Laborator 4

Lucrarea prezintă modul în care subprogramele sunt tratate folosind arhitectura MIPS.

1. Subprogramele în programarea pe calculator

Unele dintre cele mai importante concepte folosite în programare o reprezintă subprogramele. Acestea reprezintă blocuri de cod ce pot fi apelate de oriunde din interiorul programelor. Folosirea subprogramelor vine cu mai multe avantaje:

- **Abstractizarea și încapsularea** : un subprogram poate fi văzut ca un "black box", programatorul cunoaște intrările, nu este neapărat interesat de implementare, și folosește rezultatul
- **Organizarea** : codul poate fi împărțit în mai multe subprograme la care să lucreze mai multe persoane
- **Refolosirea** : subprogramele pot fi folosite și de alte persoane în afara autorului, în proiecte în viitor, etc: se evită dublarea muncii
- **Depanarea** devine mai ușoară

2. Subprogramele în arhitectura MIPS

Majoritatea arhitecturilor moderne de procesoare oferă suport pentru instrucțiuni de salt și revenire dintr-un subprogram. Limbajul de asamblare MIPS pune la dispoziție 2 instrucțiuni pentru apelul subprogramelor:

- Instrucțiunea **jal** (jump and link), care execută un salt la adresa furnizată și salvează adresa următoarei instrucțiuni în registrul `$ra` (return address), în felul acesta legând subprogramul de apel. Prin instrucțiunea următoare se înțelege următoarea instrucțiune din cod după jal, și nu instrucțiunea ce este executată după jal. Practic se dorește continuarea programului apelant din locul de unde s-a produs saltul.
- Instrucțiunea **jr** (jump register) execută un salt la adresa registrului operand. Această instrucțiune nu are legătură exclusiv cu apelul unui subprogram, dar folosind registrul `$ra` drept operand, se poate reveni în locul de unde s-a efectuat saltul, condiție necesară pentru orice porțiune de cod sa fie considerată subprogram:

```
jr      $ra
```

Mai jos poate fi observat cum arată apelul unui subprogram:

```
        .text
main:
    jal     subprog
    ...
    # End of main
    jr      $ra

subprog:
    ...
    # End of subprogram
```

```
jr      $ra
```

Programul apelant furnizează valorile de start sub forma unor argumente, iar subprogramul returnează un rezultat.

Pentru toate acestea, MIPS utilizează regiștrii:

- `$a0,...,$a3` pentru preluarea argumentelor; dacă mai mult de 4 argumente sunt necesare, este programator va decide cum vor fi aceștia furnizați
- `$v0` pentru valoarea returnată până în 32 de biți; dacă valoarea returnată are nevoie de 64 de biți se poate folosi și registrul `$v1` pentru a stoca partea superioară

3. Controlul resurselor

Apelul unui subprogram trebuie înțeles ca o colaborare între programul apelant și subprogram.

Execuția subprogramului presupune și folosirea altor regiștrii pentru calculele intermediare rezultatului. Acest lucru poate duce la alterarea valorilor unor regiștrii folosiți de programul apelant. Soluția este salvarea înainte a acestor regiștrii în memorie.

```
subu $sp,$sp,32       # Se alocă 32 bytes in stivă
sw $ra,20($sp)        # Se salvează adresa de revenire curentă
sw $fp,16($sp)        # Se salvează valoarea curentă a registrului $fp
                      (registru folosit pentru parcurgerea cadrului actual
                      din stivă)
sw $a0,12($sp)        # Se salvează valoarea curentă a registrului $a0
addiu $fp,$sp,28      # Se configurează noua valoare a registrului $fp ca
                      fiind noul început al stivei
```

În concluzie, în limbajul de asamblare MIPS, următorii pași sunt necesari pentru ca un apel de subprogram să nu perturbe funcționarea programului apelant:

- codul apelant plasează argumentele către subprogram scriind regiștrii `$a0-$a3` cu valori
- codul apelant execută saltul către subprogram apelând jal urmată de eticheta subprogramului. Adresa de revenire, PC+4, unde PC este adresa instrucțiunii jal este salvată în registrul `$ra`.
- în cazul în care subprogramul folosește registrul `$fp` (frame pointer), acest registrul va fi setat cu valoarea vârfului stivei. Fosta valoare a registrului `$fp` va fi salvată în stivă.
- subprogramul salvează toți regiștrii ce vor fi modificați în stiva de memorie. Dacă subprogramul apelează alt subprogram în timpul execuției, atunci este necesară și salvarea registrului `$ra` în stivă.
- după execuția tuturor operațiilor necesare calculării rezultatului, valoarea este stocată în `$v0-$v1`.
- Subprogramul rulează instrucțiunea `jr $ra` pentru a reveni.

4. Servicii disponibile în SPIM

Simulatorul SPIM pune la dispoziție o serie de subrutine de tipul serviciilor de sistem de operare prin intermediul instrucțiunii MIPS syscall (system call). Pentru a selecta o subrutină, programul încarcă un cod în registrul *$v0* și argumentele în regiștrii argument *$a0*, ..., *$a3* (sau *$f12* pentru operațiile în virgulă mobilă). Rezultatul este returnat în registrul *$v0* (sau *$f0* pentru operațiile în virgulă mobilă).

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $v0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $v0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $v0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

Exemplu : citirea unui întreg

```
li $v0, 5 # încărcarea codului read_int în $v0
syscall
addu $t0, $0, $v0 # mutarea numărului citit în $t0
```

Exemplu : afișarea unui întreg

```
addu $a0, $0, $t2 # mutarea numărului a fi printat în $a0
li $v0, 1 # încărcarea codului print_int în $v0
syscall
```

Exerciții

1. Scrieți un subprogram ce calculează lungimea unui string. Testați subprogramul apelându-l de mai multe ori într-un program pentru stringuri diferite
2. Scrieți un program ce va calcula elementul maxim dintr-un vector. Folosiți apeluri de subprograme.

   Creați un subprogram și pentru calcularea minimului dintr-un vector.