Arhitectura sistemelor de calcul – Laboratorul 3

I. Siruri de caractere

Sirurile de caractere sunt tablouri unidimensionale unde toate elementele ocupa 1 byte (comparative cu tablourile dimensionale de numere intregi – elemente de tipul .word – unde fiecare ocupa 4 bytes).

Importanta este observatia ca, pentru sirurile de caractere, nu avem nevoie de dimensiunea lor, pentru ca stim ca se termina cu '\0' (cu byte-ul avand valoarea 0).

Pentru a prelua un caracter de la o adresa de memorie, vom utiliza instructiunea **lb $reg, mem** (incarcam in registrul **$reg** caracterul aflat la byte-ul indicat de **mem**). In acest caz, contorul care sare locatiile de memorie se deplaseaza cu 1, si nu cu 4.

Pentru a afisa caractere pe ecran (PRINT BYTE), vom utiliza codul de sistem 11. Cand sistemul este apelat cu syscall, gaseste in $v0 codul 11 si printeaza caracterul incarcat in registrul $a0. Urmatorul program afiseaza un caracter declarat in zona de memorie:

```
.data
        ch: .byte 'a'
.text
main:
        lb $a0, ch              # incarcam in $a0 byte-ul din memorie
        li $v0, 11              # apelam codul sistem 11 (PRINT BYTE)
        syscall                 # apelam sistemul

        li $v0, 10
        syscall
```

O parcurgere a sirului de caractere va fi urmatoarea (efectul va fi similar functiei strlen, in contorul care sare locatiile de memorie vom avea, in final, lungimea sirului):

```
.data
        str: .asciiz "Sir de caractere"     # sirul de caractere se declara cu .asciiz
.text
main:
        li $t0, 0                   # $t0 sare din 1 in 1 pe locatiile de memorie
        lb $t1, str($t0)            # $t1 este byte-ul curent din sir (la $t0 distanta de str)
loop:
        beqz $t1, exit              # daca $t1 (byte-ul curent) este 0, atunci se merge la exit
        addi $t0, 1                 # altfel sarim pe urmatoarea locatie din memorie
        lb $t1, str($t0)            # accesam elementul
        j loop                      # si revenim la inceputul structurii repetitive

exit:                               # la iesire, afisam pe ecran $t0 = dimensiunea sirului
        move $a0, $t0               # incarcam in $a0 pe $t0
        li $v0, 1                   # ii dam lui $v0 codul pentru PRINT INT
```

```
        syscal                  # apelam sistemul

        li $v0, 10
        syscall
```

Exercitiu: sa se afiseze, pe ecran, sirul de caractere fara spatii. De exemplu, dandu-se in memorie "Sir de caractere" sa se afiseze la output "Sirdecaractere".

Solutie:

```
.data
        str: .asciiz "Sir de caractere"     # declar sirul de caractere
        sp: .byte ' '                       # declar un byte de spatiu
.text
main:
        li $t0, 0                           # $t0 sare pe locatiile din memorie
        lb $t1, str($t0)                    # $t1 este byte-ul curent din sirul de caractere
        lb $t2, sp                          # $t2 este byte-ul de spatiu
loop:
        beqz $t1, exit                      # daca am ajuns pe byte-ul 0, se opreste executia
        bne $t1, $t2, afisare               # daca byte-ul curent ($t1) != spatiul ($t2), afisam
cont:                                       # nu uitam, dupa afisare, sa revenim in cadrul loop-ului
                                            # daca nu se respecta conditia ca $t1 != $t2, se intra
                                            # automat la eticheta cont
                                            # (daca nu se face salt, executia continua
                                            # (cu linia urmatoare)
        addi $t0, 1                         # sarim o locatie de memorie
        lb $t1, str($t0)                    # si accesam elementul din locatie
        j loop                              # si revenim la inceputul loop-ului

afisare:
        move $a0, $t1                       # la afisare, incarcam in $a0 byte-ul curent, din $t1
        li $v0, 11                          # dam codul de sistem 11 (PRINT BYTE)
        syscall                             # apelam sistemul
        j cont                              # revenim in loop pentru a trece la urmatorul byte

exit:
        li $v0, 10
        syscall
```

Exercitii propuse:

1. Sa se determine numarul de cuvinte dintr-un sir de caractere (stiind ca separatorii sunt doar spatiile).
2. Sa se determine cuvintele de lungime maxima. (se determina lungimea maxima a unui cuvant, apoi se afiseaza pe ecran cuvintele, cate unul pe linie)
3. Dat fiind un sir de caractere, sa se salveze intr-un alt sir de caractere doar elementele situate pe pozitii pare (indexarea se face de la 0). Sir de caractere -> Srd aatr

II. Proceduri MIPS

(laborator documentat dupa fisierele din arhiva dl. Dragulici)

In implementarea si utilizarea subrutinelor sunt folositi mai ales registrii $a0 - $a3 (pentru transmiterea parametrilor actuali), $v0, $v1 (pentru transmiterea valorii returnate, in cazul functiilor), $sp (pentru gestionarea stivei - pointeaza in permanenta varful stivei), $fp (pointeaza in stiva cadrul apelului curent de subrutina), $ra (contine adresa de intoarcere din apelul curent de subrutina).

Registrul PC contine mereu adresa instructiunii care urmeaza sa se execute; el este consultat/modificat indirect de instr. ca "jal", "jr", etc.

Stiva este o zona de memorie folosita pentru stocarea de valori temporare; in particular ea este folosita la gestionarea apelurilor de subrutina (pentru stocarea temporara a unor valori legate de aceste apeluri). Stiva este gestionata in maniera FIFO, datele fiind incarcate/descarcate la un acelasi capat, numit varful stivei.

Stiva creste spre adrese mici si scade spre adrese mari, iar registrul $sp are drept rol sa retina in permanenta adresa varfului stivei (a octetului din varful stivei). Astfel, putem incarca (push) un word din $t0 in stiva cu secventa:

```
subu $sp, 4
sw $t0, 0($sp)
```

si putem descarca (pop) word-ul din varful stivei in $t0 cu secventa:

```
lw $t0, 0($sp)
addu $sp, 4
```

Comenzi importante: apelul unei proceduri se face prin instructiunea **jal eticheta** (unde eticheta este numele procedurii)

jal eticheta

```
# salt si legatura;
# efectueaza: $ra <- adr. instr. urm. si apoi salt la eticheta;
#  adica: $ra <- PC + 4; PC <- eticheta;
```

Revenirea dintr-o procedura se face prin instructiunea **jr $ra** (jump to register, unde registrul la care ne intoarcem este $ra = return address).

jr rs

```
# salt la registru;
# efectueaza: salt la adresa din rs;
#  adica: PC <- rs;
```

Modul efectiv de lucru cu procedurile:

1. Vom utiliza registrii $sp, $fp si $s0-$s7, cu specificatiile $sp = stack pointer, va pointa intotdeauna varful stivei; $fp = frame pointer, va pointa in cadrul de apel acolo unde il fixam (ca o conventie, il vom fixa **intotdeauna** sa pointeze incepand de la argumentele functiei); $s0-$s7 sunt numiti **registri salvati, ei vor fi restaurati intotdeauna la finalizarea procedurii, inainte de jr $ra**. Registrii $s vor avea semnificatia variabilelor locale din functii.

2. Inainte sa facem un apel catre o procedura prin **jal eticheta**, vom incarca, in stiva, parametrii procedurii, **in ordine inversa** (din cauza stivei). Daca avem de scris functia f(int x, int y), ordinea de incarcare este y, apoi x.

3. In procedura, **intotdeauna vom aloca loc pentru $fp**, comportamentul de baza fiind urmatorul:
   - i se salveaza lui $fp valoarea actuala pe stiva; (este posibil ca $fp sa aiba o valoare la intrarea in procedura, sa presupunem ca $fp = 9. Trebuie ca, la finalul procedurii, indiferent de ce prelucrari am avut cu $fp, sa ii redam valoarea 9, astfel ca, salvam pe stiva valoarea lui initiala); aceasta este o conventie de C si nu o putem incalca;
   - $fp pointeaza cadrul de apel (va pointa argumentele pe care le dam, astfel incat primul argument sa fie 0($fp), al doilea 4($fp) etc.).

4. In procedura, daca avem variabile locale, **intotdeauna** le vom retine in registrii $s si **intotdeauna le vom restaura valorile**. Este analog ideii de mai sus, privind $fp: daca avem nevoie de $s0, initial $s0 poate contine o valoare calculata in alta parte de program. Noi utilizam $s0 asa cum vrem local, dar la iesirea din procedura, trebuie sa ii dam valoarea pe care a avut-o inainte. Un scenariu este cel al apelurilor imbricate: vreau in procedura **pozitie_maxim(int *v, int n)** sa apelez procedura **maxim(int *v, int n).** $s0-$s7 sunt registri care sunt utilizati ca variabile locale, inainte sa apelez maxim din **pozitie_maxim**, posibil sa am in $s0 o anumita valoare (indexul curent, de exemplu). In **maxim**, posibil ca in $s0 sa calculez altceva (sa am adresa de inceput a vectorului). Nu vreau ca in **pozitie_maxim** sa ma intereseze cu ce scop este utilizat $s0 in alta procedura pe care eu o apelez, vreau doar ca, la revenirea din procedura **maxim**, eu sa am in $s0 aceeasi informatie pe care o aveam inainte de apel.

5. Registrii care intorc rezultate de functii sunt $v0 si $v1, dar putem, de fapt, sa intoarcem oricate valori, prin stiva, si in general vom utiliza a doua varianta.

Pe scurt: incarc in stiva argumentele, apelez procedura, salvez pe stiva valorile vechi pentru $fp si registrii $s, scriu cod MIPS obisnuit, iar inainte de revenirea din procedura (jr $ra), nu uit sa restaurez $fp si registrii $s.

Mod de reprezentare a stivei: vom scrie, pe parcursul intregului program, modul in care arata stiva. Conventia de scriere va fi de a inlantui elementele de la stanga la dreapta (stanga fiind varful stivei), intre paranteze rotunde. Elementele care nu sunt intre paranteze, sunt pointerii (in general $sp si $fp, dar va fi cazul uneori sa scriem proceduri pentru vectori, si atunci vom pointa si valorile acestora).

Exemplu de utilizare a stivei:

Instructiune	Stiva	Observatii
	$sp:	initial, $sp pointeaza o zona goala de memorie
subu $sp, 4	$sp:()	se aloca un spatiu in stiva, acum se pointeaza o zona goala de memorie de dimensiunea unui word (4 bytes)
sw $t0, 0($sp)	$sp:($t0)	$sp pointeaza o zona de memorie care contine valoarea din registrul $t0
subu $sp, 4	$sp:()($t0)	$sp pointeaza o zona de memorie care contine 2 elemente pe 4 bytes, prima zona este goala, a doua il contine pe $t0
sw $t1, 0($sp)	$sp:($t1)($t0)	analog mai sus **important**: putem accesta $t1 ca fiind 0($sp) si pe $t2 ca fiind 4($sp)
subu $sp, 4	$sp:()($t1)($t0)	
sw $fp, 0($sp)	$sp:($fp v)($t1)($t0)	$fp v inseamna valoarea veche a lui $fp: $fp se va modifica fata de valoarea cu care a intrat in procedura, asa ca ii salvam valoarea veche pe stiva pentru a o putea restaura
addi $fp, $sp, 4	$sp:($fp v)$fp:($t1)($t0)	$fp pointeaza la o distanta de 4 bytes fata de $sp, adica pointeaza zona ($t1)($t0) **Acest pas este foarte important pentru ca, acum ca stiva a crescut, $t1 este raportat la 4($sp), iar $t0 este raportat la 8($sp). Cu cat se adauga elemente in stiva, $t1 si $t0 se vor raporta mereu la adrese diferite fata de $sp, dar se vor raporta MEREU constant fata de $fp: $t1 va fi mereu 0($fp), iar $t0 va fi mereu 4($fp). Stiva va tot creste pe masura ce avem nevoie de variabile locale.**
subu $sp, 4	$sp:()($fp v)$fp:($t1)($t0)	
sw $s0, 0($sp)	$sp:($s0 v)($fp v)$fp:($t1)($t0)	Am adaugat o variabila locala pe care o salvam pe stiva (ii salvam valoarea veche). Din nou, observam ca acum, $t1 este la 8($sp), iar $t0 la 12($sp), dar au ramas constante fata de $fp ($t1 la 0($fp) si $t0 la 4($fp)). In program, $s0 va fi raportat si el la $fp, si anume va fi -4($fp).
lw $s0, -8($fp)	$sp:($s0 v)($fp v)$fp:($t1)($t0)	Presupunem ca am terminat programul, si vrem sa restauram valorile. Restaurarea lor se face relativ la $fp, iar $s0 primeste valoarea lui veche, care era salvata in stiva la -8($fp)
addu $sp, 4	$sp:($fp v)$fp:($t1)($t0)	Daca subu $sp, 4 aloca spatiu in stiva (pentru push), cu addu $sp, 4 simulam un pop
lw $fp, -4($fp)	$sp:($fp v)$fp:($t1)($t0)	Avem grija sa restauram si $fp-ul! El trebuie restaurat ultimul, pentru ca el gestioneaza locul elementelor in cadrul de apel.
addu $sp, 4	$sp:($t1)($t0)	Stiva a reajuns la configuratia de baza, si putem reveni in cadrul locului de apel prin **jr $ra**.

Exemplu: suma a doua numere utilizand proceduri, cu return in $v0.

1. Ne definim programul ca pana acum, gandind ca avem de scris o procedura:

```
.data
        x: .word 10
        y: .word 15
.text
# aici va fi scrisa procedura
main:
        lw $t0, y               # in incarc pe y in $t0
        subu $sp, 4             # aloc spatiu pe stiva
        sw $t0, 0($sp)          # il salvez pe y in stiva

        lw $t0, x               # il incarc pe x in $t0
        subu $sp, 4             # aloc spatiu pe stiva
        sw $t0, 0($sp)          # il salvez pe x in stiva
                                # Observatie: am incarcat mai intai y, apoi x, pentru ca mereu vom
                                # incarca argumentele in ordine inversa. Daca vrem suma(x, y)
                                # atunci ordinea va fi sa il incarcam pe y, apoi pe x (pentru ca y va fi primul
                                # pus in stiva, deci va fi al doilea de la stanga la dreapta dupa incarcarea
                                # lui x
        jal suma                # jal suma apeleaza procedura
                                # (e un jump normal, doar ca retine in $ra unde trebuie sa revina
                                # la finalizarea procedurii)

        addu $sp, 8             # cand revenim din procedura, dezalocam spatiul din stiva
                                # addu $sp, 8 = de doua ori addu $sp, 4, adica facem pop
                                # celor doua elemente de tip word

        move $a0, $v0           # afisam pe ecran rezultatul, el este in $v0 retinut
        li $v0, 1
        syscall

        li $v0, 10
        syscall
```

Implementam procedura:

```
suma:
                                # in acest punct, stiva este $sp:(x)(y)
        subu $sp, 4             # $sp:()(x)(y)
        sw $fp, 0($sp)          # $sp:($fp v)(x)(y)
        addi $fp, $sp, 4        # $sp:($fp v)$fp:(x)(y)
        subu $sp, 4             # $sp:()($fp v)$fp:(x)(y)
        sw $s0, 0($sp)          # $sp:($s0)($fp v)$fp:(x)(y)
        subu $sp, 4             # $sp:()($s0)($fp v)$fp:(x)(y)
        sw $s1, 0($sp)          # $sp:($s1)($s0)($fp v)$fp:(x)(y)
        lw $s0, 0($fp)          # se incarca in $s0 valoarea lui x
        lw $s1, 4($fp)          # se incarca in $s1 valoarea lui y
        add $v0, $s0, $s1       # in $v0 se efectueaza suma
        sw $s1, -12($fp)        # restauram $s1 (care este la -12($fp))
        sw $s0, -8($fp)         # restauram $s0
        sw $fp, -4($fp)         # restauram $fp
        addu $sp, 12            # pentru ca am adaugat trei elemente, facem pop cu 3 elemente
                                # adica addu $sp, 3*4 care este addu $sp, 12
        jr $ra                  # revenim in cadrul programului principal
```

Programul final este:

```
.data
        x: .word 10
        y: .word 15
.text

suma:
        subu $sp, 4
        sw $fp, 0($sp)
        addi $fp, $sp, 4
        subu $sp, 4
        sw $s0, 0($sp)
        subu $sp, 4
        sw $s1, 0($sp)
        lw $s0, 0($fp)
        lw $s1, 4($fp)
        add $v0, $s0, $s1
        sw $s1, -12($fp)
        sw $s0, -8($fp)
        sw $fp, -4($fp)
        addu $sp, 12
        jr $ra

main:
        lw $t0, y
        subu $sp, 4
```

```
sw $t0, 0($sp)

lw $t0, x
subu $sp, 4
sw $t0, 0($sp)

jal suma

addu $sp, 8

move $a0, $v0
li $v0, 1
syscall

li $v0, 10
syscall
```

Exercitii:

1. Sa se implementeze functia newline.

   Solutie:

```
.data
        nl: .byte '\n'
.text

newline:
        lb $a0, nl      # incarcam in $a0 byte-ul nl
        li $v0, 11      # ii dam lui $v0 codul sistem pentru PRINT BYTE
        syscall         # apelam sistemul
        jr $ra          # revenim in punctul de dupa care s-a facut apelul

main:
        jal newline     # se apeleaza procedura newline

        li $v0, 10
        syscall
```

Observatie: aceasta a fost o procedura pentru care nu a fost necesara utilizarea stivei.

2. Sa se scrie o procedura care intoarce prin $v0 valorile 1 sau 0, dupa cum argumentul x, numar intreg incarcat in stiva, este par sau nu (functia even).
3. Sa se scrie o procedura care intoarce prin $v0 valorile -1, 0 sau 1, dupa cum numarul x, incarcat ca argument prin stiva, este negativ, 0 sau pozitiv (functia sgn).
4. Sa se implementeze functia calcul(x, y, z) care evalueaza expresia

$$calcul(x,y,z) := \sum_{i=0}^{x+1} \left[(y-i)\left(z + \left[\frac{i}{3}\right]\right) + 1 \right], x,y,z \in \mathbb{N}^*$$