ARHITECTURA SISTEMELOR DE CALCUL - CURS 0x0A

PERFORMANȚA SISTEMELOR DE CALCUL

Cristian Rusu

DATA TRECUTĂ

- **sisteme multi-procesor**
- **ierarhia memoriei**
- **caching**

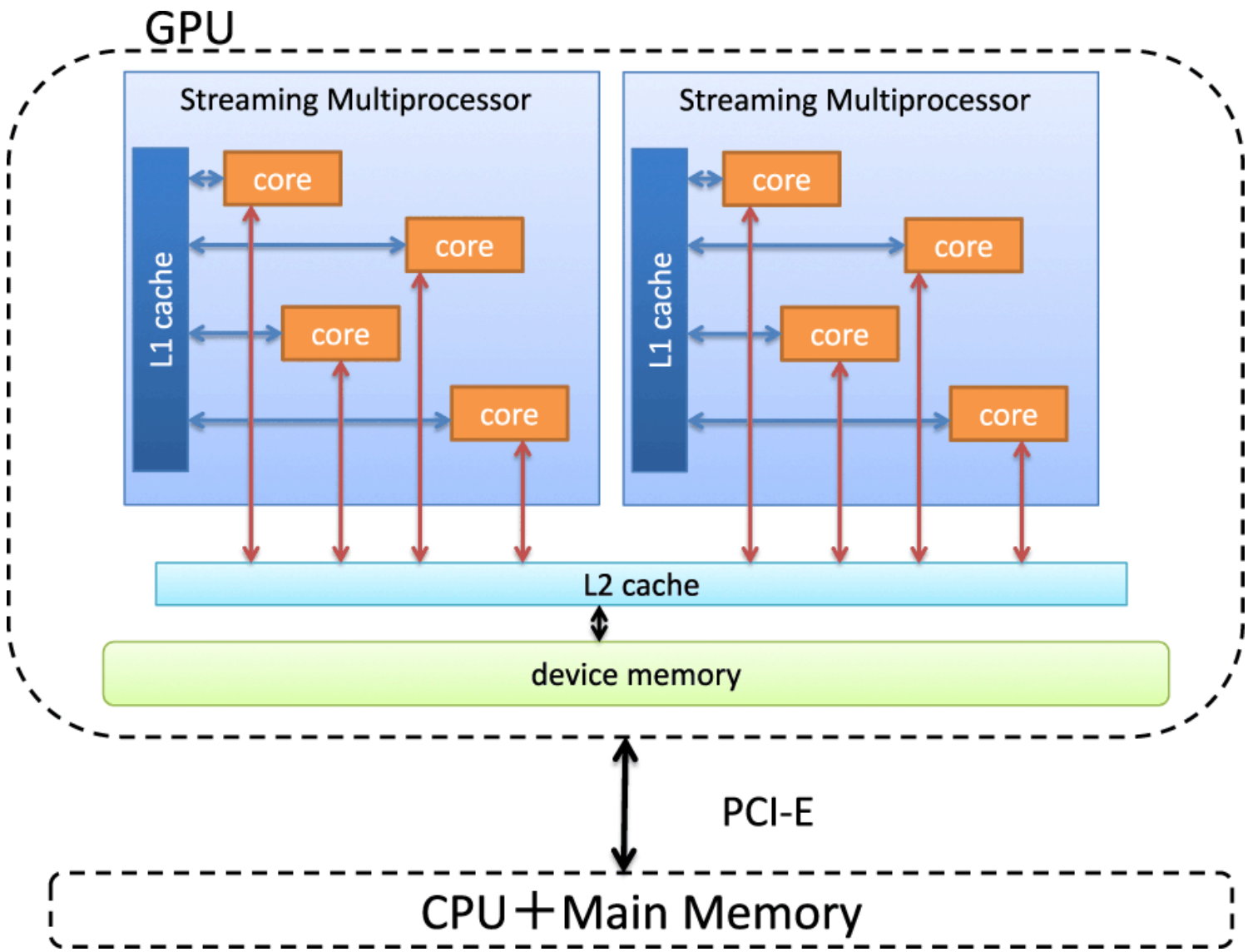
CUPRINS

- GPU
- ce este “performanța”
- cum măsurăm performanța sistemelor de calcul
- performanța CPU
- performanța limbajelor de programare
- CISC vs. RISC
- consum de energie
- la final, un demo interesant (sper)

GPU

- **este o unitate separată de calcul**
- **este conectată la un BUS de comunicare**
- **complementează capacitatea de calcul a unui CPU**
- **conțin un număr mare unități de procesare separate**
  - perfecte pentru procesarea (uniformă) unui bloc mare de date
- **sunt specializați pe un anumit set de instrucțiuni**
  - în general, nu pot realiza tot ce poate un CPU
  - dar ce pot executa, executa mult mai repede
    - SIMD/MIMD
    - hardware multithreading
    - Instruction Level Parallelism (ILP)
- **are propria sa memorie (on chip)**
  - este comparabilă cu cea a CPU (dar de obicei mai mică)
  - ierarhizare memoriei este mai simplă
  - timpii de acces nu sunt cea mai importantă caracteristică
  - bandwidth-ul (lățimea benzii de acces) este mai importantă

GPU



https://www.computer.org/csdl/journal/td/2017/09/07869389/13rRUytWF98

GPU

- **excelent pentru clasa de probleme “embarrassingly parallel”**
  - operații matrice-vector, matrice-matrice
    - calculul transformatei Fourier
  - procesarea imaginilor
  - convolutional neural networks (CNN) pentru Machine Learning
  - căutări exhaustive (brute force search)
    - căutarea hyperparametrilor
  - crypto mining
  - integrare numerică
  - simulări fizice cu scenarii diferite (condiții inițiale diferite)
  - raytracing

https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

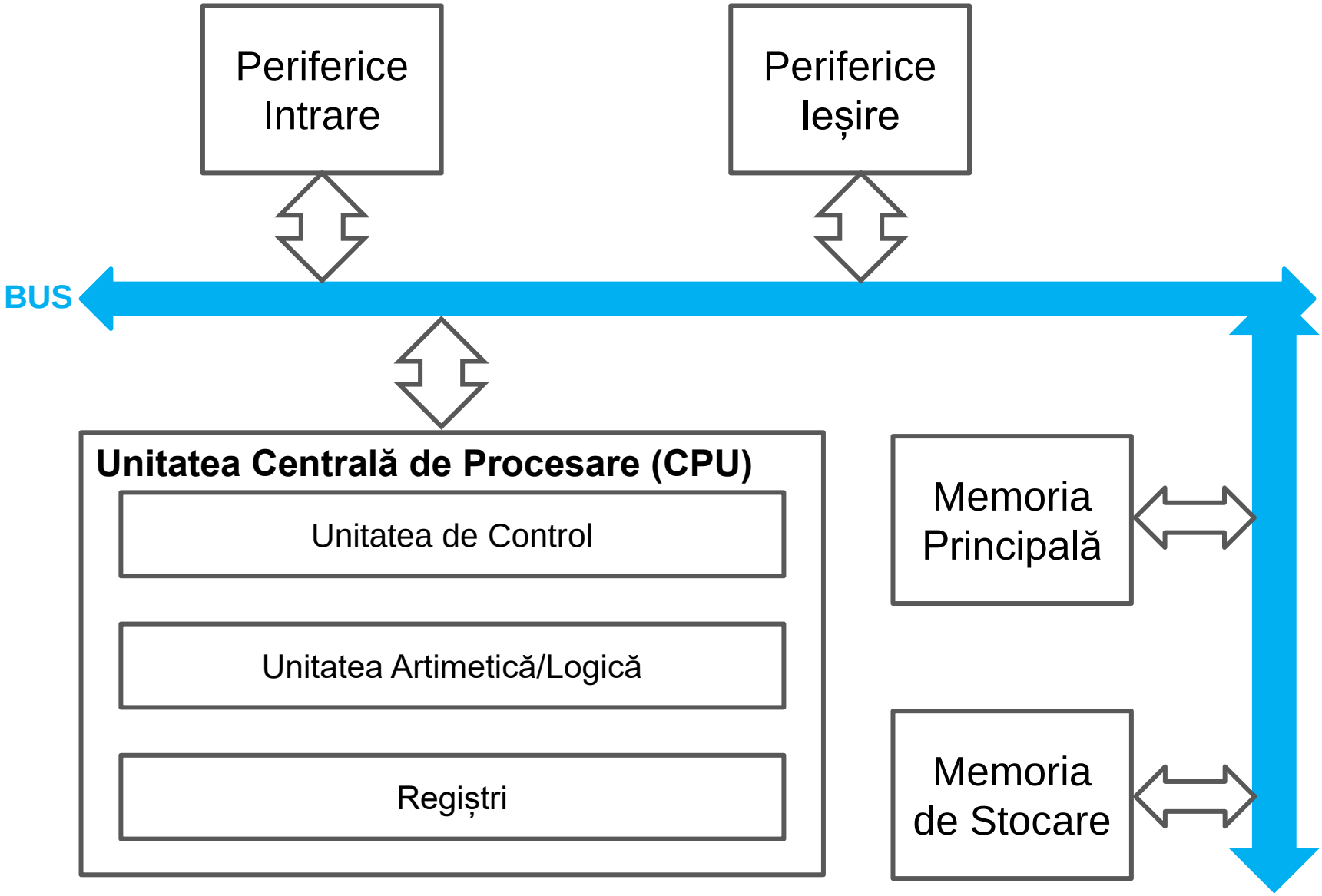
GPU

- **python ML notebook: https://colab.research.google.com/**
- **Edit → Notebook settings → Hardware accelerator: GPU**

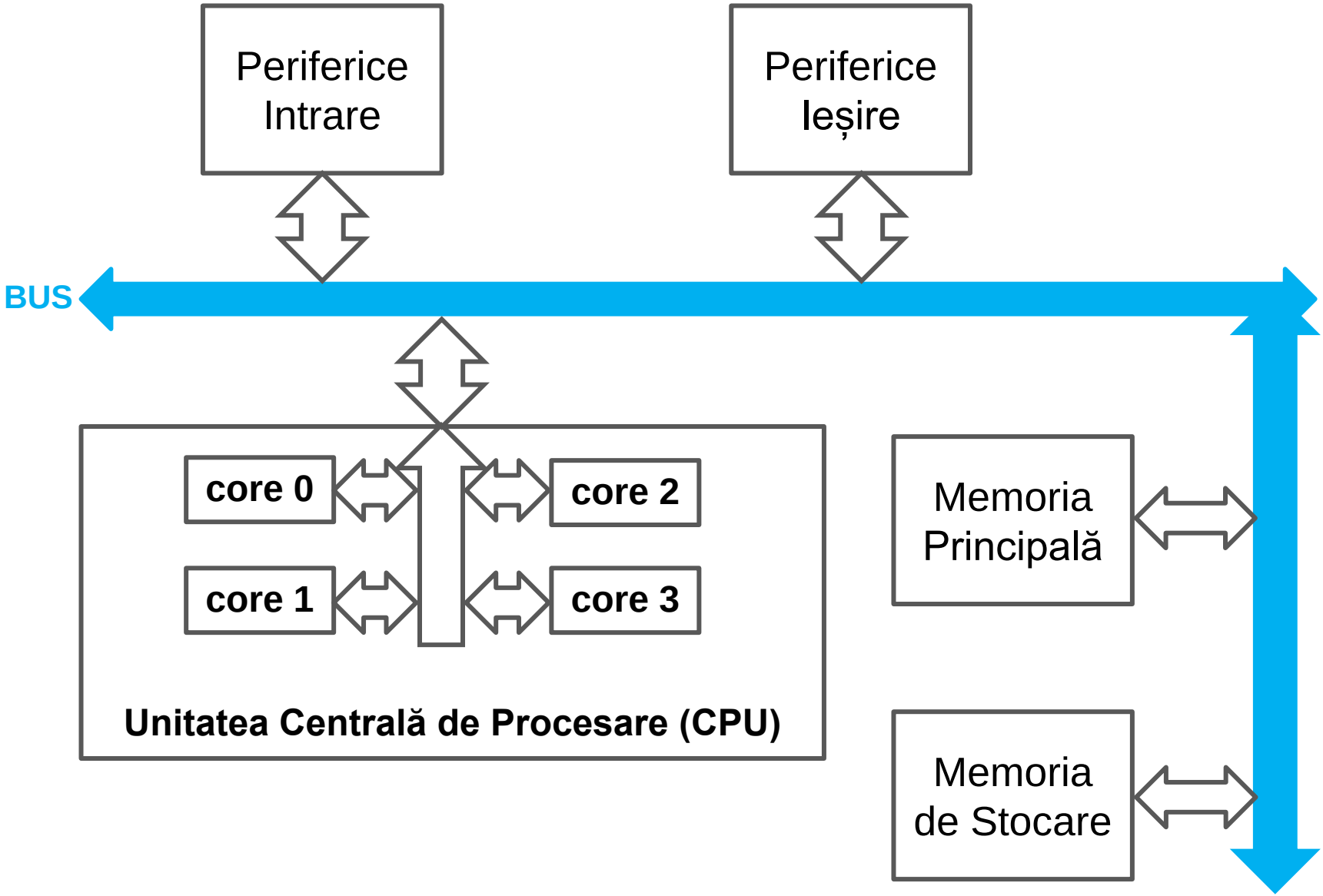
```
import tensorflow as tf
from tensorflow.python.client import device_lib
import time

def measure(x, steps):
    tf.matmul(x, x)
    start = time.time()
    for i in range(steps):
        x = tf.matmul(x, x)

    _ = x.numpy()
    end = time.time()
    return end - start


print("TensorFlow version: {}".format(tf.__version__))
print("Eager execution: {}".format(tf.executing_eagerly()))

print('')

if tf.config.list_physical_devices('GPU'):
    print('GPU ', tf.test.gpu_device_name(), ' is available\n')

# Listeaza ce e disponibil
print('Devices available:\n', device_lib.list_local_devices())

# Dimensiunea matricei
shape = (1000, 1000)
# numarul de simulari
steps = 200

print('')

print("Time to multiply a {} matrix by itself {} times:".format(shape, steps))

# pe CPU:
with tf.device('/cpu:0'):
    print("CPU: {} secs".format(measure(tf.random.normal(shape), steps)))

# pe GPU, daca este disponibil:
if tf.test.is_gpu_available():
    with tf.device('/gpu:0'):
        print("GPU: {} secs".format(measure(tf.random.normal(shape), steps)))
else:
    print('GPU: not found')
```

scrieți și voi o secvență de cod care face înmulțirea a două matrice (python) și comparați cu timpii din această secvență de cod

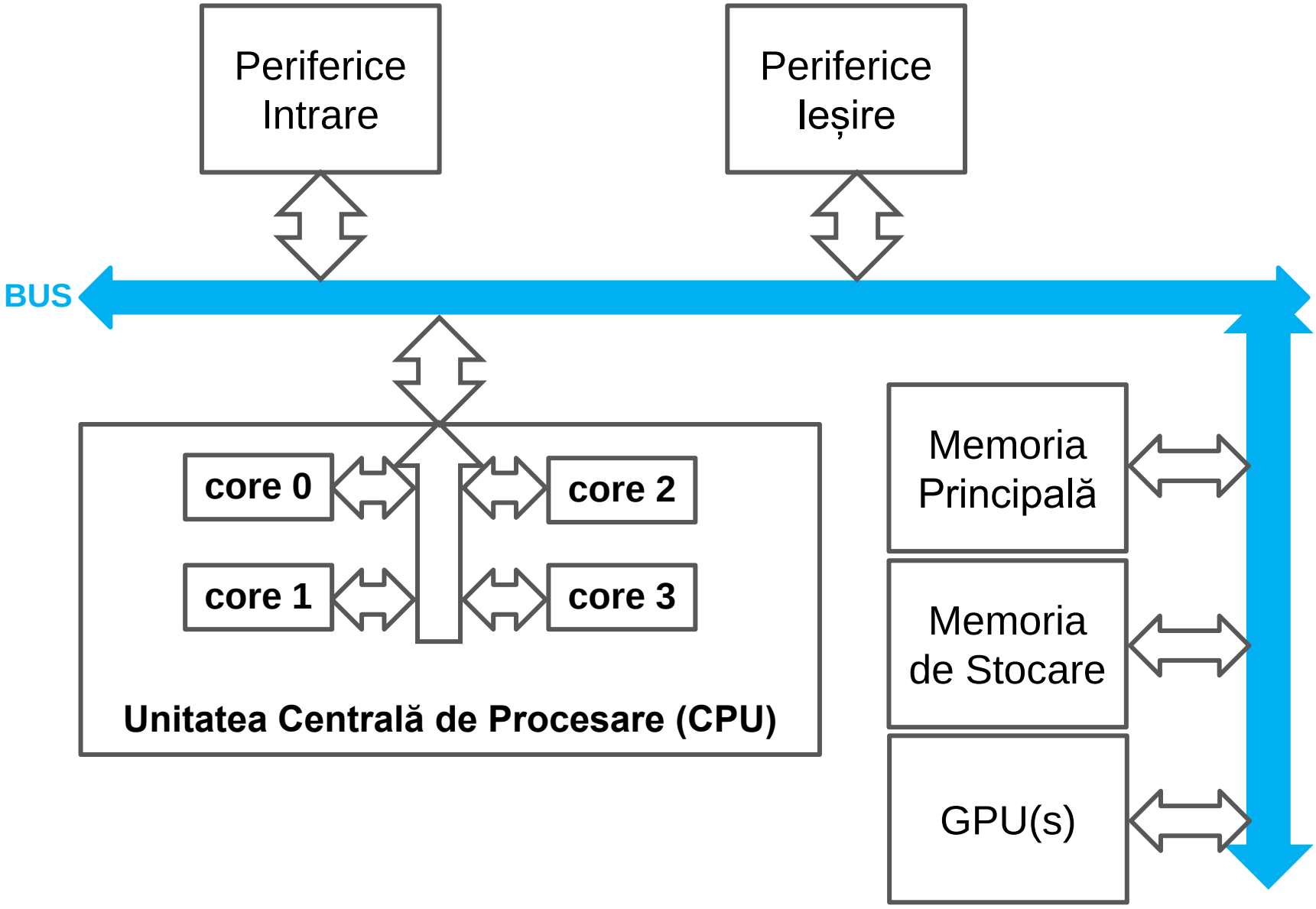
(REVIEW) ARHITECTURA DE BAZĂ



(REVIEW) ARHITECTURA MULTI-CORE



(REVIEW) ARHITECTURA MULTI-CORE ȘI GPU



CUM DEFINIM "PERFORMANȚA"?

- **numărul de instrucțiuni executate**
  - nu toate instrucțiunile sunt la fel de rapide

- **timpul de execuție a unui program**
  - executăm de mai multe ori același program
    - timpul mediu?
    - timpul cel mai rapid/lent?
  - ce program executăm?
    - fiecare program exploatează diferit arhitectura de calcul
    - unele programe exploatează în mod natural paralelismul arhitecturii de calcul

CUM DEFINIM "PERFORMANȚA"?

- **în general, este foarte dificil să definim clar ce înseamnă performanță în contextul arhitecturii calculatoarelor**
  - tipurile de programe variază foarte mult
    - este greu să înțelegem ce înseamnă un program "reprezentativ"
  - arhitecturile de execuție moderne sunt extrem de complicate
  - criteriile de performanță variază mult
    - vrem răspuns rapid?
      - chiar și asta e complicat: datele trebuie citite (de pe disc) în memorie, datele din memorie trebuie transferate la CPU, CPU-ul are o anumită viteză de execuție, apoi din nou acces memorie și/sau Input/Output
    - vrem ca sistemul să poată rula cât mai multe programe simultan?
    - vrem ca sistemul să fie rapid pe operații I/O?
    - vrem ca sistemul să fie rapid pe operații aritmetice?
    - vrem ca sistemul să ruleze jocuri rapid? (frame-rate bun)
    - o combinație de criterii?

TIMPUL DE EXECUȚIE

- **o definiție inițială pentru performanță: timp de execuție**
- **rulăm un program A pe un sistem de calcul X**
  - performanța$_X$ = (timpul de execuție a lui A pe X)$^{-1}$
  - timp de execuție mai mic → performanță mai mare
  - în general, vrem să comparăm și să spunem: sistemul de calcul X este de *n* ori mai rapid decât sistemul de calcul Y
    - performanța$_X$ (performanța$_Y$)$^{-1}$= *n*
  - timpul de execuție îl măsurăm în secunde
    - se numește *wall-clock time*, *response time* sau *elapsed time*
    - este timpul în “lumea reală”
    - de ce e complicat? avem mai mulți timpi?

TIMPUL DE EXECUȚIE

- **o definiție inițială pentru performanță: timp de execuție**
- **rulăm un program A pe un sistem de calcul X**
  - performanța$_X$ = (timpul de execuție a lui A pe X)$^{-1}$
  - timp de execuție mai mic → performanță mai mare
  - în general, vrem să comparăm și să spunem: sistemul de calcul X este de *n* ori mai rapid decât sistemul de calcul Y
    - performanța$_X$ (performanța$_Y$)$^{-1}$= *n*
  - timpul de execuție îl măsurăm în secunde
    - se numește *wall-clock time*, *response time* sau *elapsed time*
    - este timpul în “lumea reală”
    - de ce e complicat? avem mai mulți timpi?
      - un procesor execută simultan mai multe programe
      - *CPU time* = cât timp de execuție a fost alocat pe CPU

PERFORMANȚA CPU

- **pentru CPU**
  - frecvența (*clock rate*), ex. 4GHz
  - ciclu de ceas (*clock cycle*), ex. la fiecare 250 pico secunde (ps)
    - (frecvența)$^{-1}$

- **CPU time pentru A = ciclii de ceas pentru A / frecvența**
  - deci, putem micșora timpul de execuție pentru A dacă
    - reducem numărul de ciclii de ceas necesar pentru a executa A
    - mărim frecvența procesorului

PERFORMANȚA CPU

- **ce face un CPU?**
  - execută instrucțiuni
  - câți ciclii de ceas sunt necesari pentru a executa o instrucțiune?
  - vă reamintesc, nu toate instrucțiunile sunt la fel de “dificile”

operația	instrucțiuni	# cicli
operații întregi/biți	add, sub, and, or, xor, sar, sal, lea, etc.	1
înmulțirea întregilor	mul, imul	3
împărțirea întregilor	div, idiv	depinde (20–80)
adunare floating point	addss, addsd	3
înmulțire floating point	mulss, mulsd	5
împărțire floating point	divss, divsd	depinde (20–80)
fused-multiply-add floating point	vfmass, vfmasd	5

- **ciclii de ceas pentru A = număr instrucțiuni în A × număr ciclii necesar pentru a executa o instrucțiune (în medie)**

PERFORMANȚA CPU

- **fie *x*, *y*, *z* vectori care sunt de tipul *double***

operația	timpul
z[i] = x[i]	51.5 ps
z[i] += x[i]	60 ps
z[i] = x[i] + y[i]	69 ps
z[i] += x[i] + y[i]	74.5 ps
z[i] = x[i] × y[i]	75 ps
z[i] += x[i] × y[i]	93 ps
z[i] = x[i] / y[i]	1120 ps
z[i] = sqrt(x[i])	2050 ps
z[i] = log(x[i])	4200 ps
z[i] = exp(x[i])	5600 ps
z[i] = rand()	1850 ps

cum măsurăm timpul de execuție?

- pentru o secvență de cod
- pentru un executabil (Linux: time ./program)

```
#include <time.h>

clock_t start, end;
double cpu_time_used;

start = clock();
...
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

PERFORMANȚA CPU

- **combinăm ultimele două relații**

CPU time pentru A = număr instrucțiuni în A × număr ciclii necesar pentru a executa o instrucțiune (în medie) / frecvența

- **ce componentă ce afectează?**
  - algoritmul: număr instrucțiuni în A, în anumite situații speciale număr ciclii necesar pentru a executa o instrucțiune (în medie)
  - limbajul de programare: număr instrucțiuni în A, număr ciclii necesar pentru a executa o instrucțiune (în medie)
  - compilatorul: număr instrucțiuni în A, număr ciclii necesar pentru a executa o instrucțiune (în medie)
  - ISA: număr instrucțiuni în A, număr ciclii necesar pentru a executa o instrucțiune (în medie), frecvența

PERFORMANȚA CPU

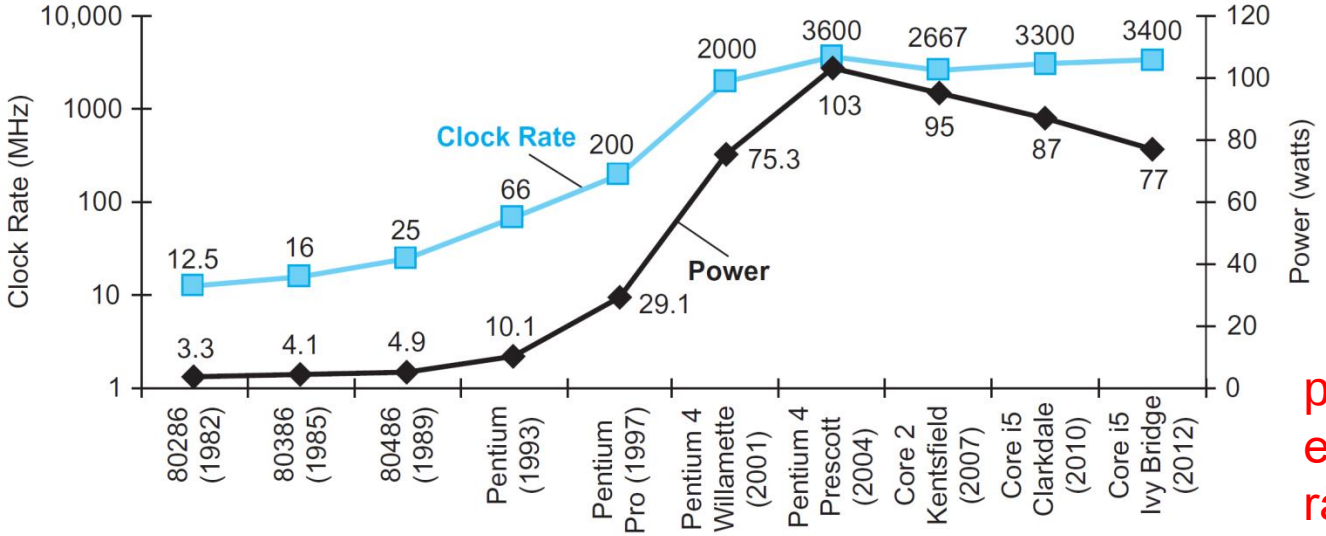
- **situația cu instrucțiunile e complicată**
  - poți executa puține instrucțiuni, dar unele pot avea nevoie de mai mulți ciclii de calcul pe CPU (deci este un trade-off)

- **situația cu frecvența procesorului e simplă**
  - frecvență mai mare → performanță mai ridicată
  - adică, mereu vrem să executăm mai repede
  - care e problema? de ce nu avem procesoare la 10GHz?

PERFORMANȚA CPU

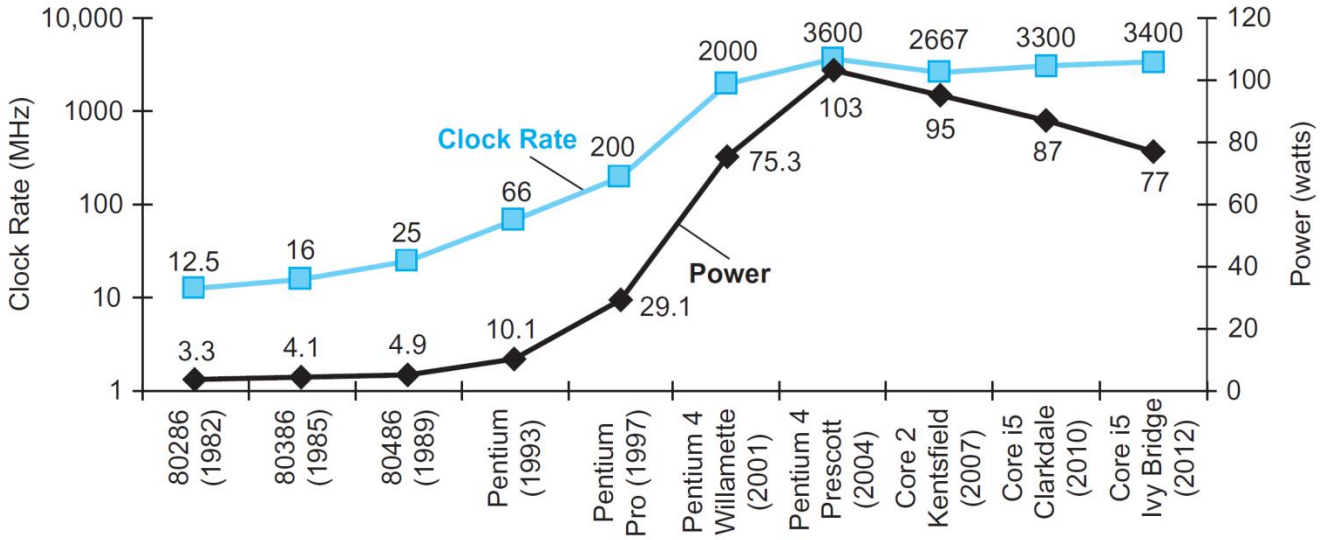
- **situația cu instrucțiunile e complicată**
  - poți executa puține instrucțiuni, dar unele pot avea nevoie de mai mulți ciclii de calcul pe CPU (deci este un trade-off)

- **situația cu frecvența procesorului e simplă**
  - frecvență mai mare → performanță mai ridicată
  - adică, mereu vrem să executăm mai repede
  - care e problema? de ce nu avem procesoare la 10GHz?



problema este energia consumată, răcirea sistemului

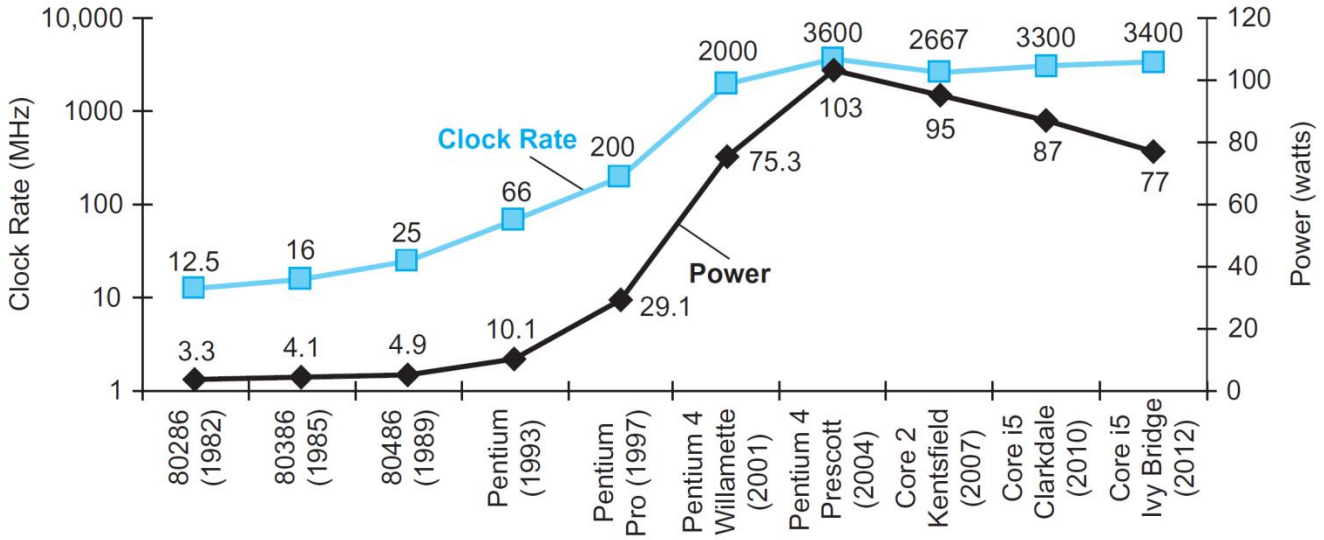
PERFORMANȚA CPU



- **puterea = O(tensiune² × frecvență)**
- **uitați-vă pe grafic:**
  - am mers de la 25MHz la 2000MHz (×100)
  - în același timp puterea a mers de la 5W la 103W (×20)
  - cum e posibil?
    - tensiunea de funcționare a circuitelor a scăzut de la 5V la 1V
    - de ce nu putem merge mult sub 1V?

PH Book, pagina 40; https://en.wikipedia.org/wiki/Processor_power_dissipation; https://en.wikipedia.org/wiki/Dennard_scaling

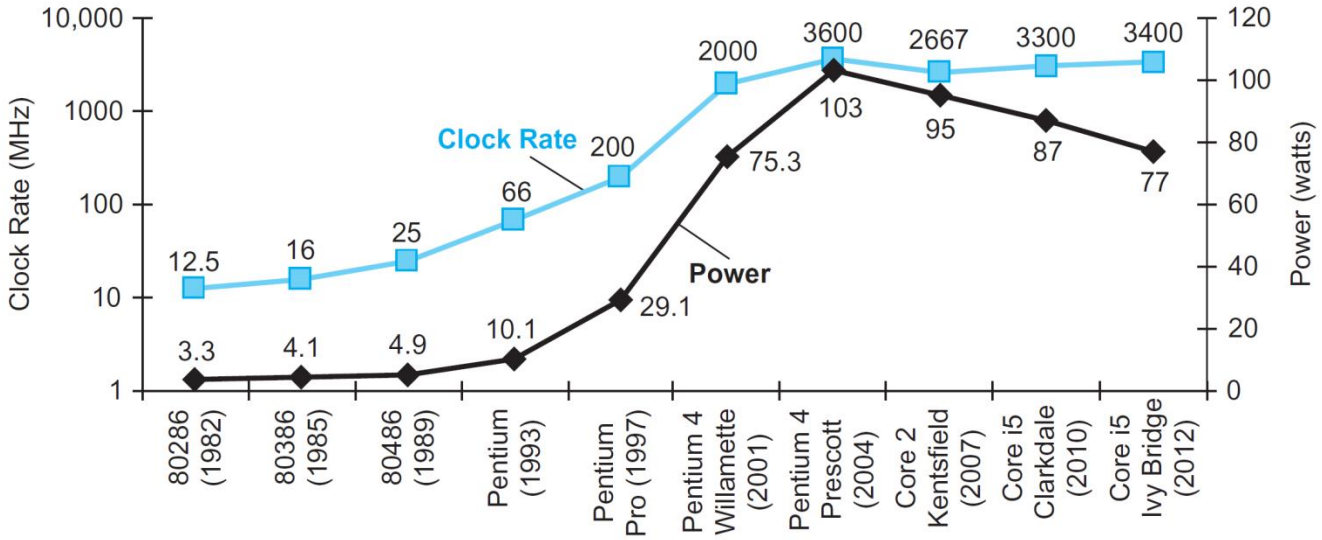
PERFORMANȚA CPU



- **puterea = O(tensiune² × frecvență)**
- **uitați-vă pe grafic:**
  - am mers de la 25MHz la 2000MHz (×100)
  - în același timp puterea a mers de la 5W la 103W (×20)
  - cum e posibil?
    - tensiunea de funcționare a circuitelor a scăzut de la 5V la 1V
    - de ce nu putem merge mult sub 1V? zgomot!

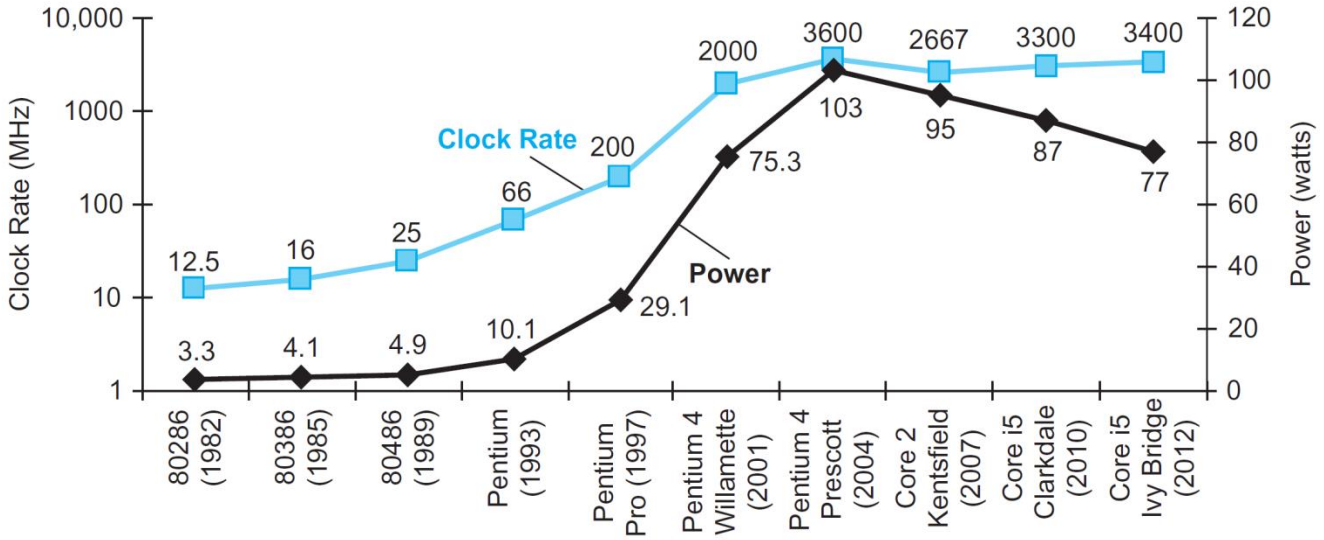
PH Book, pagina 40; https://en.wikipedia.org/wiki/Processor_power_dissipation; https://en.wikipedia.org/wiki/Dennard_scaling

PERFORMANȚA CPU



- **mai sunt două probleme la 10GHz**
  - toate celelalte componente hardware ar trebui să funcționeze la viteze comparabile (altfel, CPU așteaptă mereu)
  - (1 / (10 GHz)) * (299 792 458 (m / s)) = 2.99 centimetri
    - ce semnificație are acest calcul?

PERFORMANȚA CPU



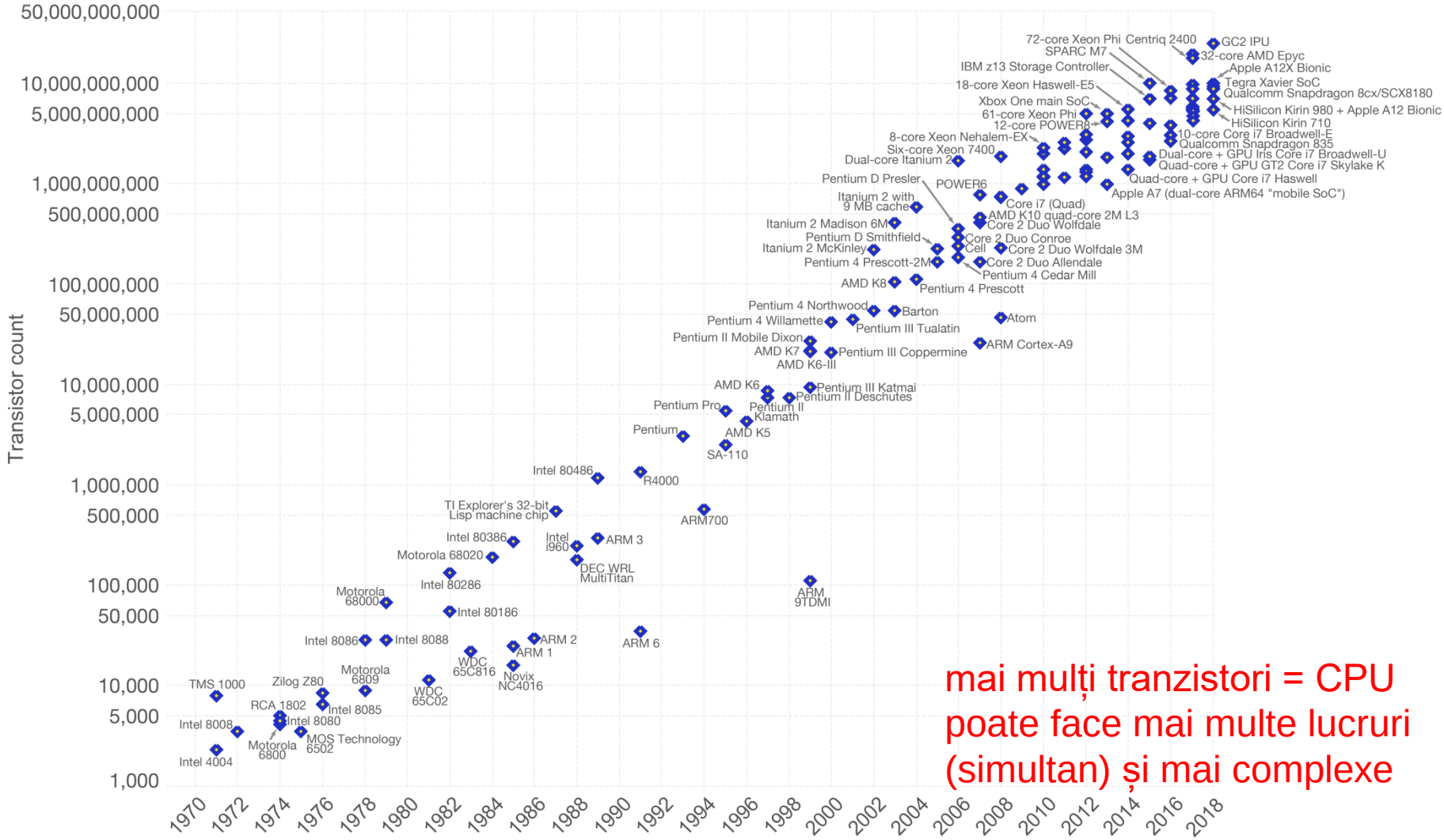
- **mai sunt două probleme la 10GHz**
  - toate celelalte componente hardware ar trebui să funcționeze la viteze comparabile (altfel, CPU așteaptă mereu)
  - (1 / (10 GHz)) * (299 792 458 (m / s)) = 2.99 centimetri
    - ce semnificație are acest calcul?
    - este distanța (ideală, maximă) pe care o poate parcurge un semnal electric emis de un CPU care funcționează la 10GHz înainte ca același CPU să mai poată emite un nou semnal

PERFORMANTA CPU

Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



mai mulți tranzistori = CPU poate face mai multe lucruri (simultan) și mai complexe

Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

PERFORMANȚA CPU

- **tehnologia curentă (dimensiunea tranzistorilor)**
  - 5nm
  - 5nm = 50 Å

https://spectrum.ieee.org/semiconductors/devices/the-nanosheet-transistor-is-the-next-and-maybe-last-step-in-moores-law

PERFORMANȚA CPU

- **tehnologia curentă (dimensiunea tranzistorilor)**
  - 5nm
  - 5nm = 50 Å (Angstrom)

https://spectrum.ieee.org/semiconductors/devices/the-nanosheet-transistor-is-the-next-and-maybe-last-step-in-moores-law

PERFORMANȚA CPU

- **tehnologia curentă (dimensiunea tranzistorilor)**
  - 5nm
  - 5nm = 50 Å (Angstrom)
  - 1 atom are un diametru de aproximativ 1 Angstrom

https://spectrum.ieee.org/semiconductors/devices/the-nanosheet-transistor-is-the-next-and-maybe-last-step-in-moores-law

PERFORMANȚA CPU

- **tehnologia curentă (dimensiunea tranzistorilor)**
  - 5nm
  - 5nm = 50 Å (Angstrom)
  - 1 atom are un diametru de aproximativ 1 Angstrom
  - intervine mecanica cuantică
    - nu mai știm exact unde este electronul
    - tranzistorul nu mai are cum să funcționeze corect (mereu)
    - nu avem cum să mai reducem dimensiunea tranzistorului și să păstrăm fiabilitatea sa
  - observație: calculatoarele cuantice nu au legatură cu această problemă

https://spectrum.ieee.org/semiconductors/devices/the-nanosheet-transistor-is-the-next-and-maybe-last-step-in-moores-law

PERFORMANȚA MULTI-CORE

- **ce se întâmplă dacă avem un sistem multi-core?**
  - putem rula mai multe programe simultan
    - același program, instanțe diferite
    - diferite programe
  - putem rula un program mai eficient (paralelizând părți ale sale)
    - presupunem că avem *s* procesoare
    - presupunem că *p*% din program poate beneficia teoretic de paralelizare/îmbunătățire (unele secțiuni de cod sunt doar secvențiale, acolo nu se poate face nimic)
    - **legea lui Amdahl** (speed-up *S*):

$$S = \frac{1}{(1-p) + \frac{p}{s}}$$

    - **legea lui Gustafson** (speed-up *S*):

$$S = 1 - p + \delta p$$

  - de multe ori, implementările paralele au nevoie să comunice date (asta poate câteodată domina calculul)

CISC VS. RISC

- **Complex Instruction Set Computers**
- **Reduced Instruction Set Computers**

CISC	RISC
ISA original	ISA apărută în anii ’80, popularitate ridicată acum (RISC-V)
hardware complicat	software complicat
instrucțiuni complicate (au nevoie de mai mulți cicli de ceas), lungime variabilă pentru instrucțiuni	instrucțiuni simple (fiecare are nevoie de un singur ciclu de ceas), lungime fixă pentru instrucțiuni
multe operații au loc memorie-memorie (citirea/scrierea în memorie este incorporată în instrucțiuni)	multe operații au loc registru-registru
suportă multe metode de adresare	metode simple (și puține) de adresare
cod scurt (puține instrucțiuni)	cod lung (multe instrucțiuni)
logica este complexă (tranzistori mulți)	logica este simplă (tranzistorii sunt alocați pentru memorie, etc.)

CISC VS. RISC

- **Complex Instruction Set Computers**
  - x86
  - Motorola

- **Reduced Instruction Set Computers**
  - MIPS (Microprocessor without Interlocked Pipelined Stages)*
  - Power PC
  - Atmel AVR (mașini Harvard)
  - PIC Microchip
  - ARM (Advanced RISC Machine)
  - RISC-V

* a nu se confunda cu Millions Instructions Per Second (MIPS)

CISC VS. RISC

- **exemplu: înmulțirea a două numere aflate în memorie**
- **MEM_LOC2 = MEM_LOC1 × MEM_LOC2**

  - Complex Instruction Set Computers
    - **MUL** MEM_LOC_1, MEM_LOC_2

  - Reduced Instruction Set Computers
    - **LOAD** MEM_LOC1, R1
    - **LOAD** MEM_LOC2, R2
    - **MUL** R1, R2
    - **STORE** R2, MEM_LOC2

* a nu se confunda cu Million Instructions Per Second

PERFORMANȚA PER WATT

- **un criteriu nou de performanță: cantitatea de energie consumată**
- **extrem de important pentru dispozitive pe baterie:**
  - laptop-uri
  - tablete
  - telefoane inteligente
  - sisteme embedded, sisteme Internet of Things (IoT)

- **toate aceste dispozitive își determină dinamic ciclul de ceas pentru a balansa consum de energie, răcire și performanță**

- **în general, aici RISC domină clar CISC**
  - domeniu activ de cercetare: New RISC-V CPU claims recordbreaking performance per watt, https://arstechnica.com/gadgets/2020/12/new-risc-v-cpu-claims-recordbreaking-performance-per-watt/

OPTIMIZAREA COMPILATORULUI

- **am văzut deja că în multe situații compilatorul face multe optimizări pentru noi**
  - se asigură că rezultatul este același
  - dacă calculul este determinist, compilatorul calculează răspunsul final și în pune explicit în fișierul compilat
  - compilatorul alege ce funcții să apeleze (exemplu: printf vs. puts)
  - compilatorul alege dacă și cum să folosească instrucțiuni speciale (exemple: SIMD, div/mul, etc.)

- **în general, compilatorul își face treaba foarte bine**
  - compilatorul e mai capabil decât mulți programatori
  - compilatorele sunt îmbunătățite de decenii, deci conțin multe structuri/trucuri pentru a optimiza codul vostru
  - poate să transforme codul vostru în Assembly care logic este mai dificil de citit/debug pentru oameni, dar mai eficient ca instrucțiuni
  - compilatorul nu este perfect, nu este mereu corect
  - compilatorul nu poate (deocamdată) înlocui un programator capabil

OPTIMIZAREA COMPILATORULUI

- **fiecare compilator are setările sale**
- **pentru GNU C Compiler (GCC)**

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

https://www.rapidtables.com/code/linux/gcc/gcc-o.html, https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

OPTIMIZAREA COMPILATORULUI

- **ce fel de optimizări suportă procesorul vostru**
- **cat /proc/cpuinfo**
  - processor : 0
  - model name : Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz
  - Byte Order: Little Endian
  - cache size : 300072 KB
  - flags : **fpu** vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca **cmov** pat pse36 clflush dts acpi **mmx** fxsr **sse sse2** ss ht tm pbe **syscall** nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 **ssse3** sdbg **fma** cx16 xtpr pdcm pcid **sse4_1 sse4_2** movbe **popcnt** tsc_deadline_timer aes xsave **avx** f16c rdrand lahf_lm abm epb tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts
  - ...

DIMENSIUNEA FIȘIERELOR

- **este un indicator destul de bun pentru numărul de instrucțiuni**
- **compilatorul/SO-ul mereu adaugă**
  - header
  - informație de mediul de execuție
  - etc.

- **pentru gcc, flagul de optimizare pentru dimensiune cod este -Os**

```c
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

```c
#include <unistd.h>
#include <sys/syscall.h>

void _start() {
    const char msg [] = "Hello World!";
    syscall(SYS_write, 0, msg, sizeof(msg)-1);
    syscall(SYS_exit, 0);
}
```

```asm
.data
        helloWorld: .asciz "Hello World!\n"

.text

.globl _start

_start:
        mov $4, %eax
        mov $1, %ebx
        mov $helloWorld, %ecx
        mov $14, %edx
        int $0x80

        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

DIMENSIUNEA FIȘIERELOR

- **este un indicator destul de bun pentru numărul de instrucțiuni**
- **compilatorul/SO-ul mereu adaugă**
  - header
  - informație de mediul de execuție
  - etc.

- **pentru gcc, flagul de optimizare pentru dimensiune cod este -Os**

```c
#include <stdio.h>
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

18k

```c
#include <unistd.h>
#include <sys/syscall.h>

void _start() {
    const char msg [] = "Hello World!";
    syscall(SYS_write, 0, msg, sizeof(msg)-1);
    syscall(SYS_exit, 0);
}
```

14k

```
.data
        helloWorld: .asciz "Hello World!\n"

.text

.globl _start

_start:
        mov $4, %eax
        mov $1, %ebx
        mov $helloWorld, %ecx
        mov $14, %edx
        int $0x80

        mov $1, %eax
        mov $0, %ebx
        int $0x80
```

8k

Hello World în sub 1kb: http://timelessname.com/elfbin/, https://reflexive.space/2018-05-05-really-small-elves/

PERFORMANȚA LIMBAJE PROGRAMARE

- **limbajul de programare + arhitectura de calcul**

- **exemplu, înmulțirea matrice-matrice**
  - python
  - java
  - C
  - C + optimizarea ciclurilor for
  - C + optimizarea compilării (optimization flags)
  - C + paralelizare
  - C + paralelizare + exploatarea cache-ului
  - C + paralelizare + exploatarea cache-ului + SIMD FP

PERFORMANȚA LIMBAJE PROGRAMARE

- **înmulțirea matrice-matrice**
  - Python (cod interpretat)
  - Java (cod compilat în bytecode și interpretat de Java VM)
  - C (cod compilat, nativ)
  - C + optimizarea ciclurilor for (cod compilat, încep optimizări)
  - C + optimizarea compilării (optimization flags)
  - C + paralelizare
  - C + paralelizare + exploatarea cache-ului
  - C + paralelizare + exploatarea cache-ului + SIMD FP

de **53292** ori mai rapid decât codul în python

- **discuție detaliată (Charles Leiserson, MIT 6.172 Performance Engineering of Software Systems)**
  - https://youtu.be/o7h_sYMk_oc?t=1177

PERFORMANȚA VS. SECURITATE

- de cele mai multe ori există o balanță între performanță și elemente de securitate în software
- de ce?

PERFORMANȚA VS. SECURITATE

- **de cele mai multe ori există o balanță între performanță și elemente de securitate în software**
- **de ce?**
  - verificări suplimentare de input de la user
  - verificări suplimentare la dimensiunea variabilelor/bufferelor
  - randomizarea adreselor de memorie este o metodă care sporește securitatea sistemului
  - criptare/decriptare date

PERFORMANȚA TOTALĂ

- **practic, performanța unui sistem de calcul se află**
  - simulând încărcarea uzuală prezisă (*predicted workload*)
  - rularea unor programe standard de test pentru măsurarea performanței (*benchmarking*)
- **Standard Performance Evaluation Corporation (SPEC)**
  - benchmarking standardizat
  - evaluează
    - performanța de calcul
    - consumul de energie
  - rezultate: http://www.spec.org/cpu2017/results/cpu2017.html

PERFORMANȚA TOTALĂ

- **este această cantitate una deterministă?**
  - dacă rulez același program (executabil) pe același sistem de calcul (arhitectura completă) cu exact aceleași setări (de exemplu folosește SIMD, etc.) și stare inițială (memorie RAM, cache inițializată la fel), am același timp de rulare?
  - NU
  - de ce?

.

PERFORMANȚA TOTALĂ

- **este această cantitate una deterministă?**
  - dacă rulez același program (executabil) pe același sistem de calcul (arhitectura completă) cu exact aceleași setări (de exemplu folosește SIMD, etc.) și stare inițială (memorie RAM, cache inițializată la fel), am același timp de rulare?
  - NU
  - de ce?
  - nu uitați de detectarea și corectarea erorilor
    - deci, sistemul vostru de calcul nu face exact aceleași operații niciodată (cu probabilitate mare)

CE AM FĂCUT ASTĂZI

- **performanța calculatoarelor**
  - CPU
  - multi-core
  - compilator
  - limbajul de programare
- **CISC vs. RISC**
- **consumul de energie**

DEMO

- **64k demo**

.

DEMO

- **64k demo**

- **ce am văzut la curs**
  - fr-08: .the .product by farbrausch | 64k intro (2000), https://www.youtube.com/watch?v=Y3n3c_8Nn2Y

- **alte 64k demos (preferatele mele):**
  - fermi paradox - mercury | 60fps | Revision 2016 | 64k, https://www.youtube.com/watch?v=JZ6ZzJeWgpY
  - Conspiracy - Darkness Lay Your Eyes Upon Me (Demoscene 2016) 64k demo, https://www.youtube.com/watch?v=WKc1cozQHrM

- **v-am arătat aceste demo-uri nu ca să faceți și voi astfel de programe, dar ca să vă arăt ce puteți face atunci când înțelegeți ce faceți**

LECTURĂ SUPLIMENTARĂ

- **PH book**
  - 1.2 Eight Great Ideas in Computer Architecture
    - **Design for Moore's Law**
    - **Use Abstraction to Simplify Design**
    - **Make the Common Case Fast**
    - **Performance via Parallelism**
    - **Performance via Pipelining**
    - **Performance via Prediction**
    - **Hierarchy of Memories**
    - **Dependability via Redundancy**
  - 1.6 Performance
  - 1.7 The Power Wall
  - 1.9 Real Stuff: Benchmarking the Intel Core i7

- **două interviuri:**
  - Jim Keller: Moore's Law, Microprocessors, and First Principles, https://www.youtube.com/watch?v=Nb2tebYAaOA
  - David Patterson*: Computer Architecture and Data Storage, https://www.youtube.com/watch?v=naed4C4hfAg

* este "P"-ul din PH book

LECTURĂ SUPLIMENTARĂ (NU INTRĂ ÎN EXAMEN)

- **Introduction to GPU programming,** https://www.youtube.com/watch?v=uvVy3CqpVbM
- **Performance matters,** https://www.youtube.com/watch?v=r-TLSBdHe1A
- **Don’t use lists,** https://www.youtube.com/watch?v=YQs6IC-vgmo
- **The Voyager computer,** https://www.youtube.com/watch?v=H62hZJVqs2o
- **Overview of computer architecture,** https://www.youtube.com/watch?v=yOa0WpMwzWk

\0

.