

Cum să treci la Moro*

- explicat prost de Pavel -

Cuprins

1	Introducere	1
2	Procesorul	2
3	Limbaaj de asamblare	2
4	Linux	3
5	Întrebări existențiale ale lui Moro	6

Acest document este făcut în mare parte pentru scopuri de divertisment.
Până și în curs apare „cmoro” - pag 8 (sau 152) - cap5.pdf

1 Introducere

Sistemul de calcul = Componenta Hardware + Componenta Software + Sistem de intreruperi

Sistem de calcul: Chestie care face calcule - nume pedantic pentru calculator.

Componenta Hardware: micro procesor, memorie, busuri, module IO¹ etc.

Componenta Software: Aplicație Sistem de operare, alte aplicații.

Componenta Hardware fara Sistem de operare = caramidă care nu face nimic.

Sistem de intreruperi: pe scurt legatura dintre hardware și software

Moduri de operare pentru aplicații:

text - MS-DOS, Linux

grafic - Windows, Linux

- Linux le face pe amândouă, deci e mai bun dăă - Profu'

Memoria poate fi

principală: RAM (Random Access Memory), ROM (Read Only Memory) - rapida dar nevolatilă în cazul RAM (moare dacă moare și curentul)

secundara: HDD (Hard Disk Drive) - lentă dar volatilă (nu moare decat dacă bagi burghiul în ea)

Din **motive istorice** procesorul Intel 8086 (pentru care a fost inițial creat DOS) împarte memoria în segmente calculând adresele astfel (și profu vrea să știm asta):

$$\text{Adresa fizica} = \text{Adresa segment} \cdot 16 + \text{Adresă Efectivă}$$

*Probabil

¹Intrare ieșire

2 Procesorul

Elementul principal al unui Sistem de calcul este Procesorul:

Procesor - bucata de silicon (și altele) care transformă numere binare în instrucțiuni executate².

Procesorul e format din:

UC (Unitate de control) - chestia care determină ce instrucțiuni să execute procesorul

UAL (Unitate aritmetică și logică) - chestia care calculează

Registrii - chestii pentru stocarea rezultatelor imediate - mai rapide decât RAM - importante în limbajul de asamblare

Cum facem calculatorul sistemul de calcul să facă ce vrem

Procesorul înțelege limbaj mașină care este interpretat ca instrucțiuni care arată cam așa:

6E6F7468696E6720746F207365652068657265500B241B402CD21

Pentru că este greu de înțeles pentru oameni, a fost creat Limbajul de asamblare, care este puțin mai rezonabil:

```
1 mov dl, 'A'
2 mov ah, 2
3 int 21h
```

Acest „Limbaj de asamblare” transformat în limbaj mașină de către un asamblor (cum ar fi TASM pentru DOS).

Mai există și alte limbaje de programare, cum ar fi C++, care sunt transformate în limbaj de asamblare (și apoi în limbaj mașină) dar asta nu-i important pt profu.

Interpretoare

În loc să transformăm un limbaj în altul mai „aproape de mașină”, putem rula într-un interpretor:

Interpretor: program care face pașii făcuți de un microprocesor (fetch - decode - execute), dar în software, nu hardware.

Un avantaj e că instrucțiunile înțelese de interpretor pot fi mai simple. Alt avantaj ar fi că putem rula același program pe mai multe arhitecturi fără a-l compila (lucru care poate dura **mult** timp³).

De exemplu javascript (ăla care face să apară 12 reclame la 2 secunde după ce intri pe un site) este rulat într-un interpretor.

3 Limbaj de asamblare

Cum asamblăm

Pe DOS⁴ exista un program debug care ne permite să „depanăm” executabile⁵. În Debug putem (deși e foarte limitat) crea programe în limbaj de asamblare.

Tot în DOS⁶ putem folosi TASM pentru a converti un fișier .asm într-unul .com, care poate fi rulat de DOS.

²La fel ca Cuzzi și Pavel care transformă băuturi nespecificate italiene (care nici macar nu sunt făcute în Italia) în cod LaTeX

³vreo 3 ore pe un procesor normal pentru clang

⁴că e mai simplu - dăă

⁵pe scurt - să vedem de ce nu merg

⁶heh

Cum arată

Pe scurt limbajul de asamblare e format din instrucțiuni simple⁷ cu nume scurte⁸ care sunt de forma:

```
1  label: mnemonic [argument[, argument]]
```

Printre cele mai importate instrucțiuni se numără:

`mov a, b` - „mută” conținutul din `b` în `a`
`add a, b`, `sub a, b` - echivalentul în c++ pt `a += b`; , respectiv `a -= b`;
`cmp a, b` - compara pe `a` cu `b`, modificand registrul „flags”, urmat de obicei de:
`jl label`, `jle`, `jg`, `je`, ... - procesorul „sare” execuția dacă în comparația precedentă `a < b`, resp `a ≤ b`, `a > b`, `a = b`, ...
`int numar` - invoca o „întrerupere” - pe scurt apelam sistemul de operare, de exemplu pentru a afișa ceva pe ecran. DOS se folosește des `int 21h`.

4 Linux

Cum instalăm Linux: are profu vreo 35⁹ de documente care explica cum sa instalam Linux.

Lucrul în BASH

Dacă deschidem un terminal¹⁰ suntem întâmpinați de un prompter (chestie care așteaptă comenzi de la tine) care arata cam așa:

```
cmoro@syst-calcul: ~$
```

Sub Linux, deoarece **totul** este un fișier (inclusiv outputul si inputul comenzilor) putem face asta:

- redirecționarea outputului:

```
1  comandă > fișier
2  #de exemplu
3  ls > f
```

tot outputul comenzii este pus in fișierul menționat, conținutul vechi al fișierului (dacă există) fiind suprascris.

- concatenarea outputului redirecționat:

```
1  comandă >> fișier
2  #de exemplu
3  ls >> f
```

la fel ca mai sus, dar conținutul vechi este păstrat, outputul fiind concatenat.

- redirecționarea inputului

```
1  comandă < fișier
2  #de exemplu
3  cat < f
```

⁷except for when they arent

⁸ahem, VFNMSUBSD

⁹da, 35

¹⁰Butonul ăla rotund din stânga-sus (stânga aia: ←), cauți „Terminal”, sau **Ctrl+Alt+T**

fișierul `f` este folosit drept input pentru comandă - de obicei exista opțiuni mai bune.

- `pipe`¹¹

```
1 cmd1 | cmd2
2 #de exemplu
3 ls | head
```

outputul comenzii 1 este folosit ca input pentru comanda 2 - mai folositor decât pare.

- `command substitution`

```
1 bla `comanda`
2 #de exemplu
3 ls `echo '/'`
4 #este inlocuit cu
5 ls /
```

tot ce este între `` este evaluat separat și apoi înlocuit în comandă. Pentru că `echo '/'` are outputul „/”, „/” este înlocuit în comandă

- rularea în background

```
1 comandă&
2 #de exemplu
3 sleep 10&
```

- `home`

```
1 ~
2 #de exemplu
3 ls ~/Desktop
```

Un nume mai scurt pentru `/home/numele tău aici/`

Comenzi

În acest prompter putem introduce comenzi cum ar fi:

- `cd dir` - schimbă directorul¹² curent.
- `pwd` - afișează directorul curent - destul de nefolositor că-ți arată deja BASHul unde ești.
- `ls dir` - afișează conținutul directorului.
- `mkdir dir` - face un director
- `cp src dst` - îl copiaza pe `src` în `dst`
- `mv src dst` - îl mută pe `src` în `dst`
- `rm file` - șterge fișierul; `rm -r dir` pentru directoare
- `ln -s src dst` - crează un link simbolic¹³ către `src`. este indicat să folosim căi de acces complete (aka `/Desktop/f.txt`, nu `f.txt`)
- `cat f1 f2 ...` - concatenează conținutul fișierelor - folosit cel mai des doar pentru a afișa conținutul unui fișierelor
- `tee f1` - afișa doar conținutul unui fișier
- `pr, fmt, lp` - nefolositor din 2005 încolo
- `wc f1` - afișează câte linii, cuvinte și câți biți sunt în fișier

¹¹nu am găsit o denumire bună în română

¹²aka folder

¹³un fel de shortcut

`diff f1 f2` - afisează diferențele dintre cele 2 fișiere
`sort f1` - sortează liniile fișierului, dăă :))
`cut f1` - selectează coloane din fișiere
`paste f1, f2, ...` - pune pe fiecare linie conținuturile fișierelor f1 ..., separate printr-un tab
`head f1 -n` - selectează primele n linii din fișier
`tail f1 -n` - selectează ultimele n linii din fișier
`chmod +x f1` - face fișierul f1 executabil

Now what?

Dacă nu știți ce face o comandă, scriți¹⁴ în prompter

```
1 man comanda
2 #sau
3 comanda --help
```

Dacă se blochează o comandă (bucă infinite sau ceva) apăsați `ctrl-c`

Editare text

Din terminal putem folosi editorul `nano` (din care ieșim cu `ctrl+x`), sau `gedit` pentru un editor grafic:

```
1 nano fisier
2 #sau
3 gedit fisier&
```

Sau cu redirectionarea outputului:

```
1 echo "Ana are mere" > fisier
```

Mai putem deschide fișierul și cu dublu click ca oamenii plictisitori.

Scripturi¹⁵ și alte executabile

În Linux putem crea scripturi¹⁶

```
1 echo "Bla bla comenzi..." > f.sh
2 bash f.sh
```

Sau putem face fișierul executabil:

```
1 echo "#!/bin/bash
2   Bla bla comenzi...
3   " > f1
4 chmod +x f1
5
6 ./f1
```

Pentru a crea executabile putem folosi și C++:

¹⁴moro - 2020

¹⁵nu, nu din alea

¹⁶comenzi BASH puse într-un fișier astfel:

```

1 echo "#include <iostream>
2   Bla bla c++
3   " > f.cpp
4
5 g++ f.cpp -o executabil
6
7 ./executabil

```

Numere de la 1 la 100¹⁷ intr-un fișier

```

1 echo '
2 for i in {1..100}; do
3   echo "$i"
4 done' > f.sh
5 bash f.sh > fisier
6
7 #sau: (va trebuie un fisier gol)
8 for i in {1..100}; do
9   echo $i >> fisier
10 done

```

5 Întrebări existențiale ale lui Moro

Î: Ce fel de probleme sunt rezolvabile cu calculatorul¹⁸?

R: Toate¹⁹ - dacă nu acu, atunci în viitor - deși pot lua mult timp.

Î: Ați instalat SO²⁰ Linux?

R: Da. Tre să știe el tot?

Î: Cum putem crea procese în SO Linux?

R: În mai multe feluri²¹.

Î: Ce face comanda x (îți da ceva complicat cu multe argumente)?

R: Intri pe explainshell.com și-ți zice

¹⁷inlocuiți 100 cu cât va trebuie

¹⁸nu-l citați cu asta

¹⁹înafara de unele chestii teoretice cum ar fi [The Halting Problem](http://en.wikipedia.org/wiki/Halting_Problem)

²⁰a se citi Sî O

²¹Mai exact din modul grafic, în C/C++ cu **fork**, **execl** și prietenii, în BASH unde pot fi în background sau nu

Î: How it feels to use Linux

R:

