Cum să treci la Moro*

- explicat prost de Pavel -

Cuprins

1 Introducere 1

2 Procesorul 2

3 Limbaj de asamblare 2

4 Linux 3

5 Întrebari existențiale ale lu' 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 IO[1] 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 in cazul RAM (moare daca moare si curentu)
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
[1]Intrare ieșire

2 Procesorul

Elementul principal al unui Sistem de calcul este Procesorul:

Procesor - bucata de silicon (și altele) care care transforma numere binare în instrucțiuni executate[2].

Procesorul e format din:

UC (Unitate de control) - chestia care determina ce instrucțiuni sa execute procesorul

UAL (Unitate aritmetica și logică) - chestia care calculeaza

Registrii - chestii pentru stocarea rezultatelor imediate - mai rapide decat 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:

```
6E6F7468696E6720746F207365652065726500B241B402CD21
```

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 in limbaj de asamblare (si apoi in limbaj masina) dar ăsta nu-i important pt profu.

Interpretoare

In loc sa transformăm un limbaj în altul mai „aproape de mașină", putem rula într-un interpretor:

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

Un avantaj e că instructiunile intelese de interpretor pot fi mai simple. Alt avantaj ar fi că putem rula același program pe mai multe arhitecturi fară a-l compila (lucru care poate dura **mult** timp[3]).

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

3 Limbaj de asamblare

Cum asamblam

Pe DOS[4] exista un program debug care ne permite să „depanăm" executabile[5]. În Debug putem (deși e foarte limitat) crea programe in limbaj de asamblare.

Tot în DOS[6] putem folosi TASM pentru a converti un fișier .asm într-unul .com, care poate fi rulat de DOS.

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

[3] vreo 3 ore pe un procesor normal pentru clang

[4] că e mai simplu - dăă

[5] pe scurt - să vedem de ce nu merg

[6] heh

Cum arată

Pe scurt limbajul de asamblare e format din instrucțiuni simple[7] cu nume scurte[8] care sunt de forma:

```
label: memonic [argument[, argument]]
```

Printre cele mai importate instructiuni se numără:

`mov a, b` - „mută” conținutul din b în a

`add a, b`, `sub a, b` - echivalentul în cpp 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 daca în comparația precedentă a $<$ b, resp a $\leq$ 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 35[9] de documente care explica cum sa instalam Linux.

Lucrul în BASH

Daca deschidem un terminal[10] 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:

```
comandă > fișier
#de exemplu
ls > f
```

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

- concatenarea outputului redirecționat:

```
comandă >> fișier
#de exemplu
ls >> f
```

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

- redirecționarea inputului

[7] except for when they arent
[8] ahem, VFNMSUBSD
[9] da, 35
[10] Butonul ăla rotund din stânga-sus (stânga aia: ←), cauți „Terminal”, sau `Ctrl+Alt+T`

```
comandă < fișier
#de exemplu
cat < f
```

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

- pipe[11]

```
cmd1 | cmd2
#de exemplu
ls | head
```

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

- command substitution

```
bla `comanda`
#de exemplu
ls `echo '/'`
#este inlocuit cu
ls /
```

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

- rularea in background

```
comandă&
#de exemplu
sleep 10&
```

- home

```
~
#de exemplu
ls ~/Desktop
```

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

Comenzi

In acest prompter putem introduce comenzi cum ar fi:

`cd dir` - schimbă directorul[12] 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

[11] nu am găsit o denumire bună în română
[12] aka folder

`ln -s src dst` - crează un link simbolic[13] catre 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 încoace

`wc f1` - afișează cate linii, cuvinte și câți biți sunt in fișier

`diff f1 f2` - afisează diferentele dintre cele 2 fișiere

`sort f1` - sortează liniile fisierului, dăă :))

`cut f1` - selectează coloane din fișiere

`paste f1, f2, ...` - pune pe fiecare linie continuturile fisierelor f1 ..., separate printr-un tab

`head f1 -n` - selectează primele n linii din fisier

`tail f1 -n` - selectează ultimele n linii din fisier

`chmod +x f1` - face fișierul f1 executabil

Now what?

Dacă nu știți ce face o comanda, scrit[14] in prompter

```
man comanda
#sau
comanda --help
```

Daca se blocheaza o comanda (bucla infinita sau ceva) apasați `ctrl-c`

Editare text

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

```
nano fisier
#sau
gedit fisier&
```

Sau cu redirecționarea outputului:

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

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

Scripturi[15] și alte executabile

În Linux putem crea scripturi[16]

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

Sau putem face fișierul executabil:

[13] un fel de shortcut
[14] moro - 2020
[15] nu, nu din alea
[16] comenzi BASH puse într-un fișier astfel:

```
echo "#!/bin/bash
  Bla bla comenzi...
  " > f1
chmod +x f1

./f1
```

Pentru a crea executabile putem folosi și C++:

```
echo "#include <iostream>
  Bla bla c++
  " > f.cpp

g++ f.cpp -o executabil

./executabil
```

Numere de la 1 la 100[17] intr-un fișier

```
echo '
for i in {1..100}; do
  echo "$i"
done' > f.sh
bash f.sh > fisier

#sau: (va trebuie un fisier gol)
for i in {1..100}; do
  echo $i >> fisier
done
```

5 Întrebari existențiale ale lu' Moro

Î: Ce fel de probleme sunt rezolvabile cu calculatorul[18]?
R: Toate[19] - daca nu acu, atunci in viitor - deși pot lua mult timp.

Î: Ați instalat SO[20] Linux?
R: Da. Tre să știe el tot?

Î: Cum putem crea procese în SO Linux?
R: În mai multe feluri[21].

Î: Ce face comanda x (îți da ceva complicat cu multe argumente)?
R: Intri pe explainshell.com și-ți zîce

[17] inlocuiți 100 cu cat va trebuie
[18] nu-l citați cu asta
[19] inafara de unele chestii teoretice cum ar fi The Halting Problem
[20] a se citi Sî O
[21] Mai exact din modul grafic, in C/C++ cu `fork`, `execl` și prietenii, în BASH unde pot fii în background sau nu

Î: How it feels to use Linux
R: