

Tutoriat 8

Stan Bianca-Mihaela, Stăncioiu Silviu

December 2020

Vreți ca la următorul tutoriat să continuăm cu recapitularea pentru seminar, sau să lucrăm niște probleme de MIPS date la teste de laborator în alți ani?

- Adăugat de tine MIPS is love, MIPS is life  38 voturi
- Adăugat de tine Gândesc ASC-ul în mod metafizic, neavând nevoie de pregătire adițională la acest capitol.  23 voturi
- Adăugat de tine Continuare seminar, lol  5 voturi

**CEI CARE AU ALES OPTIUNEA CU "GÂNDESC ASC-UL"
IN MOD METAFIZIC":**



1 MIPS is love, MIPS is life



1.1 Subiecte date de Bogdan Macovei

Exercițiul 1 Să se rezolve următorul model de test de laborator din 2019:

<https://drive.google.com/file/d/1Om0lDKOZjVsMIgSNo-qd5BV9YZ3Y886M/view?usp=sharing>

Rezolvare

Partea I

```
1 .data
2
3     v::space 400                      # vectorul pe care vom lucra
4
5 .text
6
7 suma_cifrelor_para:
8
9     subu $sp, 4                      # punem $fp pe stiva
10    sw $fp, 0($sp)                   # stiva este $sp: ($fp) (z) (
11        celealte_lucruri_care_sunt_deja_pe_stiva)
12
13    addi $fp, $sp, 4                  # face fp sa pointeeze catre cadrul nostru de apel
14    # avem $sp: ($fp) <fp_pointeaza_aici> (z) (
15        celealte_lucruri_care_sunt_deja_pe_stiva)
16
17    subu $sp, 4                      # pune $ra pe stiva
18    sw $ra, 0($sp)                   # stiva este $sp: ($ra) ($fp) <fp_pointeaza_aici> (z) (
19        celealte_lucruri_care_sunt_deja_pe_stiva)
20
21    subu $sp, 4                      # pune $s0 pe stiva
```

```

19    sw $s0, 0($sp)          # stiva este $sp: ($s0) ($ra) ($fp) <fp_pointeaza_aici>
20    (z) (celealte_lucruri_care_sunt_deja_pe_stiva)
21
22    subu $sp, 4             # pune $s1 pe stiva
23    sw $s1, 0($sp)          # stiva este $sp: ($s1) ($s0) ($ra) ($fp) <
24      fp_pointeaza_aici> (z) (celealte_lucruri_care_sunt_deja_pe_stiva)
25
26    lw $s0, 0($fp)          # in $s0 il vom avea pe z si il vom imparti succesiv la
27      10
28    li $v0, 0                # pana cand va avea valoarea 0
29      rezultatul final
30
31    parc_cifre:
32
33      beq $s0, 0, exit_parc_cifre # am ajuns la 0, deci numarul nu mai are cifre
34
35      rem $s1, $s0, 10          # in $s0 vom avea ultima cifra a numarului
36      add $v0, $v0, $s1          # adunam cifra la suma
37
38      div $s0, $s0, 10          # impartim numarul la 10, adica scoatem ultima lui cifra
39      # de la sfarsit
40
41      j parc_cifre           # parcurgem cifrele
42
43    exit_parc_cifre:         # am terminat de parcurs cifrele
44
45      rem $s0, $v0, 2          # in $s0 avem result impartirii sumei cifrelor la 2
46      # pentru a decide daca e par sau nu
47
48      seq $v0, $s0, 0           # daca suma e para, in $v0 pune 1, altfel 0
49
50      lw $s1, -16($fp)          # restauram registrii de pe stiva
51      lw $s0, -12($fp)          # adica (s1, s0, ra, fp)
52      lw $ra, -8($fp)
53      lw $fp, -4($fp)
54
55      addu $sp, 16              # da pop la stiva
56
57      jr $ra                  # ieșe din functie
58
59    evaluateaza:
60
61      subu $sp, 4              # punem fp pe stiva
62      sw $fp, 0($sp)          # stiva este $sp: ($fp) (pointer_catre_v) (n) (x) (y) (z)
63
64      addi $fp, $sp, 4          # face fp sa pointeeze catre cadrul nostru de apel
65      # avem $sp: ($fp) <fp_pointeaza_aici> (pointer_catre_v)
66      # (n) (x) (y) (z)
67
68      subu $sp, 4              # punem $ra pe stiva
69      sw $ra, 0($sp)          # stiva este $sp: ($ra) ($fp) (pointer_catre_v) (n) (x)
70
71      subu $sp, 4              # punem $s0 pe stiva
72      sw $s0, 0($sp)          # stiva este $sp: ($s0) ($ra) ($fp) (pointer_catre_v) (n)
73
74      subu $sp, 4              # punе $s1 pe stiva
75      sw $s1, 0($sp)          # stiva este $sp: ($s1) ($s0) ($ra) ($fp) (
76        pointer_catre_v) (n) (x) (y) (z)
77
78      subu $sp, 4              # punе $s2 pe stiva
79      sw $s2, 0($sp)          # stiva este $sp: ($s2) ($s1) ($s0) ($ra) ($fp) (
80        pointer_catre_v) (n) (x) (y) (z)
81
82      subu $sp, 4              # punе $s3 pe stiva

```

```

77    sw $s3, 0($sp)          # stiva este $sp: ($s3) ($s2) ($s1) ($s0) ($ra) ($fp) (
78      pointer_catre_v) (n) (x) (y) (z)
79
80    subu $sp, 4              # pune $s4 pe stiva
81    sw $s4, 0($sp)          # stiva este $sp: ($s4) ($s3) ($s2) ($s1) ($s0) ($ra) (
82      $fp) (pointer_catre_v) (n) (x) (y) (z)
83
84    subu $sp, 4              # pune $s5 pe stiva
85    sw $s5, 0($sp)          # stiva este $sp: ($s5) ($s4) ($s3) ($s2) ($s1) ($s0) (
86      $ra) ($fp) (pointer_catre_v) (n) (x) (y) (z)
87
88    lw $s0, 4($fp)          # in $s0 il vom avea pe n
89    li $s1, 0                # $s1 va fi counterul (adic i din formula)
90    lw $s2, 0($fp)          # $s2 va fi un pointer catre pozitia curenta din vector
91      # nu vom folosi un registru pentru a tine adresa de
92      # memorie
93      # a vectorului si un alt registru pentru counterul din
94      # vector.
95      # in schimb vom avea un signur registru cu adresa de
96      # memorie
97      # din vector in care ne aflam
98
99    li $s4, 0                # $s3 il vom folosi pentru uz general
100   # in $s4 vom acumula suma
101   # $s5 il vom folosi pentru uz general
102
103  parcurg:                 # parcurge vectorul
104
105  beq $s0, $s1, fin_ev
106
107  lw $s3, 0($s2)          # momentan, in $s3 vom tine valoarea lui v[i]
108
109  subu $sp, 4              # punem v[i] pe stiva pentru a apela procedura
110
111  sw $s3, 0($sp)          # stiva: $sp: (v[i]) (
112    celealte_lucruri_care_sunt_deja_pe_stiva)
113
114  jal suma_cifrelor_para  # apelam procedura suma_cifrelor_para
115
116  addu $sp, 4              # ii dam pop lui v[i] de pe stiva.
117
118  beq $v0, 0, ev_add_cond # daca in v0 am valoarea 1 nu ma intereseaza (pentru ca
119    avem (1-1)*ceva in formula,
120    # adica 0*ceva=0)
121
122  j ev_cont_parc
123
124  ev_add_cond:             # daca $v0 are valoarea 0, adunam a doua parte din
125    formula la suma         # adica $s4 += (v[i] mod x + (y - (z div 3) + i)^3)
126
127  lw $s5, 8($fp)          # in $s5 il punem pe x
128  rem $s5, $s3, $s5        # in $s5 punem v[i] mod x (momentan $s3 are tot valoarea
129    v[i])
130
131  add $s4, $s4, $s5        # adunam la suma pe v[i] mod x
132    # deci ne-a ramas de calculat doar (y - (z div 3) + i)^3
133
134  lw $s3, 12($fp)          # in $s3 il punem pe y
135  lw $s5, 16($fp)          # in $s5 il punem pe z
136  div $s5, $s5, 3           # in $s5 punem (z div 3)
137
138  subu $s3, $s3, $s5        # calculam x - (x div 3) si tinem valoarea in $s3

```

```

133      add $s3, $s3, $s1          # adunam i, deci avem  $x - (x \text{ div } 3) + i$  in $s3
134      move $s5, $s3             # copiem aceeasi valoare si in $s5, adica $s5 =  $x - (x \text{ div } 3) + i$ 
135
136      mul $s3, $s3, $s3          # in $s3 avem  $(x - (x \text{ div } 3) + i)^2$ 
137      mul $s3, $s3, $s5          # acum in $s3 avem  $(x - (x \text{ div } 3) + i)^3$ 
138
139      add $s4, $s4, $s3          # adunam  $(x - (x \text{ div } 3) + i)^3$  la suma
140
141      ev_cont_parc:
142
143      addu $s1, 1                # incrementez counterul
144      addu $s2, 4                # incrementez adresa de memorie curenta din vector
145
146      j parcurg                 # continui parcurgerea
147
148      fin_ev:
149
150      move $v0, $s4              # returnam prin $v0 rezultatul
151
152      lw $s5, -32($fp)          # restauram toti registrii de care ne-am folosit
153      lw $s4, -28($fp)          # adica ($s0..$s5, $fp, $ra)
154      lw $s3, -24($fp)
155      lw $s2, -20($fp)
156      lw $s1, -16($fp)
157      lw $s0, -12($fp)
158      lw $ra, -8($fp)
159      lw $fp, -4($fp)
160
161      addu $sp, 32              # dam pop de pe stiva acestor registrii
162
163      jr $ra                  # iesim din functie
164
165 main:
166
167      li $v0, 5                # citim n
168      syscall
169      move $t0, $v0
170
171      li $t1, 0
172      li $t2, 0
173
174      read:                   # citim cele n numere din vector
175
176      beq $t1, $t0, exit_read
177
178      li $v0, 5
179      syscall
180      sw $v0, v($t2)
181
182      addu $t1, 1
183      addu $t2, 4
184
185      j read
186
187 exit_read:
188
189      li $v0, 5                # citim x
190      syscall
191      move $t1, $v0
192
193      li $v0, 5                # citim y
194      syscall
195      move $t2, $v0
196
197      li $v0, 5                # citim z
198      syscall
199      move $t3, $v0

```

```

200
201    subu $sp, 4          # punem z pe stiva
202    sw $t3, 0($sp)
203
204    subu $sp, 4          # punem y pe stiva
205    sw $t2, 0($sp)
206
207    subu $sp, 4          # punem x pe stiva
208    sw $t1, 0($sp)
209
210    subu $sp, 4          # punem n pe stiva
211    sw $t0, 0($sp)
212
213    subu $sp, 4          # punem pointer catre v pe stiva
214    la $t0, v
215    sw $t0, 0($sp)
216
217
218    # stiva noastra este:
219    jal evaluateaza      # $sp: (pointer_catre_v) (n) (x) (y) (z)
220
221    addu $sp, 20         # dam pop la partea din stiva unde sunt argumentele
222                                # 5 (nr_de argumente) * 4 (sizeof word) bytes = 20
223
224    move $a0, $v0         # afiseaza pe ecran rezolutatul
225    li $v0, 1
226    syscall
227
228    li $v0, 10
229    syscall

```

problema1.macos

Partea II

1

Apelul de sistem READ STRING afisează un sir de caractere pe ecran. Argumentele sunt: adresa de memorie la care vrem să reținem sirul de caractere, respectiv lungimea maximă a sirului de caractere. Adresa de memorie se pune în registrul \$a0, iar dimensiunea maximă se pune în registrul \$a1. Codul pentru apelul de sistem este 8 și se pune în registrul \$v0. Rezultatul îl vom primi la adresa de memorie pe care i-am dat-o apelului de sistem.

Exemplu:

```

1 .data
2     str:.space 100 # sir de caractere de lungime 99
3             # lungimea este 99, nu 100
4             # deoarece ultimul caracter
5             # este '\0'
6 .text
7 main:
8
9     # citeste sirul de caractere
10    la $a0, str        # pune in $a0 adresa de memorie la
11        # care vreau sa retin sirul de caractere
12    li $a1, 99          # dimensiunea maxim a sirului
13    li $v0, 8            # codul pentru apelul de sistem este 8
14    syscall
15
16    # afiseaza sirul de caractere

```

```

17      la $a0, str
18      li $v0, 4
19      syscall
20          # exit
21      li $v0, 10
22      syscall

```

readstring.s

La test nu va fi nevoie să dați un exemplu aşa de mare, puteți doar să declarați un sir de caractere și să faceți apelul de sistem, nefiind nevoie să îl și afișați pe ecran. Exemplul acesta este luat cu copy-pasta din Tutoriatul 4.

2

Trebuie să aflăm reprezentarea internă a instrucțiunii add \$t0, \$t0, \$s3.

Avem:

op → 000000 (deoarece este instrucțiune din R)
 rs → \$t0 = \$8 = 01000
 rt → \$s3 = \$19 = 10011
 rd → \$t0 = \$8 = 01000
 shamt → 00000 (nu avem shiftare)
 func → 100000

Reprezentarea binară este:

0000 0001 0001 0011 0100 0000 0010 0000

Reprezentarea în hexa este:

0x01134020

3

Pentru a parcurge un sir de caractere putem avea un contor cu care să iterăm prin caracterele sirului de caractere. Pentru a extrage un caracter se poate folosi lb \$t0, sir(\$t1), unde \$t0 este destinația, \$t1 este contorul, iar *sir* este sirul de caractere sotcat la nivel de memorie.

O altă metodă este să ținem adresa de memorie din sir la poziția curentă, iar pe aceasta să o tot incrementăm, în loc să avem un contor separat. Vom avea la \$t0, sir. Asta ne va da adresa de memorie a sirului de caractere. La fiecare pas putem incrementa această adresă cu 1, iar pentru a extrage caracterul curent putem face lb \$t1 ,0(\$t0), unde \$t1 este destinația.

Cele două metode sunt echivalente întrucât amândouă se rezumă la a salva într-un registru valoarea de la o adresă de memorie. În prima metodă avem un contor relativ la adresa de memorie a sirului, iar în a doua metodă avem adresa directă din memorie către poziția din sir.

Mai multe explicații pe tema asta se găsesc în Tutoriatul 3.

ACEL MOMENT CAND SINGURELE
MATERIALE PE CARE LE AI LA
EXAMENUL DE ASC SUNT
MEME-URILE DIN TUTORIATE



1.2 Subiecte date de domnul Drăgulici



**ACEL MOMENT CAND CINEVA TE
INTREABA PE STRADA DE UNDE SA
CUMPERE UN COVRIG, IAR TU DIN
REFLEX CHIAR II SPUI DE UNDE SA
CUMPERE, CI NU IL PUI SA CAUTE
BRUTARII**



Exercițiu 1 Procedură ce primește ca parametrii prin stivă adresa unui string și lungimea lui și-i inversează ordinea caracterelor; în acest scop va parcurge stringul cu \$s0 de la început către sfârșit și cu \$s1 de la sfârșit către început și cât timp $\$s0 < \$s1$ va interschimba caracterele pointate de $\$s0,\$s1$. Procedura își va accesa parametrii cu \$fp iar apelurile vor respecta convențiile MIPS și C (privind cadrul de apel, \$fp, regiștrii salvați de apelant și apelat, etc.).

Program care aplică procedura unui string ".asciiz" declarat cu inițializare (se va inversa partea până la

caracterul nul exclusiv) și apoi îl afișează (cu "syscall").

Rezolvare:

```

1 .data
2
3     sir: .ascii "Va salut, dragi studenti." # sirul declarat in memorie
4
5 .text
6
7 invert:
8
9     subu $sp, 4
10    sw $fp, 0($sp)
11        lungime)
12
13    addi $fp, $sp, 4
14        # punem $fp pe stiva
15        # acum stiva este $sp: ($fp) (adresa_sir) (
16
17    subu $sp, 4
18    sw $ra, 0($sp)
19        lungime)
20
21    subu $sp, 4
22    sw $s0, 0($sp)
23        ) (lungime)
24
25    subu $sp, 4
26    sw $s1, 0($sp)
27        adresa_sir) (lungime)
28
29    lw $s0, 0($fp)
30        caractere
31    lw $s1, 4($fp)
32        caractere
33
34    add $s1, $s0, $s1
35        # in $s0 avem adresa catre inceputul sirului de
36        # in $s1 avem momentan lungimea sirului de
37
38    subu $s1, 1
39        # adunam la $s1 valoarea lui $s0 pentru a avea
40        # catre sfarsitul sirului de caractere
41        # academ 1 deoarece prin adunarea de mai sus
42
43        # pe care nu trebuie sa il luam in considerare
44        # in algoritmul de inversare
45
46    inv_loop:
47
48    bge $s0, $s1, sf_inv
49        # loop-ul care inverseaza
50        # daca $s0>=$s1 se opreste, adica se executa cat
51        # asa cum scrie in enunt
52
53    bne $s0, $s1, inv_loop
54
55    lb $t0, 0($s0)
56        # folosim $t0 si $t1 ca registrii auxiliari
57        # interschimbam valorile
58
59    lb $t1, 0($s1)
60
61    sb $t0, 0($s1)
62        # incrementam registrul $s0 (pentru ca el merge
63        # de la inceput spre sfarsit)
64
65    sb $t1, 0($s0)
66
67    addu $s0, 1
68        # de la sfarsit spre inceput)
69    subu $s1, 1
70        # decrementam registrul $s1 (pentru ca el merge
71        # de la inceput spre sfarsit)

```

```

46      j inv_loop          # continua algoritmul de inversare
47
48 sf_inv:
49
50    lw $s1, -16($fp)      stiva
51    lw $s0, -12($fp)      # restaureaza registrii care au fost pusi pe
52    lw $ra, -8($fp)       # adica ($s1, $s0, $ra, $fp)
53    lw $fp, -4($fp)
54
55    addu $sp, 16          # da pop la aceste registrii de pe stiva
56
57    j $ra                # procedura s-a terminat
58
59 main:
60
61    li $t0, 0              # vom afla lungimea sirului in registrul $t0
62
63    lb $t1, sir($t0)      aflarea
64
65    calc_lung:
66
67      beqz $t1, sf_calc_lung
68
69      addu $t0, 1            # daca am ajuns la \0 ne oprim
70      lb $t1, sir($t0)      # incrementeaza contorul
71
72      j calc_lung          # ia un caracter din sir
73
74
75 sf_calc_lung:
76
77      subu $sp, 4            # continua parcurgerea
78      sw $t0, 0($sp)        # pun adresa sirului de caractere pe stiva
79
80      la $t1, sir           # stiva este: $sp: (adresa_sir) (lungime)
81      subu $sp, 4            # apeleaza procedura care inverseaza sirul
82      sw $t1, 0($sp)
83
84      jal invert
85
86      addu $sp, 8            # da pop la argumentele de pe stiva
87
88      la $a0, sir           # afiseaza sirul cu syscall cum cere problema
89      li $v0, 4
90      syscall
91
92      li $v0, 10
93      syscall

```

problema1_dragus



References

- [1] Dumitru Daniel Drăgulici. *Curs+Laborator Arhitectura Sistemelor de Calcul.*
- [2] Stan Bianca-Mihaela, Stăncioiu Silviu *Tutoriat 2020, lol*
- [3] Larisa Dumitrache. *Tutoriat 2019*
- [4] Bogdan Macovei. *Laboratoare ASC 2019/ 2020*