INVESTIGATII ARHITECTURALE UTILIZÂND SIMULATORUL SPIM

1. Scopul lucrarii

Lucrarea de fata urmareste studiul benchmark-urilor existente, deprinderi practice privind programarea în asamblare MIPS.

2. Desfasurarea lucrarii

2.1. Note explicative referitoare la benchmark-ul Input.s

Benchmark-ul *Input.s* este un program de test simplu, care utilizeaza pseudoinstructiunile ce expandeaza în apeluri sistem, prezentate în lucrarea de laborator anterioara, pentru afisarea pe fereastra Consola a sirurilor de caractere, valori întregi, caractere, pentru citirea de la tastatura a variabilelor de diverse tipuri (**int, string, char**), pentru terminarea programului. Se vor observa cu ajutorul simulatorului prin intermediul ferestrelor **Data**, **Registru**, **Sesiune** si **Consola** valorile din registrii, continutul locatiilor de memorie, instructiunile efectiv executate în urma rularii pas cu pas sau în mod continuu a programului.

```
.data
d1:    .asciiz "gimme an int: "
d2:    .asciiz "gimme a char: "
d3:    .asciiz "gimme a string: "
dog:   .space 80
```

Zona de date utilizator începe la adresa 10010000h. Sirul de la eticheta d1 este memorat de la aceasta adresa pe 15 octeti (14 caractere plus un octet pentru terminatorul NULL).

d1: 10010000h → 1001000Eh (15 octeti)

d2: 1001000Fh → 1001001Dh (15 octeti)

d3: 1001001Eh → 1001002Fh (17 octeti)

dog: 10010030h → 10010080h (80 octeti de spatiu)

Initial valorile din registrii generali sunt 0. Registrul Program Counter are valoarea PC=00400000h, care este adresa primei instructiuni a programului utilizator. Registrul pointer global la date, GP=10008000h, indica spre mijlocul unei zone de date de 64K octeti (de la adresa 10000000h - la adresa 1000FFFFh) în segmentul de date statice.

Comentariu asupra secventei de instructiuni:

Operatia efectuata	Opcode-ul Pseudoinstructiunii	Nr. crt. Instr. Simpla	Expandare în instructiuni MIPS simple	Semnificatie instructiune simpla
# afisare string si citire întreg de la tastatura	puts d1	1.	lui $4, 4097	încarcare în registrul a_0 a adresei de unde se va afisa stringul.
		2.	ori $2, $0, 4	încarcare în v_0 a codului de afisare string pentru apelul sistem.
		3.	syscall	afisare pe consola a stringului de la

				eticheta d1.
# citire de la tastatura în registrul a_0	geti $\$a_0$	1.	ori $2, $0, 5	încarcare în v_0 a codului operatiei de citire întreg.
		2.	syscall	citire de la tastatura a unui întreg; rezultatul este depus în v_0.
		3.	add $4, $2, $0	transfer valoare din v_0 în a_0.
# incrementare cu 1	addi $\$a_0$, $\$a_0$, 1	1.	addi $4, $4, 1	incrementare cu 1 a registrului a_0.
# afisare continut registru a_0	puti $\$a_0$	1.	ori $2, $0, 1	încarcare în v_0 a codului de afisare întreg
		2.	addi $4, $4, $0	pregatire în a_0 argument pentru apel sistem
		3.	syscall	afisare pe consola a întregului din $\$a_0$.
# afisare caracter ‘\n’	putc ‘\n’	1.	ori $4, $0, 10	pune în a_0 codul ASCII al caracterului ‘\n’ (0x0A)
		2.	ori $2, $0, 11	încarcare în v_0 a codului de afisare

				caracter
		3.	syscall	afisare caracter pe consola
# afiseaza mesaj de la eticheta d2	puts d2	1.	lui $1, 4097	încarcare octet superior de adresa în registrul at.
		2.	ori $4, $1, 15	încarcare în a_0 adresa etichetei d2.
		3.	ori $2, $0, 4	încarcare în v_0 a codului de afisare string pentru apelul sistem.
		4.	syscall	afisare pe consola mesaj de la eticheta d2
# citeste un caracter în registrul a_0	getc $$a_0$$	1.	ori $2, $0, 12	încarcare în v_0 a codului operatiei de citire caracter.
		2.	syscall	citire de la tastatura a unui caracter; codul sau ASCII este depus în v_0.
		3.	add $4, $2, $0	transfer valoare din v_0 în a_0.
# incrementeaza	addi $$a_0$$, $$a_0$$, 1	1.	addi $4, $4, 1	incrementare cu 1 a registrului a_0

cu 1 codul ASCII al caracterului citit anterior				
# afisare continut registru a_0	puti a_0	1.	ori $2, $0, 11	încarca în v_0 codul de afisare caracter.
		2.	addi $4, $4, $0	pregatire în a_0 argument pentru apel sistem
		3.	syscall	afisare pe consola a codului ASCII din a_0.
# afisare caracter '\n'	putc '\n'	1.	ori $4, $0, 10	pune în a_0 codul ASCII al caracterului '\n' (0x0A)
		2.	ori $2, $0, 11	încarcare în v_0 a codului de afisare caracter
		3.	syscall	afisare caracter pe consola
# încarcare adresa eticheta d3	la a_0, d3	1.	lui $1, 4097	încarcare octet superior de adresa în registrul at.
		2.	ori $4, $1, 30	încarcare în a_0 adresa etichetei d3.

# încarcare cod afisare string de la eticheta d3	li v_0$, 4	1.	ori $2, $0, 4	încarcare în v_0 a codului de afisare string pentru apelul sistem.
# apel sistem	syscall	1.	syscall	afisare mesaj de la eticheta d3
# încarcare adresa eticheta dog	la a_0$, dog	1.	lui $1, 4097	încarcare octet superior de adresa în registrul at.
		2.	ori $4, $1, 47	încarcare în a_0 adresa etichetei d3.
# încarcare în registru a_1 a lungimii sirului	li a_1$, 80	1.	ori $5, $0, 80	încarcare în a_1 a lungimi sirului de la eticheta dog
# încarcare cod citire string de la tastatura	li v_0$, 8	1.	ori $2, $0, 8	încarcare în v_0 a codului de citire string pentru apelul sistem.
# apel sistem	syscall	1.	syscall	citire string de la tastatura în octetii de la adresa specificata de eticheta dog.
# afiseaza string citit anterior de la	puts dog	1.	lui $1, 4097	încarcare octet superior de adresa în registrul at.

eticheta dog				
		2.	ori $4, $1, 47	încarcare în a_0 adresa etichetei dog.
		3.	ori $2, $0, 4	încarcare în v_0 a codului de afisare string pentru apelul sistem.
		4.	syscall	afisare pe consola mesaj de la eticheta dog
# iesire din program	done	1.	ori $2, $0, 10	încarcare în v_0 a codului de iesire din program pentru apelul sistem.
		2.	syscall	apel sistem pentru iesirea din program

2.2. Probleme propuse spre rezolvare

1. Scrieti si testati un program în limbaj de asamblare MIPS care sa calculeze si afiseze primele 24 numere prime. Un numar *n* este prim daca el se divide exact doar cu 1 si cu el însusi. Pentru usurinta vom implementa întâi o rutina *test_prime(n)*, care întoarce 1 (sau un mesaj "Numarul este prim!") daca numarul este prim si 0 (sau mesaj "Numarul nu este prim!") altfel. Folosindu-ne apoi de aceasta rutina vom scrie un program care parcurge numerele naturale si testeaza daca fiecare este prim. Se vor afisa primele astfel de 24 numere, fiecare pe un

rând nou [n (initial 24) - va putea lua ulterior orice valoare]. Testarea programului se va face pe simulatorul SPIM. Se vor rula programele pas cu pas, se vor modifica parametrii (numarul de test - în cazul rutinei *testprime* si numarul de numere prime generate - în cazul programului de *generare a numerelor prime*) si se va verifica corectitudinea rezultatelor. Întrucât în fereastra Consola nu încap decât maxim 25 numere prime pe o linie si consola prezinta maxim 24 de linii, se va modifica programul astfel încât sa poata fi afisate maxim n=24×25 numere prime.

Subrutina *test_prime*

```
        .data

d1:     .asciiz "Numarul este prim! "
d2:     .asciiz "Numarul nu este prim!"

        .text
        .globl main
main:
        li      $a0, 137            # Numarul pe care-l testez (n)
        li      $v0, 2              # Primul numar prim (initial i=2)
ET:     sub     $v0, $v0, $a0       # i = i-n
        beqz    $v0, $L1            # Testez i<0
        add     $v0, $v0, $a0       # Refac i; i = i + n;
        divu    $a0, $v0            # Calculez restul împartirii lui n la i
        mfhi    $a1                 # Pun restul în a1
        beqz    $a1, $L2            # Daca restul e 0 salt la eticheta - numarul nu e
                                    # prim
        addi $v0, $v0, 1             # Cresc i-ul - urmatorul numar cu care-l împart
```

```
                                    # pe n
        j       ET

$L1:
        li      $a0, d1             # Numarul nu se împarte exact cu nici unul din
                                    # numerele mai mici decât el, numarul este prim
        li      $v0, 4
        syscall                     # Afisare mesaj de la eticheta d1
        j       END                 # Salt la sfârsitul programului

$L2:
        li      $a0, d2             # Numarul nu este prim
        li      $v0, 4
        syscall                     # Afisare mesaj de la eticheta d2
END:
        done
```

Solutia problemei 1

```
        .text
        .globl main
main:
        li      $a2, 24             # Numarul de numere prime
        li      $a0, 2              # Primul numar prim (numarul prim pe care-l
                                    # testez)
        li      $a1, 0              # Contorizarea numerelor prime
ET1:
        sub     $a1, $a1, $a2       # Mai sunt numere prime de afisat ?
        beqz    $a1, END
```

```
        add     $a1, $a1, $a2       # Refacere contor
        jal     test_prime          # Salt la testare numar prim

RESTART:
        add     $a0, $a0, 1         # Urmatorul numar de testat
        j       ET1

test_prime:
        li      $v0, 2              # Primul numar cu care se imparte numarul testat
ET:
        sub     $v0, $v0, $a0       # Am testat toate numerele mai mici decat n ?
        beqz    $v0, $L1
        add     $v0, $v0, $a0       # Refacere $v0
        divu    $a0, $v0            # Calculeaza restul impartirii lui n la valorile
                                    # anterioare lui n mod i
        mfhi    $a3                 # Restul impartirii este depus in a3
        beqz    $a3, END_PROC       # Daca restul este 0 numarul n nu este prim
        addi    $v0, $v0, 1         # Crestem i-ul urmator numar cu care-l impartim
                                    # pe n
        j       ET

$L1:
        add     $a0, $a0, $0        # Afisare numar prim pe ecran
        li      $v0, 1
        syscall
        add     $a1, $a1, 1         # Creste numarul numerelor prime afisate cu 1

        subu    $sp, $sp, 32        # Pregatire stiva pentru scrierea numarului
        sw      $ra, 20($sp)        # urmator care se va testa daca este prim
```

```
        sw      $fp, 16($sp)
        addu    $fp, $sp, 32
        sw      $a0, 0($fp)         # Memoreaza a0

        putc '\n'                   # Afiseaza un retur de car pentru ca urmatorul
                                    # numar prim sa apara pe rand nou
        lw      $a0, 0($fp)         # Refacere a0 pentru a testa daca este nr. prim
        lw      $ra, 20($sp)        # Reface adresa de revenire
        lw      $fp, 16($sp)        # Reface pointer de cadru
        addu    $sp, $sp, 32        # Reface pointer-ul de stiva

END_PROC:
        j       RESTART

END:
        done
```

2. Scrieti un program care sa calculeze factorialul unui numar natural n(de exemplu n=10). Se va folosi o rutina de calcul recursiv *fact*, care-l obtine pe n! din înmultirea lui (n-1)! cu n. Codul scris în limbaj de asamblare al acestei rutine ilustreaza cum programul afecteaza stiva, registrii $sp, $fp, $ra. Programul principal *main* creeaza cadrul de stiva si salveaza registrii $fp si $ra (pointer-ul de cadru si adresa de revenire din procedura). Vom considera dimensiunea minima a stivei de 32 octeti, mai mare decât ar fi necesar pentru memorarea celor doua registre. Sa se ruleze programul pas cu pas, sa se urmareasca continutul stivei si al registrilor, actualizarea registrilor la revenirea din subrutina si corectitudinea rezultatelor.

Solutia problemei 2

```
        .text
        .globl main
main:
        subu   $sp, $sp, 32        # Cadrul de stiva de 32 bytes
        sw     $ra, 20($sp)        # Salveaza adresa de revenire
        sw     $fp, 16($sp)        # Salveaza vechiul pointer de cadru
        addu   $fp, $sp, 32        # Seteaza noul pointer de cadru
```

Programul principal, care începe la eticheta *main*, apeleaza rutina *fact* si paseaza acesteia singurul sau argument n (initial 10). La revenirea din subrutina se va afisat mesaj plus valoarea factorialului.

```
        li     $a0, 10             # Pune argumentul în $a0
        jal    fact                # Apeleaza rutina de calcul a factorialului

        la     $a0, $LC            # Pune adresa string-ului de afisat în $a0
        li     $v0, 4              # Pune codul operatiei în v0
        syscall                    # Afiseaza mesaj
```

La sfârsit, dupa afisarea valorii factorialului, programul se încheie, dar nu înainte de a restaura registrii salvati în stiva si refacerea vechiului cadru de stiva.

```
        mflo   $a0                 # Salveaza rezultatul în a0
        li     $v0, 1              # Pune codul operatiei în v0
        syscall                    # Afiseaza valoarea factorialului

        lw     $ra, 20($sp)        # Reface adresa de revenire
```

```
        lw      $fp, 16($sp)        # Reface pointer-ul de cadru
        addu    $sp, $sp, 32        # Reface cadrul de stiva

        done                        # Încheiere program

        .data
$LC:
        .ascii "The factorial of 10 is = "
```

Rutina factorial are stuctura similara cu programul principal. Întâi se creeaza cadrul de stiva se salveaza registrii care se folosesc $a0, $fp si $ra.

```
        .text
fact:
        subu    $sp, $sp, 32        # Cadrul de stiva este de 32 bytes
        sw      $ra, 20($sp)        # Salveaza adresa de revenire
        sw      $fp, 16($sp)        # Salveaza vechiul pointer de cadru
        addu    $fp, $sp, 32        # Seteaza noul pointer de cadru

        sw      $a0, 0($fp)         # Salveaza argumentul (n)
```

Nucleul acestei rutine realizeaza calculul efectiv n! = n × (n-1)!. Se testeaza daca argumentul este mai mare decât 0. Daca nu, rutina returneaza valoarea 1. Daca este mai mare ca 0, se reapeleaza recursiv rutina *fact*, pentru a calcula (n-1)! si se înmulteste cu n.

```
        lw      $2, 0($fp)          # Load n
        bgtz    $2, $L2             # Branch if n>0
        li      $2, 1               # Returneaza 1
        j       $L1                 # Salt la codul de revenire
```

```
$L2:
        lw      $3, 0($fp)          # Load n
        subu    $2, $3, 1           # Calculeaza n-1
        move    $a0, $2             # Muta valoarea în $a0
        jal     fact                # Reapeleaza rutina de calcul a factorialului

        lw      $3, 0($fp)          # Load n
        mul     $2, $2, $3          # Calculeaza fact(n-1)*n
```

În final, rutina factorial restaureaza registrii si returneaza valoarea factorialului în registrul $v0.

```
$L1:                                # Rezultatul în $2
        lw      $ra, 20($sp)        # Restaureaza $ra
        lw      $fp, 16($sp)        # Restaureaza $fp
        addu    $sp, $sp, 32        # Reface cadrul de stiva
        j       $ra                 # Revenire în programul apelant
```

3. Scrieti un program care sa sorteze un sir de *n* (initial n=5) numere întregi prin metoda *bubblesort*. Rulati programul pentru diverse siruri de numere de dimensiuni diferite.

Solutia problemei 3

```
        .data
sir:
        .word 2,1,7,9,5
        .text
```

```
        .globl main
main:
        li $a0, 0
tsgata:
        bnez   $a0, term
        li     $a0,1
        li     $a1, 5          # a1=n-1
        li     $a2, 0          # a2=j, indicele în sir
for:
        sub    $a2, $a2, $a1
        beqz   $a2, tsgata
        add    $a2, $a1, $a2
        la     $t0, sir
        li     $t1, 4
        mul    $a3, $a2, $t1
        add    $a3, $t0, $a3   # adr=sir+4*j
        lw     $t2, ($a3)      # sir[j]
        add    $t4, $a2, 1
        mul    $a3, $t4, $t1
        add    $a3, $t0, $a3
        lw     $t3, ($a3)      # sir[j+1]
        sub    $t2, $t2, $t3
        bgtz   $t2, schimba
        add    $a2, $a2, 1
        j      for
schimba:
        add    $t2, $t2, $t3
        add    $t5, $t2, $0
        add    $t2, $t3, $0    # t2=sir[j+1]
```

```
        add     $t3, $t5, $0            # t3=sir[j]
        sw      $t3, ($a3)
        sub     $a3, $a3, $t1
        sw      $t2, ($a3)
        li      $a0, 0
        j       for
term:
        done
```

Observatii:

1. O deficienta a simulatorului SPIM consta în imposibilitatea de a varia diferiti parametrii arhitecturali ai procesorului (latenta instructiunilor, numar unitati de executie), precum si validarea sau invalidarea unor tehnici gen *forwarding*.
2. Modul preferat de rulare al programelor este pas cu pas. Programele pot fi executate si prin puncte de întrerupere, dificultatea consta însa în cunoasterea exacta a adreselor instructiunilor pe care se stabileste întreruperea.

Bibliografie

[1] **Larus J.** - *SPIM S20: A MIPS R2000 Simulator*, Morgan Kaufmann Publishers, 1993