### LUCRUL CU STIVA. SUBRUTINE

### 1. Segmentul de stiva

Segmentul de stivă reprezintă o parte a memoriei principale:

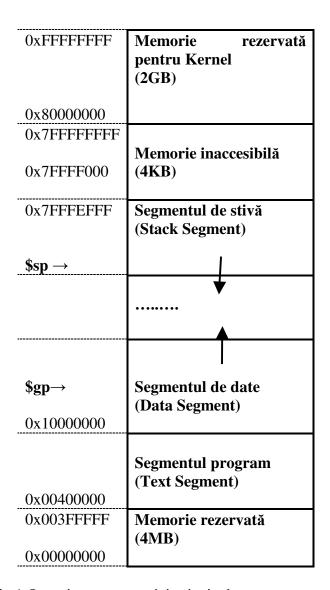


Fig.1 Organizarea memoriei principale

• Mai multe informații despre organizarea memoriei principale găsiți în laboratorul 3.

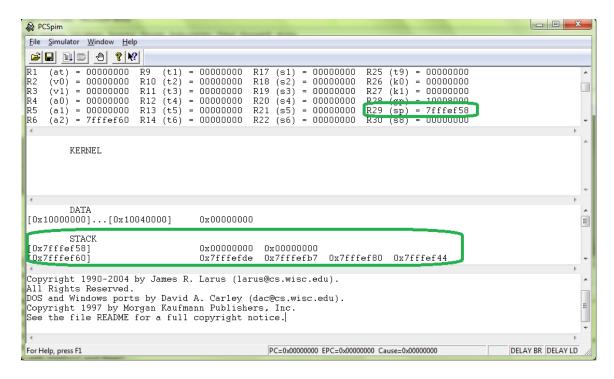
Segmentul de stivă (Stack Segment) nu are o dimensiune fixă. Aceasta începe de la adresa fixă 0x7FFEFFF și crește "în jos", în sensul că de fiecare dată când se

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 conține de fiecare dată valoarea adresei celei mai mici din stivă.

#### **■** Exercițiu:

- 1. Deschideţi PCSpim.
- 2. Observați în al treilea panou al ferestrei principale segmentul de stivă (STACK).



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

### ? Întrebări:

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

## 2. PUSH și POP

Stiva permite adăugarea și 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 și **LIFO** (Last In First Out).

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

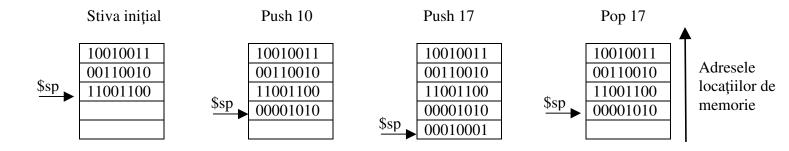


Fig.2 Creșterea stivei în jos

- Cele 2 operații specifice stivei sunt:
  - PUSH introducere în stivă;
  - POP eliminare din stivă.

Acestea au fost exemplificate în exemplul anterior.

Considerând că se folosește registrul \$t0 pentru păstrarea valorii extrasă din stivă/ care se dorește a fi introdusă în stivă, codul MIPS echivalent pentru cele 2 instrucțiuni este:

#PUSH	
subu \$sp, \$sp, 4	#se scade varful stivei pentru a indica locatia de memorie #anterioara
sw \$t0, O(\$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 și nu cu o altă valoare?

### **☐** Probleme propuse:

- 1. Pentru un sir de caractere cunoscut, să se determine șirul de caractere obținut prin oglindirea șirului inițial. De exemplu: pentru șirul "sir" se va obține "ris". Utilizați operațiile PUSH și POP.
- 2. Verificați dacă un număr dat este palindrom. Utilizați stiva.

#### 3. Subrutine

Subrutinele sunt utilizate pentru secvențe 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 și eventual întoarcerea rezultatelor.

În MIPS o subrutină înseamnă de fapt o secvență 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 instrucțiuni care trebuie executată la revenirea din subrutină, pentru a putea putea continua programul. Această valoare se păstrează în registrul \$ra. Registrul va conține adresa următoarei instrucțiuni după apelul de subrutină.

Instrucțiunile de salt folosite pentru apelul de subrutine înglobează cele 2 necesități: salvează adresa instrucțiunii următoare apelului și realizează apelul (saltul către eticheta care indică subrutina). Aceste instrucțiuni sunt: bltzal, bgezal, jal, jalr.

Instrucțiune	Operația efectuată	Format	Exemplu
bltzal rs, eticehta	Branch on Less Than Zero and Link	Ι	bltzal rs, eticheta
	Dacă rs < 0, atunci salt la etichetă ca apel_procedură: în registrul \$ra (\$31) păstrează adresa de întoarcere, adică adresa următoarei instrucțiuni		
bgezal rs, eticheta	Branch on Greater Than or Equal to Zero and Link	I bgezal \$t0, et	
	Dacă rs ≥ 0, atunci salt la etichetă ca apel_procedură: în registrul \$ra (\$31) păstrează adresa de întoarcere, adică adresa următoarei instrucțiuni		
jal eticheta	Jump and Link	J	jal et
	Salt la etichetă ca apel_procedură: în registrul \$ra (\$31) păstrează adresa de întoarcere, adică adresa următoarei instrucțiuni.  Nu este PC-relativă, adică în formatul instrucțiunii nu se păstrează deplasamentul față de instrucțiunea curentă (ca în cazul instrucțiunilor branch), ci adresa dată de etichetă.		
jalr <rd,> rs</rd,>	Jump and Link Register  Salt la etichetă cu păstrarea adresei următoarei instrucțiuni în registrul \$rd. Dacă lipsește registrul rd, atunci acesta este considerat implicit registrul ra.	R	jalr \$t1, \$t2
	Nu este PC-relativă, adică în formatul instrucțiunii nu se păstrează deplasamentul față de instrucțiunea curentă (ca în cazul instrucțiunilor branch), ci adresa dată de etichetă.		

Tabelul 1. Instrucțiunile de apel

Revenirea dintr-o subrutină înseamnă salt la adresa următoare instrucțiunii care a apelat procedură, adică la adresa stocată în \$ra. Instrucțiunea utilizată pentru revenire este jr \$ra:

Instrucțiune	Operația efectuată Format Exemplu		Exemplu
jr rs	Jump Register	R	jr \$t1
	Salt la adresa stocată în registrul rs		

Tabelul 2..Instrucțiunea jr

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

Transmiterea de parametrii și returnarea rezultatelor se realizează prin intermediul regiștrilor.

Nume	Mnemonic	Utilizare convențională	
\$0	\$zero	<b>Zero</b> register – conține întotdeauna valoarea 0.	
\$1	\$at	Assembler Temporary – este rezervat pentru asamblor.	
\$2\$3	v0v1	v0v1 Value registers - folosiți pentru reținerea rezultatelor între ale evaluărilor unor expresii sau funcții.	
\$4\$7	a0a3	Arguments – folosiți pentru transmiterea argumentelor unor subrutine (parametrii actuali). Valorile acestora nu se păstrează dupa apelul de procedură.	
\$8\$15	t0t7 Temporary registers – folosiți pentru evaluarea expresiilo Valorile acestora nu se păstrează la apeluri de proceduri.		
\$16\$23	s0s7 Saved registers - valorile acestora sunt păstrate la apelu proceduri.		
\$24\$25	t8t9	Temporary registers – folosiți pentru evaluarea expresiilor. Valorile acestora nu se păstrează la apeluri de proceduri.	
\$26\$27	k0k1	Kernel registers - rezervați pentru sistemul de operare.	
\$28	gp	Global Pointer – indică spre mijlocul unui bloc de memorie care păstrează constante și variabile globale.	
\$29	sp	Stack Pointer – indică ultima locație utilizată în stivă.	
\$30	fp	Frame Pointer - pointer spre cadrul curent în stivă.	
\$31	ra	Return Address – conține adresa de întoarcere, fiind folosit pentru evaluarea expresiilor.	

Tabelul 3. Regiştrii generali

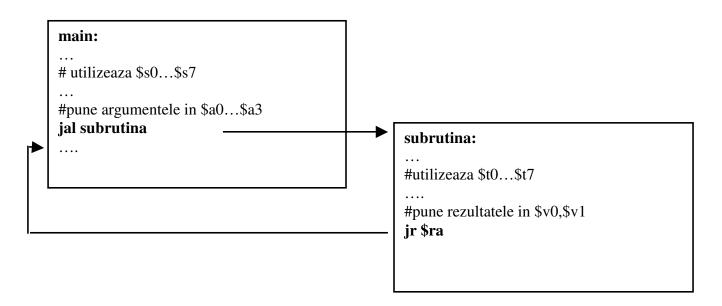
Observați folosirea regiștrilor:

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

① Mai multe informații despre regiștrii găsiți în laboratorul 2.

În cazul în care numărul de regiștrii este insuficient sau se dorește 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 informațiilor spre și 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)<sup>2</sup>.

.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 ir \$ra #revenire din subrutina #programul apelant main: li \$s0,0 *lw* \$*a*0,*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:

 $a_1 = 1;$   $a_2 = 2;$  $a_n = 3*a_{n-2} + a_{n-1}.$ 

Determinați al 5-lea element al șirului, folosind o subrutină care primește 2 termeni consecutivi și întoarce următorul element al șirului.

- 2. Determinați lungimea maximă a unei secvențe de caractere repetitive dintr-un șir de caractere. De exemplu pentru șirul "aaabbcdddee" se va întoarce 3, fiindcă "aaa" și "ddd" au lungimea maximă, egală cu 3.
- 3. Se citesc de la consolă un număr natural și o cifră. Să se afișeze "Da" dacă printre cifrele numărului citit se găsește această cifră și "Nu" în caz contrar.
- 4. Să se determine dacă o matrice are numai elemente impare pe diagonala principală.
- 5. Ordonați crescător un vector de numere întregi.

## ① Mai multe informații

MIPS32<sup>TM</sup> Architecture For Programmers -Volume II: The MIPS32<sup>TM</sup> Instruction Set <a href="http://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS\_Vol2.pdf">http://www.cs.cornell.edu/courses/cs3410/2008fa/MIPS\_Vol2.pdf</a>

MIPS Instruction Coding

http://www.cs.sunysb.edu/~cse320/MIPS\_Instruction\_Coding\_With\_Hex.pdf

Programmed Introduction to MIPS Assembly Language <a href="http://chortle.ccsu.edu/AssemblyTutorial/index.html">http://chortle.ccsu.edu/AssemblyTutorial/index.html</a>