Tutoriat 3 - Arhitectura sistemelor de calcul

Stan Bianca-Mihaela, Stăncioiu Silviu

November 2020

1 Formatul intern in virgula mobila

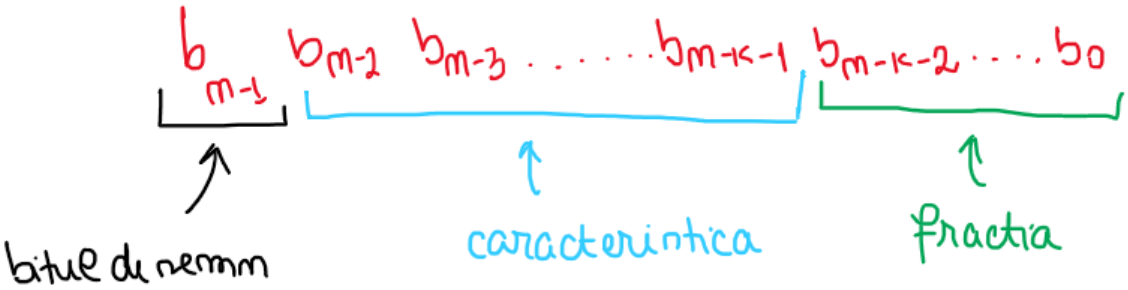
Ce vom invata in tutoriatul acesta? Care este reprezentarea interna a numerelor cu virgula (fara perioada).

Folosim standardul "IEEE 754".
In primul rand, avem 2 dimensiuni:

- n = numarul total de biti

- $2k \leq n-2$, numarul de cifre pe care le are caracteristica in reprezentarea interna

Cum interactioneaza k si n in reprezentarea numarului?



In functie de acesti 2 parametri se mai definesc:

- p = $n-k$ (precizia = numarul de cifre din mantisa + 1)
- mantisa, ρ, a carei lungime este p-1 biti
- exponentul minim : $E_{min} = -2^{k-1}+2$
- exponentul maxim sau BIAS : $E_{max} = 2^{k-1}-1$
- caracteristica : c = E + BIAS => $1 \leq c \leq 2^k - 2$

Fiecare numar va avea un E calculabil (vedem imediat cum il calculam). In functie de acesta, numerele se impart in 3 categorii:

- nereprezentabile: $E > E_{max}$
- cu reprezentare normalizata: $E \in [E_{min}, E_{max}]$
- cu reprezentare denormalizata : E< E_{min}

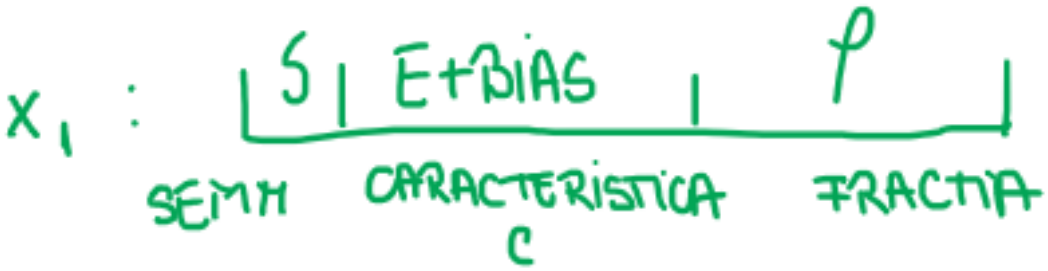
NOTATIE STIINTIFICA vs. NOTATIE STIINTIFICA NORMALIZATA

- O reprezentare in virgula mobila este notatie stiintifica daca are o singura cifra inainte de virgula.
- O reprezentare in virgula mobila este in notatie stiintifica normalizata daca are o singura cifra inainte de virgula si aceasta este nenula.

Cum arata aceste reprezentari in virgula mobila?

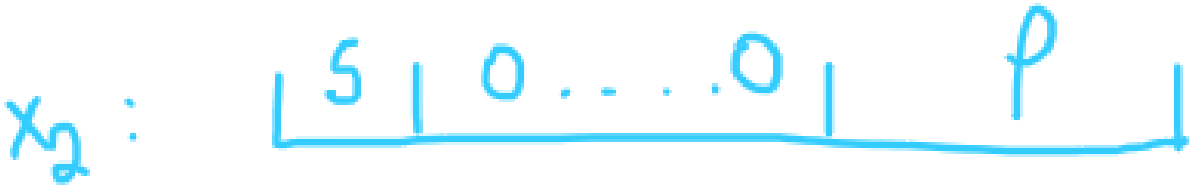
1.1 FORMATUL NORMALIZAT

$x_1 = (-1)^s * 2^E * \overline{1,\rho}$
unde s = 0 daca x_1 e pozitiv si 1 daca x_1 e negativ



1.2 FORMATUL DENORMALIZAT

$x_2 = (-1)^s * 2^{E_{min}} * \overline{0,\rho}$ cu $\rho! = 0$



1.3 ± 0

$x_3 = (-1)^s * 2^{E_{min}} * \overline{0,0}$

1.4 $\pm \infty$

1.5 NaN (not a number)

2 Tipuri de formate

Acum ca stim ca n si k snt variabilele cheie (in functie de ele se definesc toate celelalte) putem sa identificam anumite tipuri de formate:

2.1 Formatul single - pentru numere normalizate

Numarul trebuie sa apartina intervalului $[2^{-126}, 2^{127}]$.

- n=32
- k=8 => $E_{max} = BIAS = 2^{k-1} - 1 = 127$ si $E_{min} = -126$
- p=n-k=24 => $|\rho| = 23$ (numatul de cifre al mantisei)

2.2 Formatul double - pentru numere normalizate

Numarul trebuie sa apartina intervalului $[2^{-1022}, 2^{1023}]$.

- n=64
- k=11 => $E_{max} = BIAS = 2^{k-1} - 1 = 1023$ si $E_{min} = -1022$

- p=n-k=53 => $|\rho| = 52$ (numatul de cifre al mantisei)

Exemplul 1 : Transformati numarul 9,75 in format single.

Suntem in formatul single, deci n=32, k=8 si p=24.
x=9,75
E clar ca x $\notin \{\pm 0, \pm\infty, NaN\}$.
9=$(\overline{1001})_2$
0,75*2=1,50
0,50*2=1
am ajuns la 0, ne oprim
=> $9,75 = (\overline{1001,11})_2$

Ce facem mai departe? Noi vrem sa avem doar un singur 1 inainte de virgula. Deci mutam virgula cu 3 pozitii mai in fata.
$1001,11 = 1,00111 * 2^3$
Identificam acum variabilele:

- Cine e E? E=3 (puterea la care e 2 cand numarul este normalizat)
- Cine e mantisa? 00111 (ce e dupa virgula)
- Ce valoare are s, bitul de semn? Semnul lui x e pozitiv, deci s=0
- Cine este caracteristica, c? c=E+BIAS=3+127=130

Inainte sa scriem numarul in reprezentarea sa interna, mai avem de facut 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax] <=> E \in [-126, 127]$
  In cazul nostru, E=3, -126$\leq$ 3 $\leq$ 127 deci se verifica. ✓
- TESTUL DE ROTUNJIRE
  lungimea mantisei $\leq p - 1$
  In cazul nostru, lungimea mantisei este $|\rho| = 5$ si $5 \leq 23$ ✓

Vreau acum sa scriu reprezentarea interna a lui x in formatul single.

s: 0 ($x > 0$)
c: $(130)_2 = \overline{10000010}$ (daca c nu avea k=8 cifre, completam cu 0-uri)
ρ: 001110...0 (am completat cu 0-uri la final pana s-au facut 23 de cifre)

In final, x: 01000001000111000000000000000000 (reprezentarea lui 9,75 intern, pe 32 de biti)
Daca ne cere sa il scriem si in hexa?
x: 0100 0001 0001 1100 0000 0000 0000 0000
Stim ca 4 cifre din binar inseamna o cifra din hexa.
=>x:411C0000 (HEXA)

Exemplul 2 : Transformati numarul -37,125 in format double.

Suntem in formatul double, deci n=64, k=11 si p=53.
x=-37,125
x $\notin \{\pm 0, \pm\infty, NaN\}$
37=$(\overline{100101})_2$
0,125*2=0,250
0,250*2=0,5
0,50*2=1,0
am ajuns la 0, ne oprim

$=> -37,125 = (\overline{100101,001})_2$
Ce facem mai departe? Noi vrem sa avem doar un singur 1 inainte de virgula. Deci mutam virgula cu 5 pozitii mai in fata.
$100101,001 = 1,00101001 * 2^5$
Identificam acum variabilele:

- Cine e E? E=5 (puterea la care e 2 cand numarul este normalizat)
- Cine e mantisa? 00101001 (ce e dupa virgula)
- Ce valoare are s, bitul de semn? Semnul lui x e negativ, deci s=1
- Cine este caracteristica, c? c=E+BIAS=3+1023=1025

Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax] <=> E \in [-1022, 1023]$
  In cazul nostru, E=3, -1022$\leq 5 \leq$ 1023 deci se verifica ✓
- TEST ROTUNJIRE
  lungimea mantisei $\leq p - 1$
  In cazul nostru, lungimea mantisei este $|\rho| = 5$ si $5 \leq 52$ ✓

Vreau acum sa scriu reprezentarea interna a lui x in formatul double.

s: 1 ($x < 0$)
c: $(1025)_2 = \overline{10000000001}$ (daca c nu avea k=11 cifre, completam cu 0-uri)
ρ: 001010010...0 (am completat cu 0-uri pana s-au facut 52 de cifre)

In final, x: 1100000000010010100100000000000000000000000000000000000000000000 (reprezentarea lui -37,125 intern, pe 64 de biti)
Daca ne cere sa il scriem si in hexa?
x: 1100 0000 0001 0010 1001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
Stim ca 4 cifre din binar inseamna o cifra din hexa.
=>x:C012900000000000 (HEXA)

Exemplul 3 : Transformati numarul -0,125 in format particular: n=8, k=3.

Calculam valorile:

- p=n-k=5
- lungimea mantisei = $|\rho|$=$p - 1$=4
- $E_{min} = -2^{k-1} + 2 = -2$
- $E_{max} = BIAS = 2^{k-1} - 1 = 3$

x=-0,125
x $\notin \{\pm 0, \pm\infty, NaN\}$
0,125*2=0,250
0,250*2=0,5
0,50*2=1,0
am ajuns la 0, ne oprim
$=> -0,125 = (\overline{0,001})_2$
Ce facem mai departe? Noi vrem sa avem un 1 inainte de virgula. Deci mutam virgula cu 3 pozitii mai in spate.
$0,001 = 0,001 * 2^{-3}$
Identificam acum variabilele:

- Cine e E? E=-3 (puterea la care e 2 cand numarul este normalizat)
- Cine e mantisa? 0 (ce e dupa virgula)
- Ce valoare are s, bitul de semn? Semnul lui x e negativ, deci s=1
- Cine este caracteristica, c? c=E+BIAS=-3+3=0

Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax] <=> E \in [-2, 3]$
  In cazul nostru, E=-3 deci nu se verifica => avem un format denormalizat.
  Crestem E la E_{min} => $x = \overline{-0,1} * 2^{-2}$
- TEST ROTUNJIRE
  lungimea mantisei $\leq p - 1$
  In cazul nostru, lungimea mantisei este $|\rho| = 1$ si $1 \leq 4$

Vrem acum sa scriem reprezentarea interna a lui x in formatul particular.

s: 1 ($x < 0$)
c: 000 (are 3 cifre pentru ca k=3, si toate sunt 0 pentru ca suntem in format denormalizat)
ρ: 1000 (am completat cu 3 0-uri pentru ca lungimea mantisei trebuie sa fie 4)

In final, x: 10001000 (reprezentarea lui -0,125 intern, pe 8 de biti)
Daca ne cere sa il scriem si in hexa?
x: 1000 1000
Stim ca 4 cifre din binar inseamna o cifra din hexa.
=>x:88 (HEXA)

Exemplul 4: [RESTANTA SEPT 2020] Interpretati ca numar in baza 10 reprezentarea interna hexa in format single. 0xC1C80000.

0x din fata inseamna ca este reprezentarea in hexa. => $x : C1C80000(HEXA)$
C=$\overline{1100}$
1=$\overline{0001}$
C=$\overline{1100}$
8=$\overline{1000}$
0=$\overline{0000}$

Reprezentarea in binar a lui x: 1100 0001 1100 1000 0000 0000 0000 0000.

Identificam cine sunt s, c si ρ.
s: 1 (primul bit din reprezentarea in binar)
c: 10000011 (urmatoarele 8 cifre pentru ca stim ca lungimea lui c este k si k=8 in format single)
ρ: 100 1000 0000 0000 0000 0000 (restul cifrelor din reprezentarea in binar)

Observam ca c$\notin$ 0...0, 1...1 care corespund lui ± 0, respectiv $\pm\infty$. => suntem in format normalizat.

$x = (-1)^s * 2^E * \overline{1, \rho}$
Cine este E? Stim ca c=E+BIAS, iar BIAS in single este 127.
=> Transform E din baza 2 in baza 10.

$(\overline{10000011})_2^{-1} = 1 * 2^0 + 1 * 2^1 + 1 * 2^7 = 1 + 2 + 128 = 131$
$=> E = c - BIAS = 131 - 127 = 4$
Mai departe transform si ρ din baza 2 in baza 10.

$(0,1001)_2^{-1} = 1/2 + 1/16 = 9/16 = 0,5625$
Acum avem toate datele, putem sa inlocuim in x.
$x = (-1)^1 * 2^4 * \overline{1,1001} = -16 * 1,5625 = -25$

Exemplul 5 ADUNAREA IN VIRGULA MOBILA

[RESTANTA MAI 2020] Calculati 82,375 + (-1,75) folosind algoritmul de adunare in virgula mobila pentru formatul single (se va lucra cu reprezentarile matematice in baza 2 in notatie stiintifica, iar in final sa va converti rezultatul in baza 10).

x=82,375
y=-1,75
Il transformam pe x in baza 2:
82|0
41|1
20|0
10|0
5|1
2|0
1|1
0, ne oprim

$0,375 * 2 = 0,750$
$0,75 * 2 = 1,5$
$0,5 * 2 = 1,0$
0, ne oprim

$=> (82,375)_2 = \overline{1010010,011}$

Il transform si pe y in baza 2:
$0,75 * 2 = 1,5$
$0,5 * 2 = 1,0$
0, ne oprim

$=> (-1,75)_2 = -\overline{1,11}$

Eu le vreau pe ambele in notatie stiintifica:
$=> x = \overline{1010010,011} = \overline{1,010010011} * 2^6$
Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax] <=> E \in [-126, 127]$
  In cazul nostru, E=6, -126$\leq$ 6 $\leq$ 127 deci se verifica. ✓

- TESTUL DE ROTUNJIRE
  lungimea mantisei $\leq p-1$
  In cazul nostru, lungimea mantisei este $|\rho| = 9$ si $9 \leq 23$ ✓

=> $y = \overline{1,11} = \overline{1,11} * 2^0$
Facem cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax]$ <=> $E \in [-126, 127]$
  In cazul nostru, E=0, -126$\leq 0 \leq$ 127 deci se verifica. ✓
- TESTUL DE ROTUNJIRE
  lungimea mantisei $\leq p-1$
  In cazul nostru, lungimea mantisei este $|\rho| = 2$ si $2 <= 23$ ✓

Vrem acum sa facem adunarea dintre x si y. Ca sa putem sa facem asta este nevoie ca x si y sa aiba acelasi E.

0< 6 => trebuie sa aduc exponentul lui y la nivelul lui x
=> y devine: $0,00000111 * 2^6$

Acum putem face adunarea cu numarul negativ <=> scaderea:

1	,0	1	0	0	1	0	0	1	1
0	,	0	0	0	0	1	1	1	0
1	,0	1	0	0	0	0	1	0	1

=> $x + y = \overline{1,010000101} * 2^6$

Facem din nou cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax]$ <=> $E \in [-126, 127]$
  In cazul nostru, E=6, -126$\leq 6 \leq$ 127 deci se verifica. ✓
- TESTUL DE ROTUNJIRE
  lungimea mantisei $\leq p-1$
  In cazul nostru, lungimea mantisei este $|\rho| = 9$ si $9 \leq 23$ ✓

Deci rezultatul este: $\overline{1,010000101} * 2^6 = \frac{1010000101}{2^9} * 2^6 = \frac{645}{8} = 80,625$

Exemplul 6 INMULTIREA IN VIRGULA MOBILA

Inmultiti in format single x=7,75 si y=$-0,5$.

$x_2 = \overline{111,11} = \overline{1,1111 * 2^2}$
$y = -\overline{0,1} = -\overline{1} * 2^{-1}$
Facem verificariele:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax] <=> E \in [-126, 127]$
  In cazul nostru, E=2 si E=-1, $-126 \leq 2 \leq 127$, -126$\leq -1 \leq 127$ deci se verifica. ✓

- TESTUL DE ROTUNJIRE
  lungimea mantisei $\leq p - 1$
  In cazul nostru, lungimea mantisei este $|\rho| = 4$ si $4 \leq 23$, $|\rho| = 0$ si $0 \leq 23$ ✓

Observam ca nu mai trebuie sa aducem exponentii la aceeasi valoare ca la adunare.

Acum putem sa afcem inmultirea:
$$\begin{array}{ccccccc} & 1 & ,1 & 1 & 1 & 1 & \\ & 1 & ,0 & 0 & 0 & 0 & \\ \hline & 1 & ,1 & 1 & 1 & 1 & \end{array} => x * y = -\overline{1,1111} * 2^{2-1} = -\overline{1,1111} * 2^1$$

Facem din nou cele 2 verificari:

- TESTUL DE OVERFLOW / UNDERFLOW
  $E \in [E_{min}, Emax] <=> E \in [-126, 127]$
  In cazul nostru, E=1, -126$\leq 1 \leq 127$, deci se verifica. ✓

- TESTUL DE ROTUNJIRE
  lungimea mantisei $\leq p - 1$
  In cazul nostru, lungimea mantisei este $|\rho| = 4$ si $4 \leq 23$ ✓

Deci rezultatul este:
$-\overline{1,1111} * 2^1 = -\frac{\overline{11111}}{2^4} * 2^1 = -\frac{31}{8} = -3,875.$

3 MIPS - Tablouri unidimensionale (de elemente întregi)

APPLE LANSEAZA LAPTOPURI CU PROCESARE FACUTE DE EI PE ARHITECTURA ARM

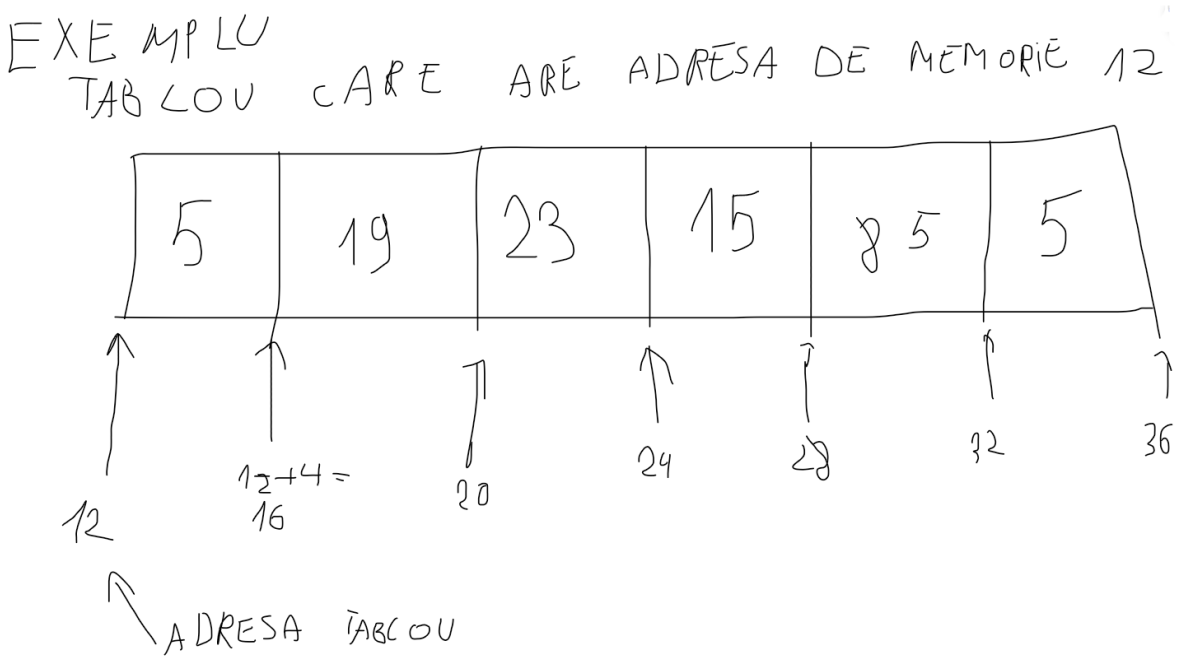
3.1 Declarare

Se declară asemănator cu datele de tip .word din tutoriatul trecut, atât că acum se scrie și lista de valori inițiale după, separare prin virgulă.
Exemplu declarare:
v:.word 5, 19, 23, 15, 85

n:.word 5

3.2 Indexare și adrese de memorie

Tablourile noastre declarate sunt practic o zonă de memorie în calculator unde se află valorile noastre. Noi putem afla un număr care ne spune exact unde se află în memorie începutul tabloului nostru (numit adresa de memorie). Dacă citim 4 bytes (adică un word) care se află la acea adresă de memorie (numărul nostru), atunci vom obține exact numărul de pe prima poziție din tablou. Dacă citim ce se află la poziția numarul_nostru + 4, vom obține exact numărul de pe a doua poziție din tablou... and so on and so forth.



Exemplu

Să se încarce în registrul $t1 valoarea de pe a 3 a poziție din tabloul v.
la $t0, v # Luăm adresa de memorie a lui v și o punem în registrul t0
addi $t0, $t0, 8 # adunăm 4 de doua ori la adresa respectivă (primul 4 pentru
# a ajunge la poziția a doua, iar a doilea 4 pentru a ajunge la poziția a 3-a)
lw $t1, 0($t0) # aici ia valoarea care se află la adresa de memorie $t0 și o copiază
# în registrul $t1. (nu îi putem da direct lw $t1, $t0, pentru ca instrucțiunea
# lw vrea să primească o adresă de memorie pentru argumentul 2, nu un
#registru.)

Acum, presupunând că avem vectorul v declarat în prima parte a materialului, atunci registrul $t1 va avea valoarea 23.

Aceasta este una din metodele de a lucra cu tablouri, este cea mai "raw" metodă în care se lucrează exact cu adrese de memorie. Ce e de reținut e cum se ia valoarea numerică de la o adresă de memorie pe care o avem ținută într-un registru și cum se găsește adresa de memorie a unui element într-un tablou.

În continuare, o să facem același lucru, doar că printr-o metodă mai simplă, iar de acum, vom folosi această metodă pentru a lucra cu tablouri. Ne putem gândi că indexăm un array ca în alte limbaje de programare,

iar în cazul acesta, putem ignora adresa de început a array-ului și doar să luăm elementele de la un anumit index. Diferența dintre ce vom face acum și indexarea din celelalte limbaje de programare constă în faptul că nu vom putea direct să spunem poziția de unde vrem elementul, ci va trebui sa dăm poziția înmulțită cu 4 (deoarce un word ocupa 4 bytes). Așadar, ne putem gândi că primul element are indexul 0, al doilea are indexul 4, al treilea are indexul 8 și tot așa.

Folosind acest approach, putem accesa elementele de la o poziție dată în două moduri.

1. Dacă avem un indice care este o constantă.
Să zicem că vrem să copiem valoarea de la a 3-a poziție în registrul $t1, atunci vom face:

```
la $t0, v # Luăm adresa de memorie a lui v și o punem în registrul t0
lw $t1, 8($t0) # Luăm valoarea de pe a 3 a poziție și o copiem în registrul
# $t1. Obervăm că acest approach este foarte asemănător cu primul approach
# prezentat, singura diferență fiind că aici nu incrementăm adresa de
# memorie, ci îi spunem direct cu cât să incrementeze adresa de memorie
# atunci când încărcăm valoarea.
```

2. Dacă avem un indice care este ținut într-un registru.
Aceasta metodă o vom folosi cel mai frecvent (cel puțin acum la început).
Considerăm tot exemplul de mai sus, în care vrem sa copiem valoarea de pe a 3-a poziție în registrul $t1.

```
li $t0, 8 # punem în $t0 indexul pentru a 3-a poziție.
lw $t1, v($t0) # Luăm valoarea de pe a 3 a poziție și o copiem în registrul
# $t1. Obervăm că de această dată nu am mai luat deloc adresa de memorie a
# tabloului, ci pur și simplu atunci când am copiat valoarea în registrul $t1,
# i-am spus de unde să ia valoarea de la indexul nostru, adică din tabloul v.
```

Acum putem vedea asemănarea dintre această metodă și alte limbaje de programare. În C ce am facut acum s-ar traduce în: int t1 = v[t0]; (o diferenta de notat ar fi ca in C indexul nu este multiplu de 4), care seamănă cât de cât cu această ultimă scriere din MIPS.

Observație:

Toate cele 3 metode prezentate sunt echivalente. Adică ele îi spun calculatorul să ia o valoare de la o

adresă de memorie. Singurul lucru care diferă este scrierea. Dacă facem un calcul, ne va ieși că în toate cele 3 metode, argumentul 2 al lui lw are valoarea egală cu adresa de memorie a celui de-al treilea element. În primul caz avem 0 + valoarea_din_t0, unde în $0 avem valoarea adresei de memorie a lui v + 8. Deci această sumă dă: adresa de memorie a lui v + 8, adică fix adresa de memorie a celui de-al treilea element. În al doilea caz avem 8 + valoarea_din_t0, unde în $0 avem valoarea adresei de memorie a lui v. Deci această sumă dă: adresa de memorie a lui v + 8. În ultimul approach avem adresa de memorie a lui v + valoarea_din_t0, unde în $0 avem indexul pentru poziția a 3-a, adică 8. Deci suma este: adresa de memorie a lui v + 8. Deci toate cele 3 sume sunt egale.

Observație:

La ultimele 2 approach-uri putem avea si indici negativi (evident, care in modul sunt divizibili cu 4). Dacă de exemplu avem într-un registru $t0 adresa de memorie a lui n (declarat mai sus). Atunci -4($t0) ar fi adresa de memorie a ultimului element din tabloul v, adică 85. (altă observație care se face este că toate lucrurile din memorie se află unele după altele.)

3.3 Exerciții

Problema 1: Se dă un vector în memorie, Să se afișeze pe ecran elementele lui.

```
.data
    v:.word 5, 19, 25, 13, 8 # vectorul nostru din memorie
    n:.word 5                # lungimea vectorului
    sp:.asciiz " "
.text
main:
    lw $t0, n                # încarcă lungimea vectorului în registrul $t0
    li $t1, 0                # counterul nostru pentru loop
    li $t2, 0                # indicele de unde vom accesa vectorul
                             # nu îl folosim pe $t1 pentru asta
                             # deoarce avem nevoie ca indicele nostru pentru vector
                             # să fie multiplu de 4

    loop:
        beq $t0, $t1, exit

        lw $a0, v($t2)       # copiază elementul de la poziția $t2 din vector
                             # in registrul $a0 pentru a o afisa pe ecran
        li $v0, 1
        syscall

        la $a0, sp
        li $v0, 4
        syscall

        addi $t1, $t1, 1     # incrementează counterul
        addi $t2, $t2, 4     # incrementează indicele nostru pentru vector.
                             # este incrementat cu 4 bytes deoarece un word înseamnă
                             # 4 bytes

        j loop

    exit:
    li $v0, 10
    syscall
```

Observație: Ce se întâmplă dacă schimbăm valoarea lui n din 5 în 6? Încercați.

Problema 2: Se dă $n \in N$ stocat în memorie, Să se citească un vector de n elemente numere întregi.

```
.data
    v:.space 28             # 7 elemente, adică 28 de bytes
    n:.word 7
.text
main:

    lw $t0, n
    li $t1, 0
    li $t2, 0

    loop:
        beq $t0, $t1, exit

        li $v0, 5
        syscall
        sw $v0, v($t2)      # salvează valoarea citită in vectorul v
                            # la poziția $t2

        addi $t1, $t1, 1
        addi $t2, $t2, 4

        j loop

    exit:

    li $v0, 10
    syscall
```

Bonus: Să se și afișeze vectorul citit.

Problema 3: Se citesc $n \in N$ și un vector de n numere întregi. Să se afișeze pe ecran elementele pare prin două parcurgeri (una pentru citire și una pentru afișare).

```
.data
    v:.space 400            # vector de 100 de elemente (400 = 100 * 4)
    sp:.asciiz " "
.text
main:

    li $v0, 5
    syscall
    move $t0, $v0

    li $t1, 0
    li $t2, 0

    read:                   # citește vectorul (ca la problema anterioară)

        beq $t0, $t1, solve

        li $v0, 5
        syscall
        sw $v0, v($t2)

        addi $t1, $t1, 1
        addi $t2, $t2, 4

        j read

    solve:                  # afișează pe ecran

    li $t1, 0               # t1 va fi counterul nostru care va merge din 4 in 4 poziții
    li $t2, 0               # indexul curent din vector (merge tot din 4 in 4 poziții
                            # deci va fi incrementat cu 16, adica 4 * 4)

    li $t2, 0

    loop:

        beq $t0, $t1, exit  # daca am parcurs vectorul ne oprim

        lw $t3, v($t2)      # luăm valoarea din array la poziția noatră și o ținem in $t3

        rem $t4, $t3, 2     # calculăm restul imărțirii la 2 in $t4 pentru a verifica
                            # dacă numărul este par

        beq $t4, 0, par     # dacă e par, il vom afișa
        j cont              # dacă nu, continuăm loop-ul in mod nomrla
        par:                # dacă am ajuns aici, e clar că numărul e par

            move $a0, $t3   # copiază in a $a0 valoarea pară pentru a o afișa
            li $v0, 1
            syscall         # afișează valoarea

            la $a0, sp
            li $v0, 4
            syscall         # afișează spațiu intre rezultate

        cont:

            addi $t1, $t1, 1
            addi $t2, $t2, 4

            j loop

```

Obervație: Pentru a simula un if putem face în felul următor:

```
b__ ceva_condiție, label_if
j continuare # dacă condiția nu este adevărată, programul continuă în mod
# normal.
label_if:
# aici vom avea codul din if
continuare:
# aici va continua programul indiferent dacă s-a executat sau nu codul din if.
```

Problema 4: Se citește un număr natural n, $n \leq 200$, urmat de n numere întregi. Să se parcurgă vectorul (adică cele n numere întregi citite) din 4 în 4 poziții, iar atunci când este întâlnită una din valorile 1 sau 2, să se afișeze valorile de pe următoarele 3 poziții din vector. Când este întâlnită valoarea 99, programul trebuie să se oprească. Se garantează că inputul este corect. (adică nu te va pune sa afișezi pe ecran valori care sunt în afara vectorului)

Exemplu:

Input:
12
1
9
10
3
2
3
11
0
99
30
40
50

Output:
9 10 3
3 11 0

Explicație: Se citește prima oară numărul de elemente din vector, adică 12. După care se citesc 12 valori. Prima valaore din vector este 1, deci trebuie să afișăm următoarele 3 valori. Trecem la a 4-a poziție din vector, unde valoarea este 2, deci iar trebuie să afișăm următoarele 3 valori. Apoi vom ajungem la valoarea 99, deci programul trebuie să se oprească.

```
.data
    v:.space 800                # vector de 200 de elemente (800 = 200 * 4)
    sp:.asciiz " "
    nl:.asciiz "\n"             # pentru a afișa rânduri noi intre răspunsuri
.text
main:

    li $v0, 5
    syscall
    move $t0, $v0

    li $t1, 0
    li $t2, 0

    read:                       # citește vectorul (ca la problema anterioară)

        beq $t0, $t1, solve

        li $v0, 5
        syscall
        sw $v0, v($t2)

        addi $t1, $t1, 1
        addi $t2, $t2, 4

        j read

solve:                          # rezolvă problema

    li $t1, 0                   # t1 va fi counterul nostru care va merge din 4 în 4 poziții
    li $t2, 0                   # indexul curent din vector (merge tot din 4 în 4 poziții
                                # deci va fi incrementat cu 16, adica 4 * 4)

    loop:

        bge $t1, $t0, exit      # dacă am ieșit din vector, ne oprim (teoretic nu ar fi
                                # nevoie de această linie dacă avem garantat
                                # că există mereu un 99 care să ne oprească)

        lw $t3, v($t2)          # luăm valoarea din array la poziția noatră și o ținem în $t3

        beq $t3, 99, exit       # dacă am găsit 99, programul se oprește

        beq $t3, 1, afis_case   # dacă am dat de valoarea 1 afișăm următoarele 3 valori
        beq $t3, 2, afis_case   # dacă am dat de valoarea 2 afișăm următoarele 3 valori

        j cont_loop             # dacă am ajuns aici, nicio condiție nu este îndeplinită
                                # deci continuă loop-ul normal

        afis_case:              # dacă am ajuns aici, e clar că $t3 este egal fie cu 1, fie cu 2

            move $t4, $t2       # luăm poziția la care suntem in vector și o salvăm în $t4

            addi $t4, 4         # incrementăm poziția pentru a afișa elementul următor
            lw $a0, v($t4)      # încărcăm în $a0 valoarea elementului următor pentru a o afișa
            li $v0, 1
            syscall             # afișăm valoarea pe ecran

            la $a0, sp          # afișăm spațiu pe ecran
            li $v0, 4
            syscall

            addi $t4, 4         # facem la fel și pentru următorul număr care trebuie afișat
            lw $a0, v($t4)
            li $v0, 1
            syscall

            la $a0, sp
            li $v0, 4
            syscall

            addi $t4, 4         # la fel și pentru al treilea
            lw $a0, v($t4)
            li $v0, 1
            syscall

            la $a0, nl          # pune un rând nou
            li $v0, 4
            syscall

        cont_loop:

            addi $t1, $t1, 4    # continuă loop-ul, adică incrementează poziția curentă cu 4
            addi $t2, $t2, 16   # incrementează indexul cu 16. (4 * 4)

            j loop

exit:

    li $v0, 10
    syscall


```

Problema 5: Se citește un număr natural n, $n \leq 200$, urmat de n numere întregi. Să se parcurgă vectorul (adică cele n numere întregi citite) din 4 în 4 poziții, iar atunci când este întâlnită una din valorile 1 sau 2, se consideră următorile valori astfel: a = numărul de pe poziția următoare în vector, b = numărul care urmează dupa a, c = numărul care urmează după b. Dacă valoarea întâlnită în vector este 1, atunci să se realizeze următoarea operație din C: v[c] = v[a] + v[b];. Dacă valoarea întâlnită în vector este 2, atunci să se realizeze urmatoarea operație din C v[c] = v[a] * v[b];. Când se întâlnește valoarea 99, programul trebuie să se oprească. Înainte de a parcurge vectorul, să se pună în vector la poziția 2 valoarea 12, iar la poziția 3, valoarea 2. Se cere să se afișeze valoarea de la poziția 0 din vector după ce programul își termină executarea. Este garantat că datele de intrare sunt valide.

Exemplu (în care nu am schimbat valorile de pe pozitiile 2 si 3 din vector pentru a fi mai usor de ilustrat programul, in rezolvare nu trebuie omis acest pas):

Input:
12
1
9
10
3
2
3
11
0
99
30
40
50
Output:
3500

Explicație: Ca si data trecută prima oară este întâlnit 1, deci a = 9, b = 10, c = 3. Deoarece avem valoarea 1, facem adunarea numerelor de la pozițiile 9, respectiv 10, si o salvăm la poziția 3. Vom avea: v[3] = v[10] + v[9];, deci la a 3 a poziția 3 nu vom mai avea valoarea 3, ci valoarea v[10] + v[9], adica 30 + 40, adică 70. Acum întâlnim valoarea 2, deci a = 3, b = 11, c = 0. Deoarece avem valoarea 2, facem înmulțirea numerelor de la pozițiile 3, respectiv 11 și o salvăm la poziția 0. Vom avea v[0] = v[3] * v[11];, deci la prima poziție nu vom mai avea valoarea 1, ci valoarea v[3] * v[11], adică 70 * 50 = 3500. Întâlnim 99, deci ne oprim și afișăm valoarea de pe prima poziție, adică 3500.

```
.data
    v:.space 800
    sp:.asciiz " "
    nl:.asciiz "\n"
.text
main:

    li $v0, 5
    syscall
    move $t0, $v0

    li $t1, 0
    li $t2, 0

    read:

        beq $t0, $t1, solve

        li $v0, 5
        syscall
        sw $v0, v($t2)

        addi $t1, $t1, 1
        addi $t2, $t2, 4

        j read

solve:

    la $t1, v                # încarcă în $t1 adresa de memorie a vectorului
    li $t2, 12               # pune într-un registru valoarea 12
    sw $t2, 4($t1)           # pune pe a doua poziție din vector valoarea 12

    li $t2, 2
    sw $t2, 8($t1)           # pune pe a treia poziție din vector valoarea 2

    li $t1, 0
    li $t2, 0

    loop:

        bge $t1, $t0, exit

        lw $t3, v($t2)

        beq $t3, 99, exit

        beq $t3, 1, adunare   # acum avem 2 cazuri, unul pentru adunare, unul pentru înmulțire
        j cont1               # dacă nu a apărut 1, poate a apărut 2, deci mergem mai jos să verificăm

        adunare:

            move $t4, $t2     # obține cele 3 numere a, b, c ca în programul anterior
            addi $t4, $t4, 4
            lw $t5, v($t4)

            addi $t4, $t4, 4
            lw $t6, v($t4)

            addi $t4, $t4, 4
            lw $t7, v($t4)

            add $t5, $t5, $t5 # înmulțește poziția lui a cu 4 (pentru ca așa "indexăm" noi)
            add $t5, $t5, $t5 # doua adunări repetate înseamnă o înmulțire cu 4

            lw $t5, v($t5)    # obține valoarea din vector de la poziția a

            add $t6, $t6, $t6 # la fel cum am făcut pentru a, facem și pentru b
            add $t6, $t6, $t6

            lw $t6, v($t6)

            add $t5, $t5, $t6 # facem adunarea

            add $t7, $t7, $t7 # la fel, înmulțim poziția cu 4 pentru a putea
            add $t7, $t7, $t7 # accesa în vectorul nostru

            sw $t5, v($t7)    # salvează rezultatul la poziția c

            j cont2           # ne ducem la sfârșitul loop-ului pentru că
                              # știm că nu vom intra pe cazul în care
                              # valoarea este 2

        cont1:

        beq $t3, 2, inmultire # dacă valoarea citită e 2, este cazul pentru înmulțire

        j cont2               # dacă valoarea nu e nici 1, nici 2, atunci continuă loop-ul

        inmultire:            # codul pentru v[c] = v[a] * v[b], asemănător cu cel de adunare
                              # majoritatea codului se repetă, probabil s-ar putea face mai elegant
                              # desi atunci succesiunea de branch-uri ar devenit confuzantă
                              # deci, în scop didactic vom avea "copy pasta" aici.

            move $t4, $t2
            addi $t4, $t4, 4
            lw $t5, v($t4)

            addi $t4, $t4, 4
            lw $t6, v($t4)

            addi $t4, $t4, 4
            lw $t7, v($t4)

            add $t5, $t5, $t5
            add $t5, $t5, $t5

            lw $t5, v($t5)

            add $t6, $t6, $t6
            add $t6, $t6, $t6

            lw $t6, v($t6)

            mul $t5, $t5, $t6 # înmulțește cele două numere

            add $t7, $t7, $t7
            add $t7, $t7, $t7

            sw $t5, v($t7)

        cont2:

            addi $t1, $t1, 4
            addi $t2, $t2, 16

            j loop

exit:

    lw $a0, v                # ia valoarea de pe prima poziție și o afișează
    li $v0, 1
    syscall

    li $v0, 10
    syscall
```

Fun fact: Aceasta este prima parte din problema care a fost dată anul trecut la Advent of Code în cea de a doua zi. Acolo vectorul este dat ca numere separate prin virgulă. Pentru a ne fi ușor să interpretăm inputul în MIPS am schimbat metoda de input (n citit, urmat de alte n numbere). Pentru a converti inputul acestei probleme de AOC la input ușor de citit în MIPS, putem folosi următorul script în Python:

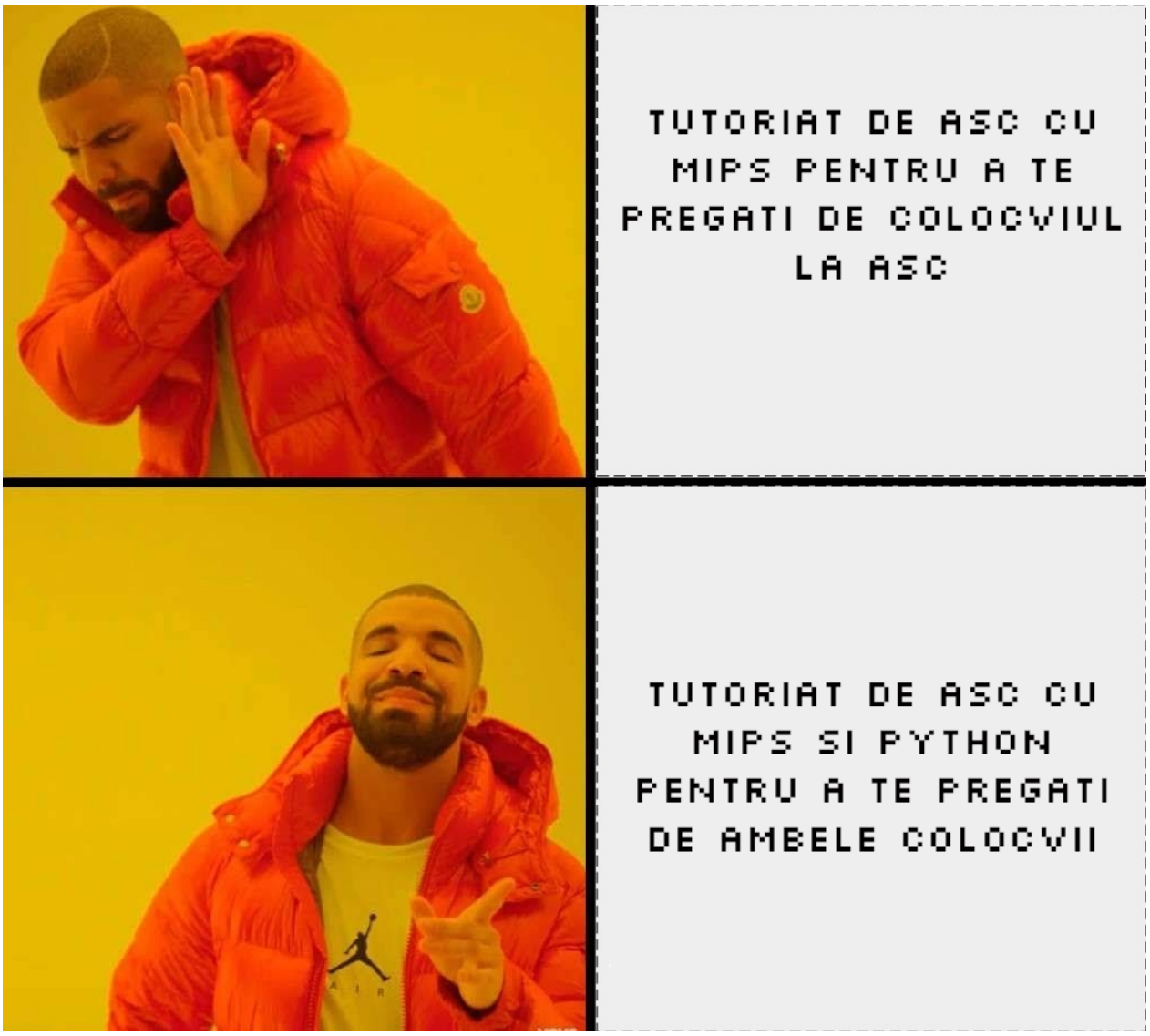
```python
inputfile = open("raw_input.txt")                          # deschide fisierul din care citim
outputfile = open("processed_input.txt", "a")              # deschide fisierul in care scriem

numbers = [int(x) for x in inputfile.readline().split(",")] # parseaza vectorul de numere

outputfile.write(str(len(numbers)) + "\n")                 # scrie numarul de elemente din vector

for number in numbers:                                     # parcurge array-ul
    outputfile.write(str(number) + "\n")                   # scrie numerele separate prin rânduri noi

outputfile.close()                                         # închide fișierul în care scriem
inputfile.close()                                          # închide fișierul din care citim
                                                           # la colocviu la PA să nu uitați să
                                                           # închideți fișierele
                                                           # cei care corectează au și lucrui de genul
                                                           # în vedere
```



Problema 6: Se dă o matrice declarată în memorie de forma:

```
2       a:.word 1, 2, 3, 4
3         .word 5, 6, 7, 8
4         .word 9, 10, 11, 12
5       n:.word 3
6       m:.word 4
```

Să se afișeze pe ecran.

```
.data
    a:.word 1, 2, 3, 4
      .word 5, 6, 7, 8
      .word 9, 10, 11, 12
    n:.word 3
    m:.word 4
    sp:.asciiz " "
    nl:.asciiz "\n"
.text
main:

    lw $t0, n                       # incarc in $t0 numărul de linii
    lw $t1, m                       # incarc in $t1 numărul de coloane

    li $t2, 0                       # conter pentru iterarea liniilor

    print:                          # "for" pentru linii

        beq $t2, $t0, exit

        li $t3, 0                   # counter pentru iterarea coloanelor

        print_line:                 # "for" pentru coloane

            beq $t3, $t1, end_line

            move $t4, $t2           # calculează poziția in matrice
                                    # poziția e dată de formula p = (l * m) + c
                                    # unde l este linia, iar c este coloana
            mul $t4, $t4, $t1
            add $t4, $t4, $t3

            add $t4, $t4, $t4       # inmulțim $t4 cu 4 pentru a fi poziție validă
            add $t4, $t4, $t4       # in tabloul nostru

            lw $a0, a($t4)          # incarc valoarea din tablou in registrul $a0
            li $v0, 1               # pentru a o afișa pe ecran
            syscall

            la $a0, sp
            li $v0, 4
            syscall

            addi $t3, $t3, 1
            j print_line

        end_line:

            la $a0, nl
            li $v0, 4
            syscall

            addi $t2, $t2, 1

            j print

exit:
```

3.4 Mai multe exerciții

Problema 1: Se citesc $n \in N^*$ și un vector de n numere naturale. Să se afișeze maximul și toate pozițiile pe care apare.

Problema 2: Se citesc $n \in N^*$ și doi vectori $v, w \in Z$ ordonați crescător. Să se interclaseze cei doi vectori într-un vector z și să se afișeze z.

References

[1] Dumitru Daniel Drăgulici. *Curs Arhitectura Sistemelor de Calcul.*

[2] Larisa Dumitrache. *Tutoriat 2019*

[3] Bogdan Macovei. *Laboratoare ASC 2019/ 2020*

[4] Advent Of Code. *Day 2 2019*