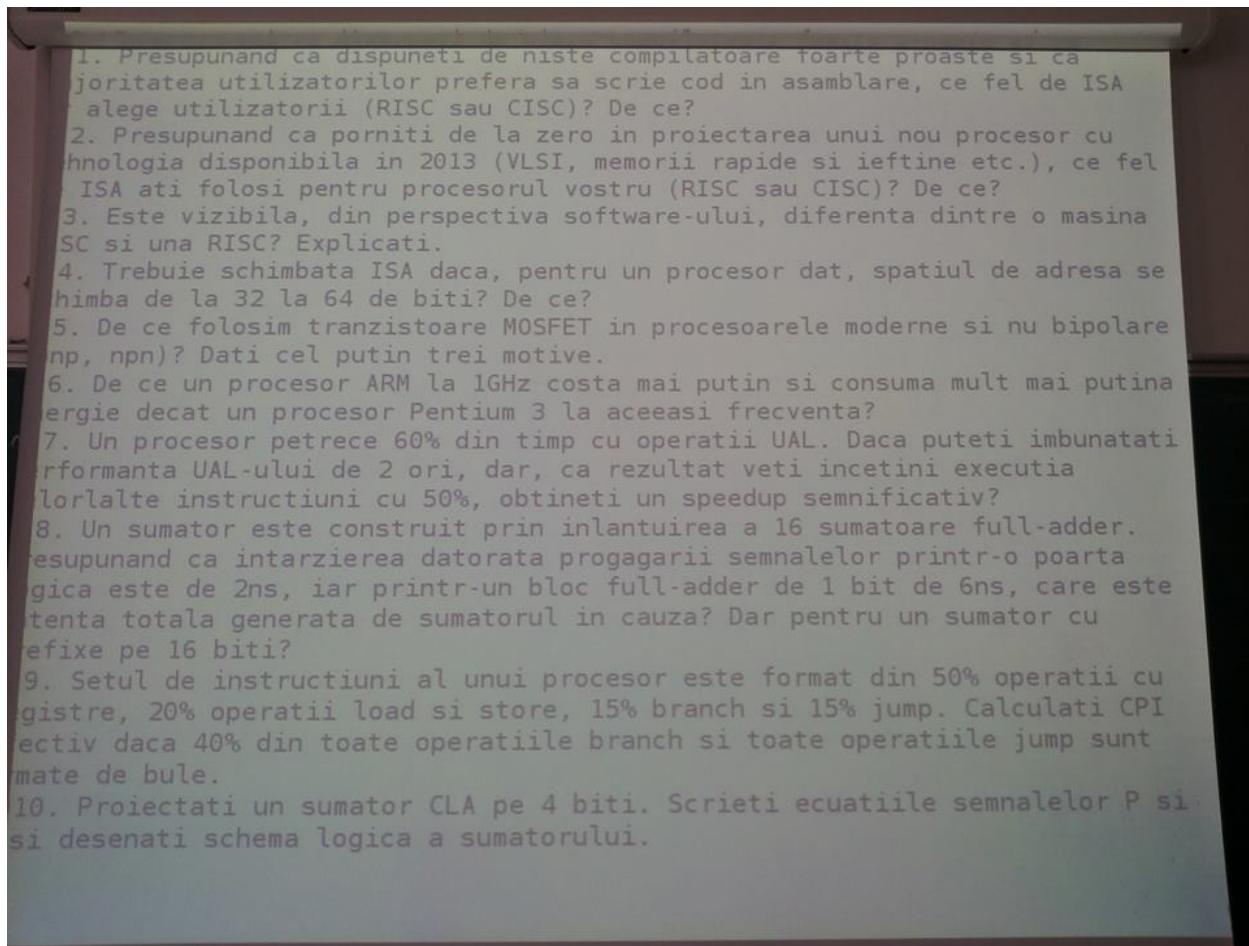


Tentativa de a face aceste subiecte nășpa să pară mai puțin nășpa :/



1. CISC, deoarece compilatorul va depune mai puțin efort să traduca cod high-level în cod de assembly, dacă cineva scrie în asta ceva. În plus, codul va semăna mult cu un cod într-un limbaj high-level, deci va fi mai ușor să se scrie programe complexe.

"The attitude at the time was that hardware design was more mature than [compiler design](#) so this was in itself also a reason to implement parts of the functionality in hardware or microcode rather than in a memory constrained compiler (or its generated code) alone. After the advent of RISC, this philosophy became retroactively known as complex instruction set computing, or CISC."

2. Today, the Intel x86 is arguably the only chip which retains CISC architecture. This is primarily due to advancements in other areas of computer technology. The price of RAM has decreased dramatically. In 1977, 1MB of DRAM cost about \$5,000. By 1994, the same amount of memory cost only \$6 (when adjusted for inflation). Compiler technology has also become more sophisticated, so that the **RISC** use of RAM and emphasis on software **has become ideal**.

For more info: <http://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/riscisc/>

3. Nu cred ca e vizibila din perspectiva software-ului de nivel inalt, dar e vizibila din perspectiva compilatorului si a celor ce scriu in assembly.

4. (wiki) **x86-64** este o extensie a setului de instrucțiuni **x86**. Adaugă suport pentru registre generale pe 64 de biți, mai multe adrese de memorie virtuală și numeroase alte îmbunătățiri. Asadar, trebuie extins ISA-ul.

5.

- comuta mai repede
- rezistenta mai *cumva* (dar *cumva* de bine)
- genereaza mai putin zgomot (cica, aparent se folosesc bipolare la amplificatoare, tocmai pentru ca *nu* genereaza zgomot; e dubios)
- consuma mai putin (??)
- sunt controlate in tensiune, nu in curent (ca bipolarele)! In procesoare, tensiune = valoare logica, deci MOSFET all the way - cred ca asta e.

6.

ARM este RISC, deci mai putine tranzistoare, deci mai ieftin si mai power-efficient.

Pentium este CISC, deci mai multe tranzistoare, deci mai scump si mai power-hungry.

+ Curs 2, slide 29 cu modalitatea de productie. Cilindrul de siliciu- la ARM sunt bucati cu suprafete mai mici de siliciu si pierderile sunt mai mici in schimb la Pentium suprafetele sunt mai mari si pierderile sunt mai mari si astfel baietii baga in pret :D. (numar de patratele intr-un cerc)

7. $\text{timp} = 0.6 * 0.5 + 0.4 * 1.5 = 0.9 < 1$, deci e un speed-up cvasi-semnificativ. (Pe bune, cat inseamna semnificativ?)

si speedup-ul = $1/(0.6*0.5 + 0.4 * 1.5) = 1/0.9 = 1.11$

8. ripple-carry: $16 * 6\text{ns} = 96\text{ns}$ (?)

din cursul 5 - slide 9, intreleg ca un ripple carry are $2k$ porti pt k biti

=> Tripple = $32 * 2\text{ns} + 16 * 6\text{ns} = 160\text{ns}$?

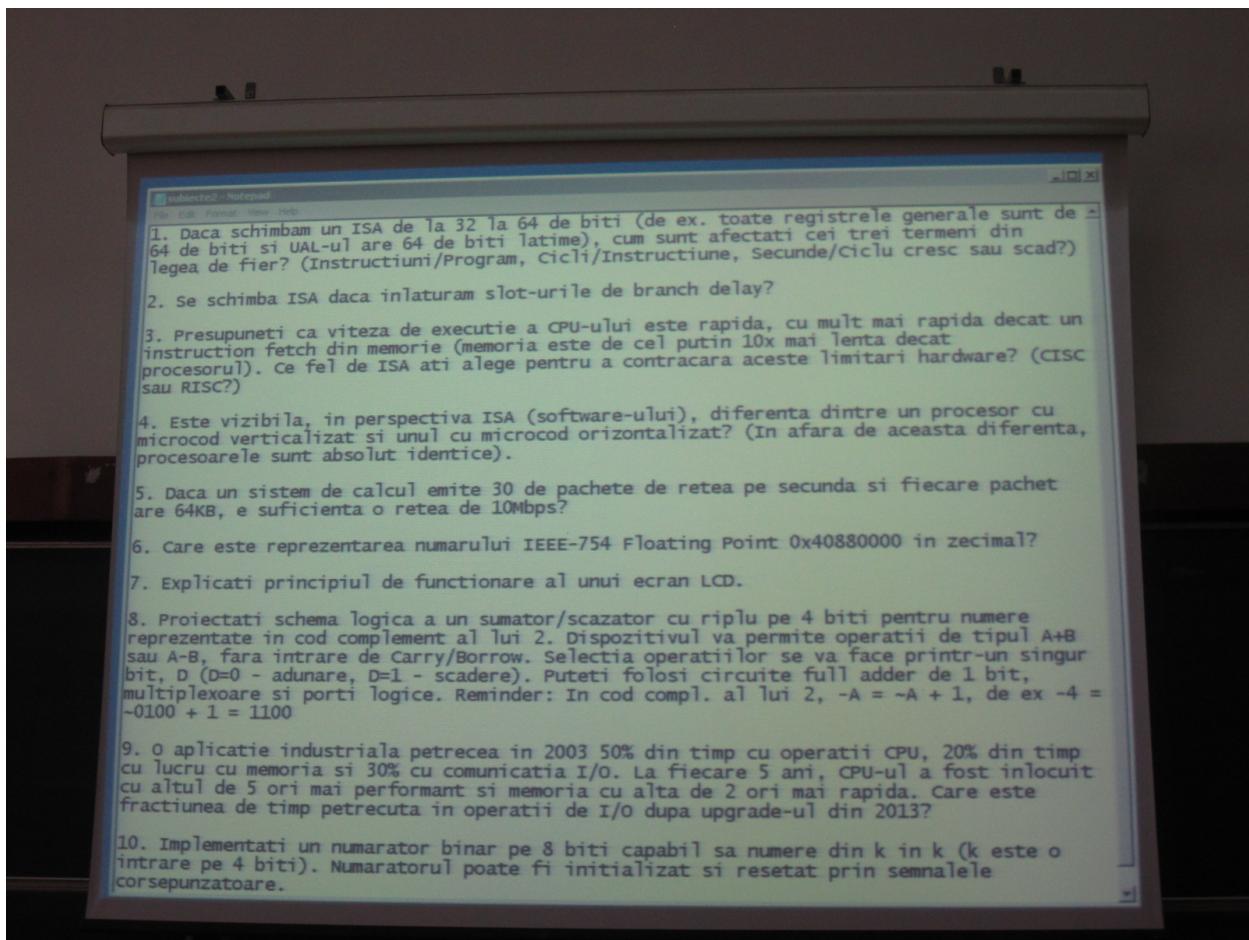
prefixe: ?!

$\text{RC} = 16 * 6 = 96\text{ns}$

Prefixe = $2\text{ns} + \log_2(16 * 4) \text{ ns} + 2\text{ns}$

9. nu ne mai trebuie si alte informatii?,
Nu, CPI = 1 + 0.4 * .15 + .15 = 1.21(curs 8 slide 58)

10. gasim in lab:



1. instr/prog si secunde/ciclu cred ca raman la fel, dar nu stiu ce sa zic de cicli/inst

"Instructions per program depinde de codul sursă,
compilator și ISA

- Cycles per instructions (CPI) depinde de ISA și
parchitectură
- Time per cycle depinde de parchitectură și tehnologia de
bază în care e implementat procesorul

ce intelegh eu de aici e ca instructions / program si cpi s-ar modifica

2. Da, daca nu avem branch delay irosim 1-2 instructiuni ce puteau fi executate, punand in locul lor NOPs

3. CISC, deoarece se pot executa operatii direct pe memorie -> mai putine memory fetch-uri

4. Nu cred, din cate intelegh, microcodul tranzlateaza instructiunile in operatii de circuite, in afara de faptul ca cel orizontal poate fi mai rapid, nu cred ca se observa vreodata.

de la wikipedia citire "Microcode is generally not visible or changeable by a normal programmer, not even by an [assembly](#) programmer."

5. Nu. $30/\text{s} * 64\text{KB} = 15360\text{Kbps} = 15\text{Mbps} > 10\text{Mbps}$

6. 4.25

0x40880000 - se transforma fiecare cifra in baza 2

0100 0000 1000 1000 0000 0000

primul bit e semnul, adica 0 \Rightarrow pozitiv

urmatorii 8 biti sunt exponentul + 127

adica 1000 0001 = 129 \Rightarrow exponentul = $129 - 127 = 2$

rezultul reprezinta mantisa, adica 000 1000 0000 0000 la care se adauga un 1 in fata

$\Rightarrow 1000 1000 0000 0000$ adica $1.000 1000 0000 0000 * 2^{\text{exponent}}$

$\Rightarrow 100.01 = 4.25$

7. Este în laborator, era ceva de gen că el are un buffer de caractere statice (hard-coded) ținute ca o matrice care corespunde unei coloane a LCD-ului. Revin după ce mă uit mai bine prin lab

8. Asta nu era deja făcută printr-un lab ?

9.

$$x + y + z = s, x = 0.5s, y = 0.2s, z = 0.3s$$

$$x/25 + y/4 + z = s'$$

$$0.5/25 s + 0.2/4 s + 0.3 s = s' \Rightarrow s' = 0.37 s$$

$$z = 0.3 s$$

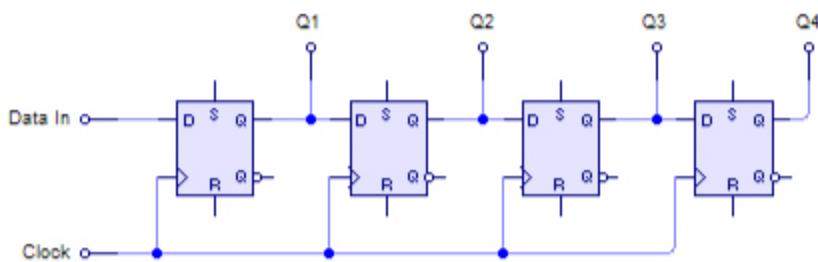
$$\Rightarrow z / s' = 0.3 / 0.37 = 0.81 \Rightarrow 81\%$$

Sau altfel $30 / (30 + 50/25 + 20/4) = 81\%$

Subiecte CN ziua 2 restante - **ATENTIE! FOARTE PRECISE!!! =))**

1. Proiectati un registru de shiftare cu incarcare seriala si descarcare paralela, ce trebuie modificat pentru a-l transforma in serial in / serial out? (nu mai stiu exact)
2. O problema cu legea lui Amdahl.
3. Ce se poate intampla dupa un numar mare de intreruperi imbriicate? Cum se rezolva de obicei aceasta situatie?
4. E vizibila din perspectiva ISA diferenta RISC CISC?
5. Dati un exemplu de hazard load/store si un hazard de control.
6. Pasii executati de procesor in tratarea unei intreruperi.
7. Se da nr. 0x98 in hexa, virgula fixa, cu semn, in cod complement fata de 2 si 4 biti pt partea fractionara. Transformati in zecimal.
8. O problema cu periferice si cat la suta petrece procesorul cu fiecare periferic.
9. O problema de tipul dacă un sistem de calcul emite 30 de pachete de retea pe secunda si fiecare pachet are 64 KB, e suficientă o retea de 10 Mbps?
10. O problema cu procente date de instructiuni urmatoare in diferite procente de bule si trebuie calculat CPI efectiv.

1.



6. Răspundem la cererea de întrerupere prin semnalul IntAck
Notificăm procesorul că avem o întrerupere de tratat
Setăm masca de întreruperi a.î. noua întrerupere să fie demascată

Se intrerup toate instructiunile aflate in procesare, se salveaza starea curenta, se procezeaza intreruperea si se revine in starea initiala.

(1p) Tocmai v-ati angajat la Google (pe un salariu considerabil) si prima voastră sarcină este să asigurați protecția celor mai de preț tehnologii experimentale dezvoltate de companie, care sunt tinute într-un depozit secret. Pentru aceasta va trebui să construiți un sistem care nu deschide usa depozitului decât dacă este introdus codul corect. În pauza de pranz ati facut deja majoritatea muncii, tot ce mai ramane de facut este logica de control.

Dispozitivul care controlează usa este bazat pe FPGA și are 10 intrări de la 10 butoane (unul pentru fiecare cifră: **button0**, **button1**, ..., **button9**), o intrare **reset** (în cazul în care ai introdus codul gresit) și o ieșire **open_door** (care deschide usa). Scrieti un modul Verilog care să controleze acest dispozitiv și să aibă urmatoarea funcționalitate:

- modulul așteaptă combinată corectă pentru a deschide usa (o succesiune de apasări, nu mai multe în același timp)
- starea initială a modulului este: nu a fost apasat niciun buton până acum și usa este închisă
- butonul **reset** aduce modulul în stare initială oricând ar fi apasat
- dacă un buton este apasat atunci intrarea corespunzătoare lui este 1, altfel este 0 (ex: butonul 0 neapasat => **button0 == 0**, butonul 0 apasat => **button0 == 1**)
- **open_door == 0** => usa este închisă, **open_door == 1** => usa este deschisă
- dacă, în orice punct al introducerii codului, se apasă un buton gresit modulul nu deschide usa orice să apasă și trebuie apasat **reset**
- dacă a fost introdus codul corect atunci usa va ramane deschisă orice să apasă, iar pentru a închide usa trebuie apasat **reset**
- codul pentru care modulul va deschide usa este **260384**

Exemplu de funcționare: codul este 1234, se introduce 1235... => usa nu se deschide, se introduce 01234... => usa nu se deschide, se introduce 1234 => usa se deschide, se introduce 12345... => usa se deschide

Precizări:

- 0.5p graful starilor + 0.5p modulul Verilog
- codul pe care îl scrieți nu trebuie să fie sintetizabil, ci poate fi un pseudo-Verilog, atât timp cât este în mare corect
- nu ezitați să folosiți comentarii ca să ne ajutați să înțelegem ce ati facut
- intrările și ieșirile modulului sunt date ca atare, nu trebuie să va ocupeți de debouncing sau legarea lor la un port al FGPA
- seful vostru e cam dur (și cheală)

