LUCRUL CU STIVA. SUBRUTINE

1. Segmentul de stiva

🕮 Segmentul de stivă reprezintă o parte a memoriei principale:

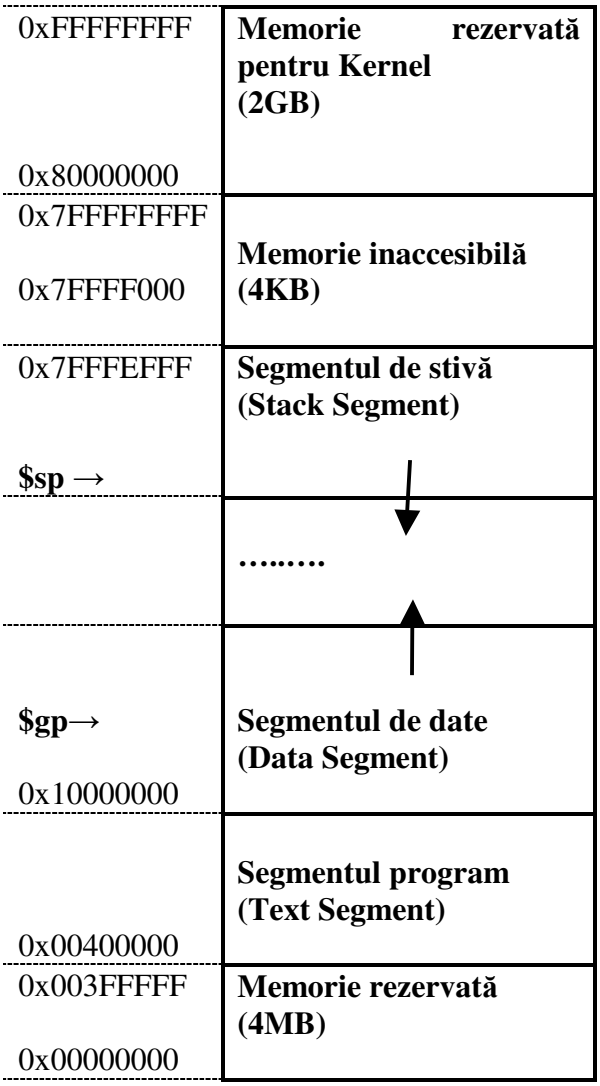


Fig.1 Organizarea memoriei principale

🛈 Mai multe informaţii despre organizarea memoriei principale găsiţi în laboratorul 3.

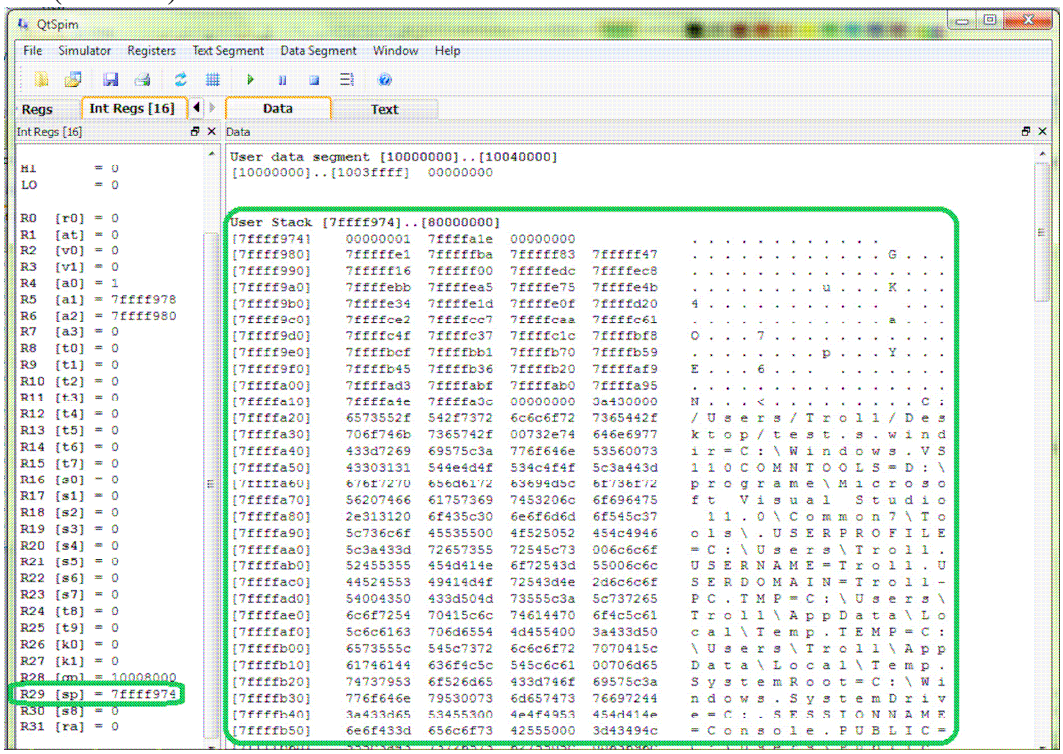
🕮 Segmentul de stivă (Stack Segment) nu are o dimensiune fixă. Aceasta începe de la adresa fixă 0x7FFEFFF şi creşte „în jos", în sensul că de fiecare dată când se

introduce o noua valoare în stivă, aceasta se păstrează în adresa imediat mai mică decât ultima adresă utilizată în stivă.

Vârful stivei este întotdeauna indicat de registul $sp. Acesta conţine de fiecare dată valoarea adresei celei mai mici din stivă.

Exerciţiu:

1. Deschideţi QtSpim.
2. Observaţi în al treilea panou al ferestrei principale segmentul de stivă (STACK).



3. Observaţi că valoarea registrului general **sp** este adresa vârfului stivei.

Întrebări:

1) Ce dimensiune are o valoare memorată în stivă?
2) Cât poate creşte stiva în jos?

2. PUSH şi POP

Stiva permite adăugarea şi extragerea elementelor de la un singur capăt, numit **vârful stivei**. Aceasta înseamnă că ultimul element introdus în stivă este primul care iese. De aceea, stiva este denumită şi **LIFO** (**L**ast **I**n **F**irst **O**ut).

Spre exemplu, în stiva din figura de mai jos se introduc valorile 10 şi 17, apoi se extrage un element. Se observă clar că ultimul element introdus este primul extras.

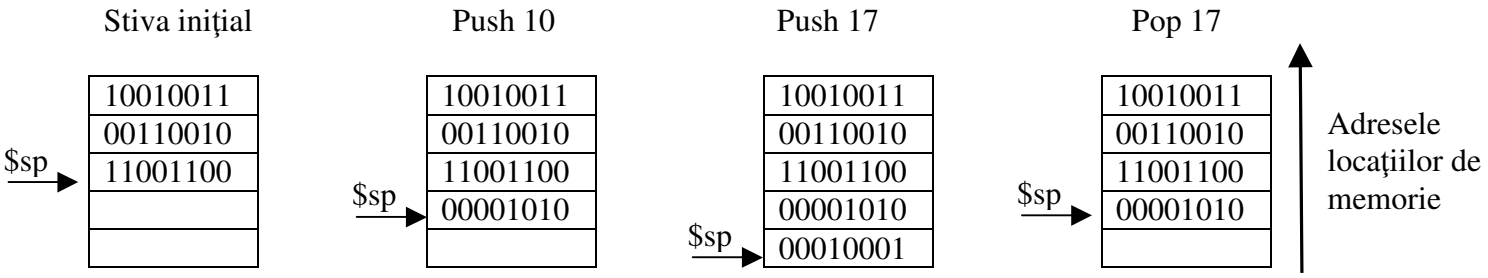


Fig.2 Creşterea stivei în jos

📖 Cele 2 operaţii specifice stivei sunt:

- PUSH – introducere în stivă;
- POP – eliminare din stivă.

Acestea au fost exemplificate în exemplul anterior.

📖 Considerând că se foloseşte registrul $t0 pentru păstrarea valorii extrasă din stivă/ care se doreşte a fi introdusă în stivă, codul MIPS echivalent pentru cele 2 instrucţiuni este:

```
#PUSH
subu $sp, $sp, 4      #se scade varful stivei pentru a indica locatia de memorie
                      #anterioara
sw $t0, 0($sp)        #store la adresa indicata de varful stivei a unui valori stocare
                      #intr-un registru

#POP
lw $t0, 0($sp)        #load in registru a valorii de la adresa din varful stivei
addu $sp, $sp, 4      #se incrementeaza varful stivei pentru a indica urmatoarea locatie
                      #de memorie
```

? Întrebări:

1) De ce se incrementează/decrementează valoarea din registrul $sp cu valoarea 4 şi nu cu o altă valoare?

Probleme propuse:

1. Pentru un sir de caractere cunoscut, să se determine şirul de caractere obţinut prin oglindirea şirului iniţial. De exemplu: pentru şirul „sir” se va obţine „ris”. Utilizaţi operaţiile PUSH şi POP.
2. Verificaţi dacă un număr dat este palindrom. Utilizaţi stiva.

3. Subrutine

Subrutinele sunt utilizate pentru secvenţe de cod folosite de mai multe ori în cadrul programului, pentru a evita duplicarea codului.

Lucrul cu subrutine presupune:

- apelarea subrutinei;
- transmiterea parametrilor necesari;
- revenirea din subrutină şi eventual întoarcerea rezultatelor.

În MIPS o subrutină înseamnă de fapt o secvenţă de cod indicată de o etichetă. În general, subrutinele se plasează după main: (însă în segmentul de program .text), pentru ca să nu se execute fără a fi apelate:

```
.text
        subrutina:
                #cod subrutina
        main:
                #cod main
```

Apelul unei subrutine se face printr-un salt la o etichetă după care urmează corpul subrutinei. Este însă necesar ca înainte de apelarea subrutinei, să se păstreze adresa următoarei instrucţiuni care trebuie executată la revenirea din subrutină, pentru a putea putea continua programul. Această valoare se păstrează în registrul $ra. Registrul va conţine adresa următoarei instrucţiuni după apelul de subrutină.

Instrucţiunile de salt folosite pentru apelul de subrutine înglobează cele 2 necesităţi: salvează adresa instrucţiunii următoare apelului şi realizează apelul (saltul către eticheta care indică subrutina). Aceste instrucţiuni sunt: bltzal, bgezal, jal, jalr.

Instrucţiune	Operaţia efectuată	Format	Exemplu
bltzal rs, eticehta	**B**ranch on **L**ess **T**han **Z**ero and **L**ink Dacă rs < 0, atunci salt la etichetă ca apel_procedură: în registrul $ra ($31) păstrează adresa de întoarcere, adică adresa următoarei instrucţiuni	I	*bltzal rs, eticheta*
bgezal rs, eticheta	**B**ranch on **G**reater Than or **E**qual to **Z**ero **a**nd **L**ink Dacă rs ≥ 0, atunci salt la etichetă ca apel_procedură: în registrul $ra ($31) păstrează adresa de întoarcere, adică adresa următoarei instrucţiuni	I	*bgezal $t0, et*
jal eticheta	**J**ump **a**nd **L**ink Salt la etichetă ca apel_procedură: în registrul $ra ($31) păstrează adresa de întoarcere, adică adresa următoarei instrucţiuni. Nu este PC-relativă, adică în formatul instrucţiunii nu se păstrează deplasamentul faţă de instrucţiunea curentă (ca în cazul instrucţiunilor branch), ci adresa dată de etichetă.	J	*jal et*
jalr <rd,> rs	Jump and Link Register Salt la etichetă cu păstrarea adresei următoarei instrucţiuni în registrul $rd. Dacă lipseşte registrul rd, atunci acesta este considerat implicit registrul ra. Nu este PC-relativă, adică în formatul instrucţiunii nu se păstrează deplasamentul faţă de instrucţiunea curentă (ca în cazul instrucţiunilor branch), ci adresa dată de etichetă.	R	*jalr $t1, $t2*

Tabelul 1. Instrucţiunile de apel

📖 **<u>Revenirea dintr-o subrutină</u>** înseamnă salt la adresa următoare instrucţiunii care a apelat procedură, adică la adresa stocată în $ra. Instrucţiunea utilizată pentru revenire este jr $ra:

Instrucţiune	Operaţia efectuată	Format	Exemplu
jr rs	Jump Register Salt la adresa stocată în registrul rs	R	*jr $t1*

Tabelul 2..Instrucţiunea jr

Observaţi că instrucţiunea are o aplicabilitate mai largă, putând face salt la o adresă stocată într-un registru oarecare, nu neapărat $ra.

📖 **Transmiterea de parametrii şi returnarea rezultatelor** se realizează prin intermediul regiştrilor.

Nume	Mnemonic	Utilizare convenţională
$0	$zero	**Zero** register – conţine întotdeauna valoarea 0.
$1	$at	**A**ssembler **T**emporary – este rezervat pentru asamblor.
$2…$3	v0…v1	**V**alue registers - folosiţi pentru reţinerea rezultatelor întregi ale evaluărilor unor expresii sau funcţii.
$4…$7	a0…a3	**A**rguments – folosiţi pentru transmiterea argumentelor unor subrutine (parametrii actuali). Valorile acestora nu se păstrează dupa apelul de procedură.
$8…$15	t0…t7	**T**emporary registers – folosiţi pentru evaluarea expresiilor. Valorile acestora nu se păstrează la apeluri de proceduri.
$16…$23	s0…s7	**S**aved registers - valorile acestora sunt păstrate la apeluri de proceduri.
$24…$25	t8…t9	**T**emporary registers – folosiţi pentru evaluarea expresiilor. Valorile acestora nu se păstrează la apeluri de proceduri.
$26…$27	k0…k1	**K**ernel registers - rezervaţi pentru sistemul de operare.
$28	gp	**G**lobal **P**ointer – indică spre mijlocul unui bloc de memorie care păstrează constante şi variabile globale.
$29	sp	**S**tack **P**ointer – indică ultima locaţie utilizată în stivă.
$30	fp	**F**rame **P**ointer - pointer spre cadrul curent în stivă.
$31	ra	**R**eturn **A**ddress – conţine adresa de întoarcere, fiind folosit pentru evaluarea expresiilor.

Tabelul 3. Regiştrii generali

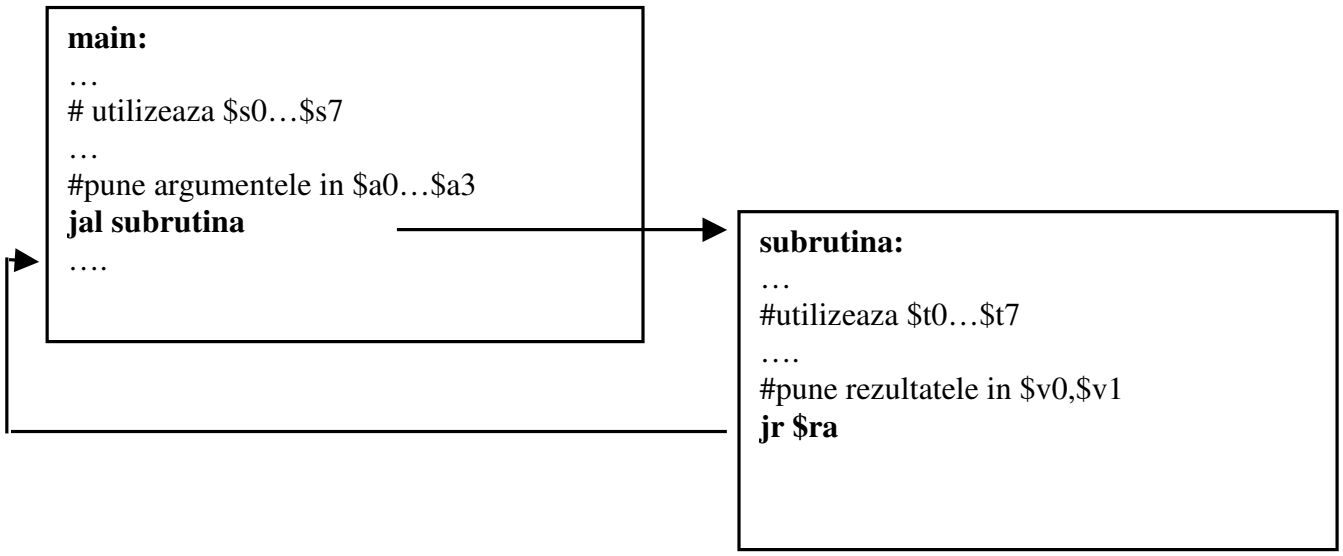
Observaţi folosirea regiştrilor:

- a0…a3 pentru transmiterea argumentelor;
- v0, v1 pentru întoarcerea valorilor;
- s0…s7 pentru utilizarea în main;
- t0…t7 pentru utilizarea în subrutină.

ⓘ Mai multe informaţii despre regiştrii găsiţi în laboratorul 2.

În cazul în care numărul de regiştrii este insuficient sau se doreşte restaurarea valorilor (în cazul subrutinelor, la valoarea acestora de la intrarea în subrutină; în cazul rutinei apelante la valorile de înainte de apel) se utilizează stiva. Aceasta permite stocarea unor valori temporare. Segmentul de date poate fi de asemenea folosit pentru transmiterea informaţiilor spre şi dinspre subrutine.

🕮 O reprezentare schematică a unui apel de subrutină, cu tot ce implică aceasta, este expusă mai jos:



🖫 **Problemă rezolvată:**

Fiind date 3 valori identificate prin etichetele x, y şi z, să se determine valoarea expresiei:

$$(x-1)^2 + (y-1)^2 + (z-1)^2$$

folosind o subrutină care pentru un număr oarecare t calculează $(t-1)^2$.

```
.data
x: .word 10
y: .word 11
z: .word 12

.text

proc:                   #subrutina
move $t0,$a0            #preluarea parametrului
```

```
li $t1,1                #calcule
sub $t0,$t0,$t1
mulo $t0,$t0,$t0
move $v0,$t0            #pastrarea valorii returnate in registrul $v0
jr $ra                  #revenire din subrutina

main:                   #programul apelant
li $s0,0

lw $a0,x        #incarcarea parametrului pentru subrutina proc
jal proc        #apelul de subrutina
add $s0, $s0,$v0 #continuarea codului la revenirea din subrutina

lw $a0,y        #apelarea subrutinei cu y ca parametru
jal proc
add $s0, $s0,$v0

lw $a0,z        #apelarea subrutinei cu z ca parametru
jal proc
add $s0, $s0,$v0

li $v0,10       #oprire executie program
syscall
```

Probleme propuse:

1. Fie şirul următor definit recursiv:
   $a_1 = 1$;
   $a_2 = 2$;
   $a_n = 3*a_{n-2} + a_{n-1}$.
   Determinaţi al 5-lea element al şirului, folosind o subrutină care primeşte 2 termeni consecutivi şi întoarce următorul element al şirului.
2. Determinaţi lungimea maximă a unei secvenţe de caractere repetitive dintr-un şir de caractere. De exemplu pentru şirul „aaabbcdddee” se va întoarce 3, fiindcă „aaa” şi „ddd” au lungimea maximă, egală cu 3.
3. Se citesc de la consolă un număr natural şi o cifră. Să se afişeze „Da” dacă printre cifrele numărului citit se găseşte această cifră şi „Nu” în caz contrar.
4. Să se determine dacă o matrice are numai elemente impare pe diagonala principală.
5. Ordonaţi crescător un vector de numere întregi.

ⓘ Mai multe informaţii

MIPS32™ Architecture For Programmers -Volume II: The MIPS32™ Instruction Set
http://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS_Vol2.pdf

MIPS Instruction Coding
http://www.cs.sunysb.edu/~cse320/MIPS_Instruction_Coding_With_Hex.pdf

Programmed Introduction to MIPS Assembly Language
http://chortle.ccsu.edu/AssemblyTutorial/index.html