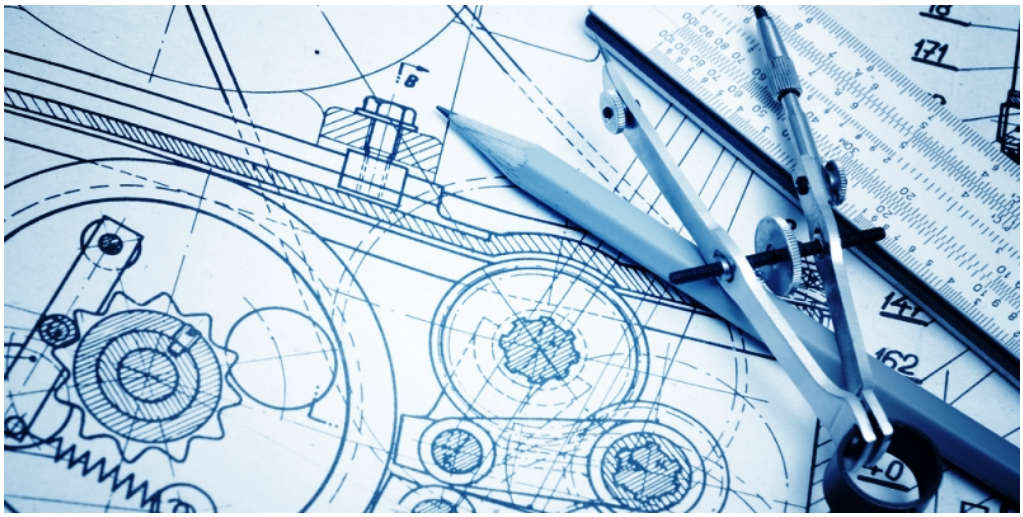
Computer Science & Engineering Department

Computer Science & Engineering

cs.pub.ro
acs.curs.pub.ro
Emil-Ioan Slușanschi

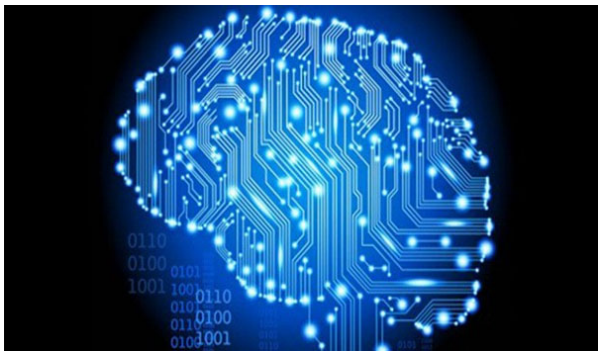
University Politehnica of Bucharest



Educatie Inginereasca

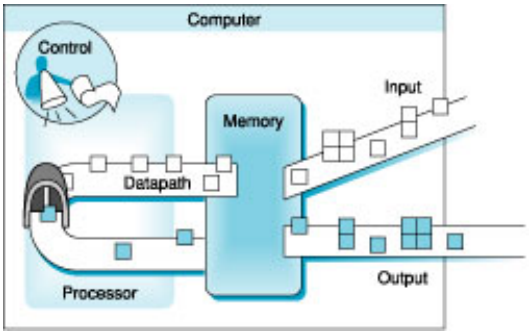
Ce este un Computer?

- Un instrument pentru a rezolva probleme
  - Prelucrarea datelor/numerelor
  - Prelucrarea algoritmilor
- Un procesor ce controleaza un sistem (avion, fabrica, monitor de inima, trafic, robot, etc.)
  - Senzori / Intrari
  - Actuator / Iesiri
  - Functii
  - Stari



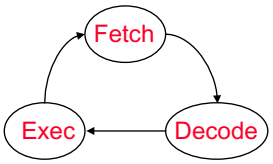
@Yale Patt - University of Texas at Austin

Ce este un Computer? (2)



- Components:
  - Processor (Datapath & Control)
  - Input (mouse, keyboard)
  - Output (display, printer)
  - Memory (cache, main memory, disks, CD/DVD)
  - Network
- Our primary focus: the Processor (**Datapath** & **Control**)
  - Implemented using billions of transistors
  - Impossible to understand by looking at each transistor
  - We need abstraction!

Processor Organisation



- **Control** needs to have circuitry to
  - Decide which is the next instruction and input it from memory
  - Decode the instruction
  - Issue signals that control the way information flows between datapath components
  - Control what operations the datapath's functional units perform
- **Datapath** needs to have circuitry to
  - Execute instructions - functional units (e.g. adder, multiplier, etc.) and storage locations (e.g. register file)
  - Interconnect the functional units so that the instructions can be executed as required
  - Load data from and store data to memory

Typical Design Goals for Processors

- Maximize performance (runtime, throughput, etc.)
- Minimize memory space (embedded systems)
- Minimize power consumption (mobile systems)
- Reduce design time (time-to-market)
- Minimize cost ($)

The Four Design Principles (CN)

- Simplicity favours regularity
- Smaller is faster
- Make the common case fast
- Good design demands good compromises

Educatia Inginereasca

- Ce trebuie sa stie un inginer?
  - Sa foloseasca unelte (HW/SW)
  - Sa proiecteze si sa dezvolte/realizeze sisteme
- Pentru aceasta trebuie sa invete/inteleaga:
  - Cum sunt reprezentate numerele
  - Cum functioneaza calculatoarele
  - **Cum functioneaza un algoritm pe un calculator**
  - Cum se ajunge de la senzori/intrari prin programe la actuatori/iesiri
- Ceea ce nu este neaparat necesar:
  - Excel, Word, Web browsing (random), invatare pe de rost

Ingineri vs. Probleme

- Problem solving **is** programming
- Inginerii au rezolvat mereu probleme – asta este treaba lor!
- Inginerii NU descriu problemele lor unor matematicieni, si asteapta de la acestia o ecuatie care sa modeleze problema…
- Se asteapta ca inginerii sa poata sa descrie singuri problemele lor in mod matematic / algoritmic
- In cazurile in care se apeleaza la serviciile unui matematician, trebuie sa ne asiguram ca dialogul este “**coerent**” si “**util**”
  - Ambii utilizeaza un limbaj comun

Ingineri vs. Programatori

- Problemele de astazi sunt in mare majoritate rezolvate de programe
- Putem sa incredintam rezolvarea problemei unui om care nu stie (mai) nimic despre tehnologia utilizata in rezolvarea problemei?
- Trebuie sa ne asteptam ca inginerul sa poata descrie problema in mod algoritmic?
- In cazul in care apelam la serviciile unui programator trebuie sa ne asiguram ca exista un dialog “**coerent**” si “**util**” intre inginer si programator?

Introduction to Computing

- De ce este acesta un subiect esential pentru **toti** inginerii – si necesita mai mult decat o prezentare punctuala a unor tehnici de programare?
  - Competente de baza – fizica / analiza / electronica
    - Proiecteaza procesorul ce va fi folosit
  - Engineering is about Design
    - Studentii pot intelege mai bine lucrurile intr-o abordare bottom-up
  - Engineering is about Tradeoffs (Totul se plateste…)
    - Exemple tipice: iterativ vs. recursiv, latenta vs. bandwidth, dimensiune vs. cost
  - Engineering is about State
    - Exemple multiple in HW si SW
  - Inginerii doresc sisteme deterministice
    - Totul trebuie sa functioneze “corect” si “predictibil”

Programarea in Limbajul X

- De ce aceasta abordare e gresita?
  - Abordarea (traditionala) este aproape totdeauna top-down
    - Astfel se ajunge la memorare in loc de intelegere
  - Efectele memorarii:
    - Students don’t ever get it!
    - Daca nu au memorat tot, nu realizeaza greselile pe care le fac
    - Daca au memorat tot, au impresia ca forma e cea mai importanta si au impresia ca au inteles desi le lipseste intelegerea de fond…
    - Cookbook education == “Forma fara fond!”
  - Nu ofera nicio intuitie asupra uneltelor importante
  - Nu ofera nicio intuitie in modul de interactionare intre procesor si restul sistemului
  - Nu ofera nicio intuitie asupra compromisurilor (tradeoffs) si starilor sistemului

Ce este important?

- Motivatia si design-ul sunt **Top-down**
- Educatia/invatarea trebuie sa fie **Bottom-up** – pentru a facilita intelegerea
- Abstractizarea este vitala
- Pentru a proiecta bine un sistem, trebuie mai intai intelese componentele (necesare ale) acestuia!
- Trebuie mers de la concret catre abstract
- Trebuiesc traversate si intelese nivelele intermediare
- Memorarea NU este intelegere
- Studentii lucreaza mai bine in grupuri

Why Computer Science **& Engineering**?

- Embarrassing if you are a BS in CSE and can't make sense of the following terms: DRAM, pipelining, cache hierarchies, I/O, virtual memory, …
- Embarrassing if you are a BS in CSE and can't decide which processor to buy wrt different metrics: 2.2 GHz Xeon or 3.6 GHz A8
  - Performance vs. power
- Obvious first step for chip designers, compiler & OS writers
- **Will knowledge of the hardware help you write better programs?**

Must a Programmer **care** about Hardware?

- Must know how to reason about program performance and energy
- Memory management: if we understand how/where data is placed, we can help ensure that relevant data is **nearby**
- Thread management: if we understand how threads **interact**, we can write **smarter** multi-threaded programs
  - Why do we care about multi-threaded programs?
  - **Because most HW is now multi/many-core**

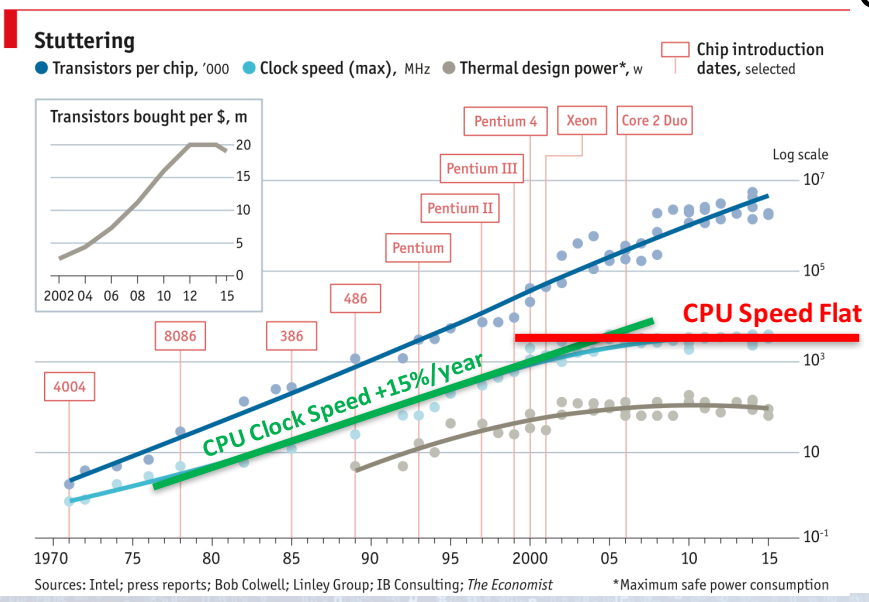
Ce vom discuta la Calculatoare?

- Discussions about the hardware-software interface
  - What do you need to know to achieve best possible performance
- Introduction to the design of major components of a computer system, how they function together in an executing program
- But what do we mean by a computer?
  - Different types: mobile, embedded, laptop, desktop, server,…
  - Different uses: AI, auto, graphics, finance, genomics,…
- Best way to learn:
  - Focus on a specific instance and learn how it works
  - While learning general principles and historical perspectives

De ce sa discutam de HW/SW?

- You want to call yourself a “computer scientist/engineer”
- You want to build HW/SW people use
  - So you need to deliver **performance** at **low cost**
- You need to make a purchasing decision or offer “expert” advice
- Both hardware and software affect performance
  - The algorithm determines number of source-level statements
  - The language/compiler/architecture determines the number of machine instructions
  - The processor/memory determines how fast machine-level instructions are executed

Why is CSE Exciting Today?



ACM Communications, February 2019

Important Trends

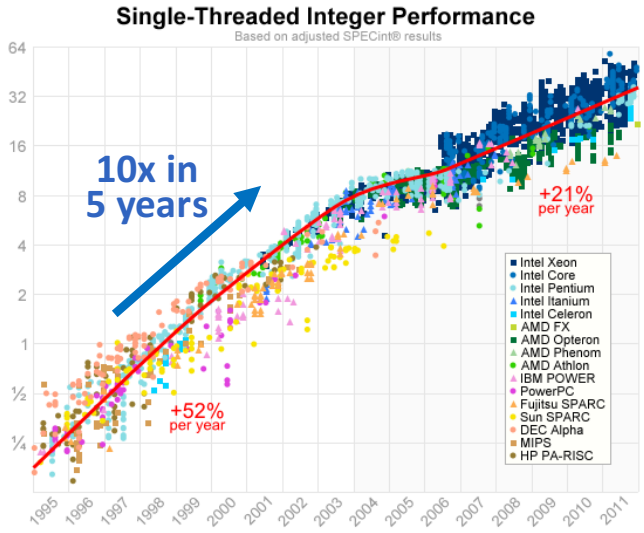
- Running out of ideas to improve single thread performance
- Power wall makes it harder to add complex features
- Power wall makes it harder to increase frequency
- Historical contributions to performance:
  - Better processes (faster devices) ~**20%** (eventually disappearing)
  - Better circuits/pipelines ~**15%** (trending lower)
  - Better organization/architecture ~**15%**

Practical Example

- +**300x** speedup for matrix vector multiplication
  - Data level parallelism: **3.8x**
  - Loop unrolling and out-of-order execution: **2.3x**
  - Cache blocking: **2.5x**
  - Thread level parallelism: **14x**
- Further, one can use accelerators to get an additional **100x**

Predicting the Future

- Predictable performance improvements
- Predictable market
  - Everybody buys a new PC every X years
- Predictable expenses
  - Cost of new integrated circuit fab
- CEOs and Wall Street like this



Opportunities

- Computer architecture today
  - General purpose: optimized for speed
  - Few players: Intel, AMD, ARM,…
- Future directions?
  - Applications have different requirements
    - Battery life / energy
    - Security & privacy
- Best solutions depend on application
  - Many players: HW / SW
  - **Innovation matters more than Moore's Law**

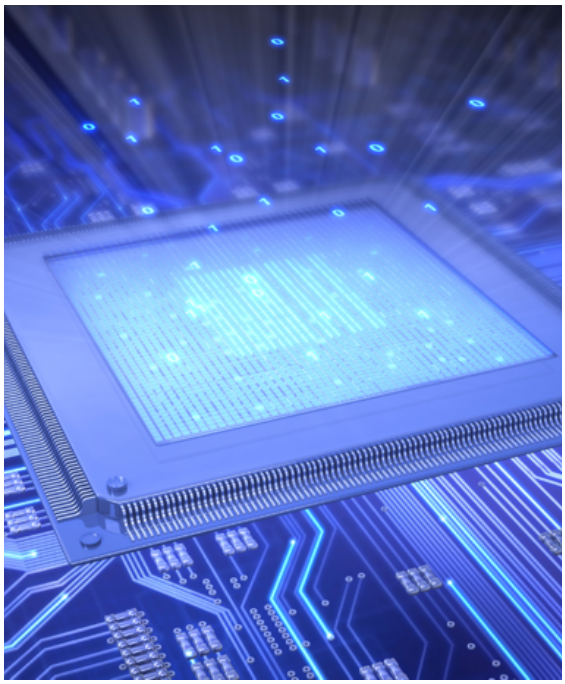
What Does This Mean to a Programmer?

- Today, one can expect at most a 20% annual improvement; even lower if the program is not multi-threaded
  - A program needs many threads
  - The threads need efficient synchronization and communication
  - Data placement in the memory hierarchy is important
  - Accelerators should be used whenever possible

Challenges for Hardware Designers

- Find efficient ways to
  - Boost single-thread performance
  - Improve data sharing & synch techniques
  - Boost programmer productivity
  - Manage the memory system
  - Build accelerators for important kernels
  - Reduce system energy per instruction

Slides & Support Material Acknowledgements



Yale Patt - University of Texas at Austin 2007
CSE 331 - Penn State University 2007
CS61C - University of Berkeley 2016
CS/ECE 3810 - University of Colorado 2017
Ryan J. Leng

Thank you for your attention

Computer Science & Engineering Department

emil.slusanschi@cs.pub.ro
cs.pub.ro