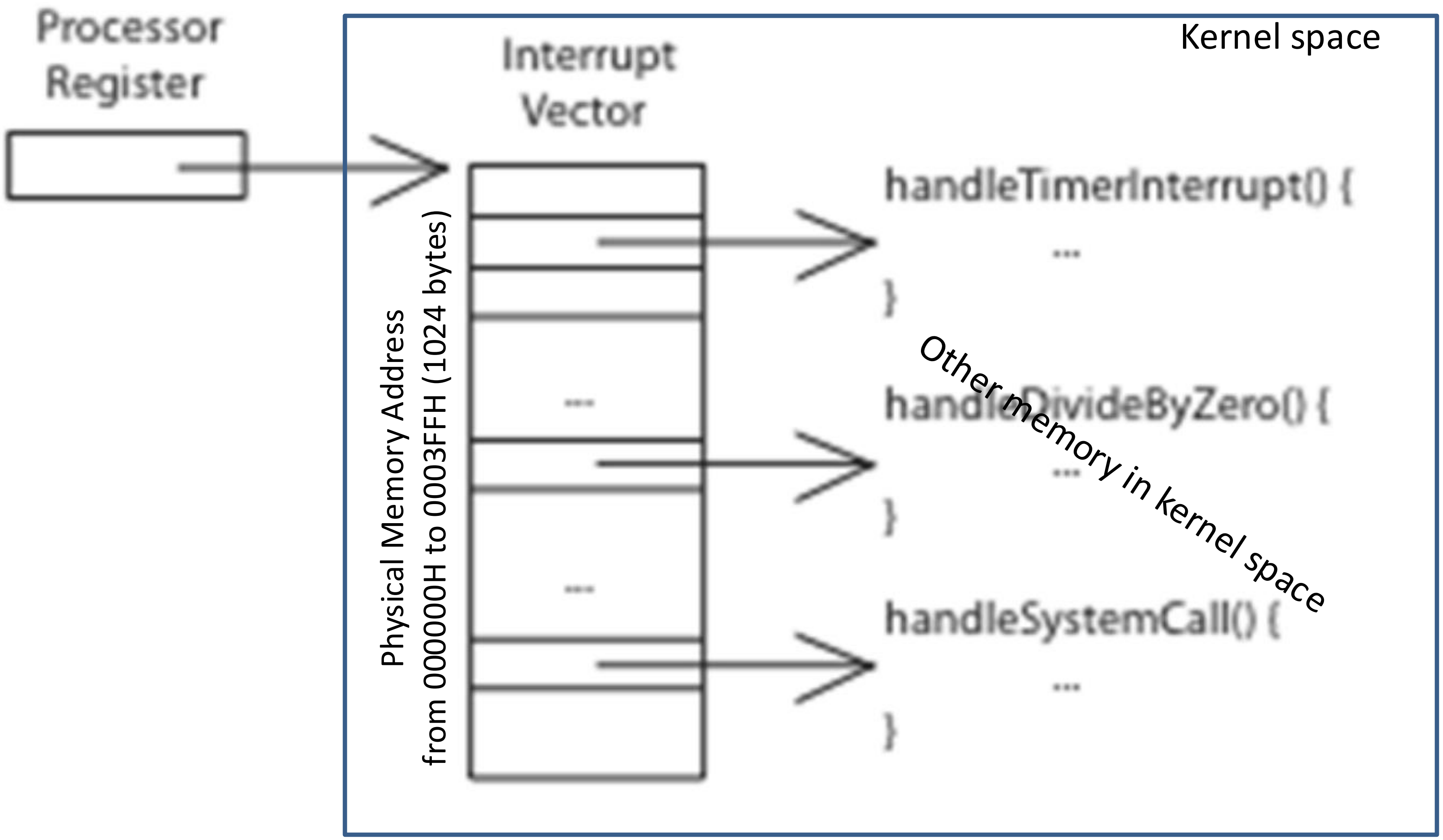
There are two images available, each for a different type of computer:

64-bit PC (AMD64) server install image
Choose this to take full advantage of computers based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron,
32-bit PC (i386) server install image
For almost all PCs. This includes most machines with Intel/AMD/etc type processors and almost all computers that run

A full list of available files, including BitTorrent files, can be found below.

CPU’s Privileged Instructions

- Some instructions just add two values
- Some instructions are **privileged**
  - E.g., set the segmentation boundary of the memory
- CPU has a 1-bit register to check if currently in **user-mode** or **kernel-mode**
  - If (user-mode & this-instruction-is-privileged) then
    - generate an “insufficient privilege access” **exception**
- On an exception / a hardware interrupt
  - CPU will go to a hardcoded memory address to lookup the corresponding handler



T.Anderson book

Hacking?

- So, a malicious user writes code to access other memory region through using some privileged instruction directly?
  - Any normal program must run on top of your OS and your OS won't permit your program to set that bit
  - So the most powerful attack is that you gain **physical access** to a machine and insert a boot device and reboot to your own OS…
    - But this is not the kind of security we concern

System Calls

- A benign program that wants to do something low-level?
  - Through **system calls**
    - written by OS developers
    - exposed for anyone to write program on

```
int a_sys_call() {
    1-bit register = kernel-mode;
    … access the kernel memory
    1-bit register = user-mode;
}
```