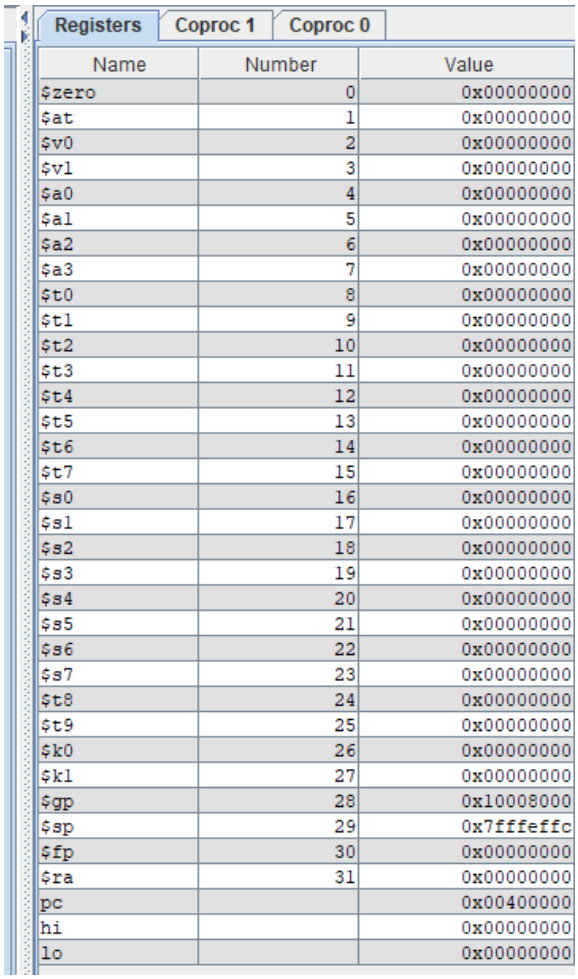
STIVA-Introducere

In MIPS exista o mare bucata de memorie rezervata unui singur registru. Acesta parte din memorie poate fi folosita ca un vector, utilizatorul poate naviga prin ea cu un indice in ce parte doreste. Totusi, conform conventiei, nu putem folosi acest spatiu ca un vector ci ca o stiva. In continuare se va prezenta cum putem accesa stiva, cum putem rezerva loc pe stiva, incarca variabile in aceasta, citi din ea si cum putem elibera loc rezervat .

STIVA-Alocare

In poza de mai jos putem observa toti registri principali dintr-un procesor .



Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

Stiva, sau mai precis, adresa ultimei valori din stiva, se afla in registrul $sp.

Se observa ca adresa din $sp este 7fffeffc, ceea ce insemna ca stiva are inaltimea de 2 147 483 647 octeti (deoarece 2 147 483 647 in hexazecimal este 7fffeffc).

La inceputul programului valoarea din $sp reprezinta inaltimea maxima, deci nu voi putea sa ma duc la adresa $sp+1 deoarece aceasta nu exista, cum nu voi putea sa incarc nici in adresa actuala a lui $sp. Pentru a putea incarca variabile pe stiva, mai intai, va trebui sa scad din $sp numarul de octeti de care am nevoie. Prin actiunea de scadere al lui $sp se muta pointerul spre fundul stivei si astfel ajung la o adrasa la care pot incarca variabile, numesc aceasta scadere “alocare de spatiu pe stiva”. De mentionat este ca nu conteaza valorile ce se afla pe spatiu alocat, prin conventie, acestea nu mai sunt folositoare si se considera drept valori reziduale.

Programul de mai jos aloca 8 octeti pe stiva :

```
1   .data
2
3   .text
4   main:
5
6   #aloc 8 octeti pe stiva
7   subi $sp,$sp,8
8
9   #exit
10  li $v0, 10
11  syscall
```

Dupa rularea programului $sp a devenit 7fffeff4, 8 octeti au fost eliberati si sunt gata pentru a fi utilizati.

STIVA- Incarcare de valori

Pentru a incarca valori in stiva sintaxa este identica cu cea de incarcare intr-un, folosind adresa lui $sp drept adresa initiala(v[0]). Desi s-ar putea utiliza valori negative precum v[-1], prin conventie, acestea nu se accepta,se pot folosi numai valorile din stiva alocate adica pot incarca de la adrese mai mari sau egale cu $sp.

In programul de mai jos se exemplifica incarcarea unui word(numar intreg pe 4 octeti) citit din memorie in stiva

```
1   .data
2   n: .word 1
3   .text
4   main:
5   #citesc valoarea din memorie
6   lw $t0,n
7
8   #aloc 4 octeti pe stiva
9   subi $sp,$sp,4
10
11  #incarc n pe stiva
12  sw $t0,0($sp) # sp[0]=n
13
14  #exit
15  li $v0, 10
16  syscall
```

OBSERVATIE : Pointerul $sp se schimba cand aloc spatiu pe stiva, de aceea offsetul trebuie si el ajustat mereu cand se aloca spatiu pe stiva

Exemplu:

```
.data
n: .word 1
m: .word 2
.text
main:
#citesc valorile din memorie
lw $t0,n
lw $t1,m

#aloc 4 octeti pe stiva
subi $sp,$sp,4

#incarc n pe stiva
sw $t0,0($sp) # sp[0]=n

#aloc alti 4 octeti pe stiva
subi $sp,$sp,4
#ATENTIE, dupa aceasta comanda numarul n se gaseste la adresa sp[1]

#incarc m pe stiva
sw $t1,0($sp) # sp[0]=m , sp[1]=n
#chiar daca n a fost incarcat la adresa sp[0] cand am mai alocat 4 octeti
#aceasta valoare a devenit sp[1]
```

Totusi , pentru a evita acesta problema programul se poate rescrie astfel incat tot spatiul necesar este alocat la inceput:

```
.data
n: .word 1
m: .word 2
.text
main:
#citesc valorile din memorie
lw $t0,n
lw $t1,m

#aloc pentru ambele variabile spatiu
subi $sp,$sp,8

#incarc n si m pe stiva
sw $t0,0($sp) # sp[0]=n
sw $t1,4($sp) # sp[1]=m
#la final voi avea sp[0]=n, sp[1]=m
```

STIVA-Citire din sitva

Citire din stiva este indentica cu citirea din vector. De asemenea trebuie sa tinem cont din nou de offset, daca mai alocam inca un numar n de octeti pe stiva, pentru a ajunge la valoarea ce inainte era tinuta minte la adreasa sp[k], va trebui sa mergem la adresa sp[k+n].

Exemplu citire 1 :

```
1  .data
2  n: .word 1
3  m: .word 2
4  .text
5  main:
6  #citesc valorile din memorie
7  lw $t0,n
8  lw $t1,m
9
10 subi $sp,$sp,4
11 sw $t0,0($sp)
12
13 #citesc n de pe stiva
14 lw $t1,0($sp)
```

Exemplu citire 2:

```
1  .data
2  n: .word 1
3  m: .word 2
4  .text
5  main:
6  lw $t0,n
7  lw $t1,m
8
9  subi $sp,$sp,8
10 sw $t0,0($sp)
11 sw $t1,4($sp)
12
13 #citesc n si m de pe stiva
14 lw $t3,0($sp) #t3=sp[0]
15 lw $t3,4($sp) #t4=sp[1]
16
```

STIVA-Eliberare de memorie

Asemanator alocarii de memorie, eliberarea de memorie reprezinta marirea pointerului $sp pana la adresa sa originala.

Daca o adresa de memorie unde am stocat un numar se elibereaza, conform conventiei, numarul acela se transfoma intr-o valoare reziduala.

Alocarea si dezalocarea pot avea loc oriude si de oricate ori intr-un program dar,conform conventiei, orice spatiu alocat intr-o functie va fi eliberat tot in aceasta (daca am alocat 4 octeti in main trebuie sa eliberez 4 octeti tot in main), totusi aceasta conventie se mai poate incalca pentru usurarea programului.

Exemplul:

```
1  .data
2  .text
3  main:
4
5  subi $sp,$sp,8# aloc loc pe stiva
6
7  #... lucrez cu cei 8 octeti alocati pe stiva ...
8
9  addi $sp,$sp,8 # eliberez stiva
10
11 #exit
12 li $v0, 10
13 syscall
```

STIVA-Functii cu transmitere de parametri prin stiva

In continuare vom explica rolul registrului $fp,salvarea si restaurarea variabilelor pe stiva ,folosirea stivei in cazul transmiterii parametrilor catre o functie si a intoarceri rezultatelor prin stiva .

STIVA – Salvarea si restaurarea elementelor pe stiva

In programare stiva este folosita la apeluri de functii pentru a conserva valoarile ce se transmit in antetul acesteia

Exemplu: fie programul urmator in C

```
void func(int a,int b)
{
	a=a+5;
	b=b+5;
}

int main()
{
	int a=2,b=3;
	func(a,b);
	return 0;
}
```

Pentru programul prezentat functia func are trivialul scop de a marii cu 5 cele 2 numere pe care le primeste. Totusi, dupa apelul functiei func, la intoarcerea in main, valorile lui a si b vor ramane cele de dinaintea apelarii (2 si 3), valorile din functie (7 si 8) fiind uitate. Pentru a putea simula un astfel de fenomen in MIPS vom salva valorile lui a si b in stiva, vom apela functia si, la intoarcerea in main, vom incarca(restaura) din stiva in registrul lui a si registrul lui b valorile lor initiale , de dinainte de apel, astfel:

```
1  .data
2  .text
3  main:
4  #initilizez cei 2 registri cu 2 si 3
5  li $t0,2
6  li $t1,3
7
8  subi $sp,$sp,8 # aloc spatiu pe stiva
9  sw $t0,0($sp) # salvez valoarea lui $t0 pe stiva
10 sw $t1,4($sp) #bsalvez valoarea lui $t1 pe stiva
11
12 jal func  #apelez functia func
13
14 lw $t0,0($sp) # restaurez valoarea lui $t0 din stiva
15 lw $t1,4($sp) # restaurez valoarea lui $t1 din stiva
16 addi $sp,$sp,8 # eliberez stiva
17
18 #exit
19 li $v0, 10
20 syscall
21
22 func:
23 addi $t0,$t0,5 # alterez valoarea lui $t0
24 addi $t1,$t0,5# alterz valoarea lui $t1
25 jr $ra # ma intorc in main
```

STIVA-Intoarcerea rezulatatelor prin stiva

Stiva poate fi folosita si pentru a intoarce rezultate din functie .

Fie urmatorul program in C:

```
int suma(int a,int b)
{

	int c=a+b;
	return c;

}

int main()
{

	int a=2,b=3;
	a=suma(a,b);

	return 0;

}
```

In programul de mai sus functia suma intoarce o valoarea in variabila “a”. In MIPS, pentru a intoarce o valoare se va folosi tot stiva doar ca de data aceasta vom altera adresa din $sp unde este salvata valoarea lui “a” astfel incat la restaurare vom avea valoarea returnata din suma in “a” astfel:

```
1  .data
2  .text
3  main:
4  #initilizez cei 2 registri cu 2 si 3
5  li $t0,2
6  li $t1,3
7
8  subi $sp,$sp,8 # aloc spatiu pe stiva
9  sw $t0,0($sp) # salvez valoarea lui $t0 pe stiva
10 sw $t1,4($sp) #bsalvez valoarea lui $t1 pe stiva
11
12 jal func  #apelez functia func
13
14 lw $t0,0($sp) # returnez din functie noua valoare a lui $t0
15 lw $t1,4($sp) # restaurez valoarea lui $t1 din stiva
16 addi $sp,$sp,8 # eliberez stiva
17
18 #exit
19 li $v0, 10
20 syscall
21
22 func:
23 add  $t2,$t0,$t1 # adun cele 2 numere intr-un registru oarecare
24 sw $t2,0($sp) # salvez valoarea ce trebuie returnata in locul unde era valoarea lui "t0"
25 jr $ra # ma intorc in main
```

Totusi, acesta este un caz particular, in cazul general functia main() ar arata astfel:

```
int main()
{

        int a=2,b=3,c;
        c=suma(a,b);
        return 0;

}
```

In cazul acesta vom aloca o noua pozitie pe stiva in care vom salva valoarea returnata astfel:

```
.data
.text
main:
#initilizez cei 2 registri cu 2 si 3
li $t0,2
li $t1,3

subi $sp,$sp,12 # aloc spatiu pe stiva pentru 3 word-uri
sw $t0,0($sp) # salvez valoarea lui $t0 pe stiva
sw $t1,4($sp) #bsalvez valoarea lui $t1 pe stiva

jal func  #apelez functia func

lw $t0,0($sp) # restaurez valoarea lui $t0 din stiva
lw $t1,4($sp) # restaurez valoarea lui $t1 din stiva
lw $s0,8($sp) # returnez valoarea functiei "func"
addi $sp,$sp,12 # eliberez stiva

li $v0, 10
syscall

func:
add $t2,$t0,$t1 # adun cele 2 numere intr-un registru oarecare
sw $t2,8($sp) # salvez valoarea ce trebuie returnata in locul unde era valoarea lui "t0"
jr $ra # ma intorc in main
```

STIVA- registrul $fp

Presupunem urmatorul scenariu: o functie ce poate apela in interiorul ei un numar de functii necunoscut. Pentru un astfel de program folosirea $sp-ului este inutila deoarece nu se stie cati octeti s-au alocat, deci nu putem determina offset-ul. Pentru rezolvarea unei astfel de probleme exista registrul $fp ce , prin conventie, va tine minte adresa $sp-ului de la apelul functie, astfel valorile din functie se vor gasi fie imediat deasupra valorii fp-ului(fp[0],fp[1],fp[2]…), fie imediat dedesupt (fp[-1], fp[-2], fp[-3]…) alegerea fiind la indemana programatorului.

OBSERVATIE: Conform conventiei $fp-ul este obligatoriu de salvat inainte de apeluri, restaurant la terminarea apelului si folosit in interiorul functiei in locul $sp-ului.

Fie urmatoarea functie in C:

```
int incrementare(int n)
{

	return n++;

}

int main()
{

	int n=1,m;
	m=incrementare(n);
	return 0;

}
```

Echivalentul sau in MIPS, respectand toate conventiile este:

```
.data
.text
main:
li $t0,1 # initializez $t0

subi $sp,$sp,12 # aloc spatiu pe stiva pentru t0,fp, si valoarea returnata de functie
sw $t0,0($sp) # salvez valoarea lui $t0 pe stiva
sw $fp,4($sp) # salvez valoarea lui $fp pe stiva
move $fp,$sp # ii dau lui $fp valoarea lui $sp pentru a lucra cu el in functie

jal increment  #apelez functia func

lw $t0,0($sp) # restaurez valoarea lui $t0 din stiva
lw $fp,4($sp) # restaurez valoarea lui $t1 din stiva
lw $s0,8($sp) # returnez valoarea functiei "func"
addi $sp,$sp,12 # eliberez stiva

li $v0, 10
syscall

increment:
addi $t1,$t0,1 # t1=t0+1
sw $t1,8($fp) # fp[2]=t1
jr $ra # ma intorc in main
```