# INVESTIGATII ARHITECTURALE UTILIZÂND SIMULATORUL DLX

## 1. Scopul lucrarii

Lucrarea de fata urmareste studierea programelor de test, deprinderi practice privind programarea în limbajul de asamblare al procesorului DLX, precum si o analiza cantitativa si calitativa a rezultatelor obtinute în urma simularii efectuate pe benchmark-urile existente.

## 2. Desfasurarea lucrarii

### 2.1. Apeluri sistem

Pentru citirea de la tastatura si afisarea în fereastra "DLX I/O" a sirurilor de caractere, valorilor întregi, caractere, pentru terminarea programului, sunt utilizate apeluri sistem, un mic set de servicii ale sistemului de operare prin instructiuni (**trap**).

Instructiunile **trap** sunt similare apelurilor sistem UNIX/DOS, respectiv functiilor din biblioteca C - open(), close(), read(), write() si printf(). Orice fisier înainte de a fi prelucrat trebuie deschis. Functia open() determina deschiderea unui fisier existent si returneaza o valoare întreaga pozitiva numita *descriptorul de fisier*. Acesta identifica în continuare fisierul respectiv în toate operatiile realizate asupra lui. Functia read() primeste ca si parametru descriptorul unui fisier deschis în prealabil prin intermediul functiei open si realizeaza operatia de citire. Ea returneaza numarul de octeti cititi efectiv din fisier. Functia write() este similara cu read(), doar ca realizeaza transferul de date în sens invers, din memoria principala în fisier. La terminarea prelucrarii unui fisier acesta trebuie închis cu ajutorul functiei close(). Aceasta returneaza 0 la o închidere reusita si –1 în caz de eroare.

Descriptorii de fisier 0,1 si 2 sunt rezervati fisierelor standard - stdin, stdout and stderr. Intrarile si iesirile DLX pot fi controlate cu acesti descriptori. Adresa parametrilor necesari apelurilor sistem trebuie încarcata în registrul R14. Toti parametrii trebuie sa fie pe 32 de biti, exceptie facând registrii flotanti dubla precizie care sunt pe 64 de biti. Sirurile de caractere sunt referite cu adresa lor de început. Rezultatul apelului va fi returnat în registrul R1. Daca apare vreo eroare în timpul apelului sistem, registrul R1 este setat cu valoarea -1 si, daca simbolul "_errno" este setat la valoarea A atunci se returneaza un cod de eroare la adresa de memorie A iar simularea continua, altfel simularea este întrerupta [1].

| Serviciul | Codul Apel Sistem | Argumentele | Rezultatul |
|---|---|---|---|
| Open file | 1 | 1. Numele fisierului: adresa unui sir încheiat cu terminatorul null, care contine calea spre fisierul care va fi deschis.<br>2. Mod: modul de deschidere a fisierului.<br>3. Flaguri suplimentare: drepturi de executie | Deschiderea unui fisier pentru citire sau scriere. Fisierele deschise sunt automat închise la resetarea DLX sau oprirea WinDLX.<br>Descriptorul de fisier este returnat în registrul R1. |
| Close file | 2 | 1. Descriptorul fisierului ce urmeaza a fi închis. | Un fisier descris anterior cu Trap 1 este închis.<br>În caz de succes în registrul R1 se returneaza 0, altfel -1. |
| Read block from file | 3 | 1. Descriptorul fisierului din care se citeste blocul. | Un bloc dintr-un fisier sau o linie de la intrare (stdin) |

| | | | |
|---|---|---|---|
| | | 2. Adresa zonei unde se va stoca blocul citit.<br>3. Dimensiunea blocului ce va fi citit (în octeti) | poate fi citit cu acest apel sistem. Numarul actual de octeti citit este returnat în registrul R1.. |
| Write block to file | 4 | 1. Descriptorul de fisier în care se va scrie.<br>2. Adresa blocului ce va fi scris.<br>3. Dimensiunea blocului (în octeti). | Un bloc poate fi scris în memorie sau la iesirea standard. Numarul de octeti efectiv scrisi este returnat în registrul R1. |
| Formatted Output to Standard Output | 5 | 1. Formatul de afisare al string-ului.<br>2. Argumente în conformitate cu formatul de afisare. | Echivalentul functiei de biblioteca printf(). Numarul de octeti transferati la iesire (stdout) este returnat în registrul R1. |
| Exit | 0 | - | Încheierea programului. |

Detalii suplimentare legate de modurile de deschidere a fisierelor, drepturilor de acces si executie asupra acestora pot fi gasite studiind functiile de biblioteca C. [2]

## 2.2. Note explicative referitoare la benchmark-ul Fact.s

Benchmark-ul *Fact.s* este un program de test simplu, care calculeaza si afiseaza pe ecran factorialul unui numar, introdus de la tastatura cu ajutorul modulului *Input.s* si memorat în registrul $R_1$ al procesorului DLX.

Se vor observa cu ajutorul simulatorului prin intermediul ferestrelor **Pipeline**, **Registru**, **Cod, Diagrama ciclului de tact, Statistica,** valorile din

registri, continutul locatiilor de memorie, instructiunile efectiv executate în urma rularii pas cu pas sau în mod continuu a programului, nivelul pipeline în care se afla instructiunile în fiecare ciclu de tact, precum si rezultatele simularii.

```
;**** WINDLX : Calculul Factorialului unui numar ****
;-------------------------------------------------------------------------------------------
;Programul necesita subrutina INPUT.s
;Se citeste un numar de la tastatura si calculeaza factorialul sau
;(tipul rezultatului: double)
; Rezultatul este afisat pe consola
;-------------------------------------------------------------------------------------------

            .data
                        ;incepe segmentul de date - valoare implicita 0x1000
Prompt:     .asciiz     "An integer value >1 : "
                        ;0x1000 - adresa de inceput a segmentului de date
                        ;memoreaza un string de 0x16 octeti

PrintfFormat: .asciiz   "Factorial = %g\n\n"
                        ;0x1017 - adresa la care se memoreaza un sir de 0x10
                        ;octeti
            .align 2
                        ;datele vor fi memorate pe cuvant
PrintfPar:  .word PrintfFormat
                        ;0x1028 - adresa la care se memoreaza un cuvant pe 4
                        ;octeti (adresa 0x1017)
PrintfValue: .space8
                        ;0x102C - se aloca 8 octeti necesari stocarii rezultatului
```

```
        .text
                ;segmentul de cod incepe la simbolul main - valoare
                ;implicita 0x100
        .global main
main:
;*** Se apeleaza procedura de intrare pentru citirea numarului de la tastatura ***

        addi   r1,r0,Prompt     ;r1<-adresa mesaj de la eticheta <Prompt>
        jal InputUnsigned       ;apel subrutina Input.s unde va avea loc
                                ;citirea efectiva a numarului de la tastatura

;*** Initializare valori ***
;*** La iesirea din procedura Input.s numarul citit se afla in registrul r1 ***

        movi2fp f10,r1          ;r1 -> d0 d0 - folosit pe post de registru contor
        cvti2d f0,f10           ;converteste valoarea intreaga din f10 in dubla
                                ;precizie si o depune in f0
        addi r2,r0,1            ;1 -> d2 d2 - pastreaza rezultatul. Initializat cu 1.
        movi2fp f11,r2          ;r2 -> f11
        cvti2d f2,f11           ;converteste valoare intreaga din f11 in dubla
                                ;precizie in f2
        movd  f4,f2             ;1  -> d4 d4 - constanta 1

;*** Iesirea din bucla se face daca d0 = 1 ***

Loop:   led f0,f4               ;se testeaza daca d0<=1
        bfpt Finish             ;salt la eticheta Finish daca d0<=1

;*** Inmultire si reapelare bucla ***
        multd f2,f2,f0          ;rezultat <- rezultat*contor [f2 <- (f2*f0)]
```

```
            subd f0,f0,f4       ;decrementare contor cu constanta 1 [f0<- (f0-1)]
            j Loop              ;rebuclare

Finish:     ;*** Afisare rezultat pe consola ***
            sd PrintfValue,f2   ;memoreaza rezultat (dubla precizie) la adresa
                                ;PrintfValue
            addi r14,r0,PrintfPar   ;pregatire adresa pentru afisare mesaj +
                                    ;rezultat
            trap 5              ;intrerupere software ce are ca efect afisarea pe
                                ;ecran a continutului locatiei de memorie de la
                                ;adresa data de registrul r14

;*** Incheiere program ***
            trap 0              ;intrerupere fara efect asupra DLX (procesor +
                                ;memorie) care anunta incheierea executiei
                                ;programului
```

**Comentariul subrutinei Input.s**

```
;*************** WINDLX: Citirea unui numar întreg pozitiv ***************
;-------------------------------------------------------------------------------------------
;Apelul subprogramului se face prin salt la simbolul "InputUnsigned"
;În momentul apelului subrutinei, în R1 trebuie sa existe adresa unui sir care se
;încheie cu terminatorul null
;Valoarea citita de la tastatura este returnata în R1
;Se modifica continutul registrilor R1,R13,R14
;-------------------------------------------------------------------------------------------
```

```
        .data

;*** Zona de date necesara pentru instructiunea trap de citire ***

ReadBuffer: .space80
                        ;0x1034 – adresa la care se memoreaza valoarea citita
ReadPar:    .word 0,ReadBuffer,80
                        ;0x1084 – adresa unde se gasesc parametrii apel trap 3

;*** Zona de date necesara pentru instructiunea trap de scriere ***

PrintfPar:  .space4
                        ;0x1090
SaveR2:     .space4
                        ;0x1094 – 4 octeti necesari pentru memorarea lui R2
SaveR3:     .space4
                        ;0x1098 – 4 octeti necesari pentru memorarea lui R3
SaveR4:     .space4
                        ;0x109C – 4 octeti necesari pentru memorarea lui R4
SaveR5:     .space4
                        ;0x10A0 – 4 octeti necesari pentru memorarea lui R5

            .text
;segmentul de cod al subrutinei începe la simbolul global InputUnsigned – 0x144
InputUnsigned:

            .global     InputUnsigned
;*** Salvarea continutului registrelor R2, R3, R4, R5 ***
```

```
        sw SaveR2,r2        ;memorare registrul R2 la adresa 0x1094
        sw SaveR3,r3        ;memorare registrul R3 la adresa 0x1098
        sw SaveR4,r4        ;memorare registrul R4 la adresa 0x109C
        sw SaveR5,r5        ;memorare registrul R5 la adresa 0x10A0

;*** Salvare registrul R1 – contine adresa sirului "An integer value >1 :" ***

        sw PrintfPar,r1     ;memorare registrul R1 la adresa 0x1090
        addi r14,r0,PrintfPar
                            ;pregatire parametru pentru apelul sistem trap 5
        trap 5              ;apel  sistem de afisare mesaj în fereastra DLX–
                            ;I/O

;*** Apel sistem de citire valoare întreaga de la tastatura ***
        addi r14,r0,ReadPar     ;pregatire parametru pentru apel trap 3
        trap 3              ;apel  sistem de citire valoare în fereastra DLX –
                            ;I/O

;*** Determinarea valorii citite; în urma apelului sistem valoarea citita [codul
;Ascii al fiecarui octet + terminatorul null (0x0A)] este memorat la adresa data de
;ReadBuffer ***
        addi r2,r0,ReadBuffer   ;R2 – initializat cu adresa simbolului
                                ;ReadBuffer
        addi r1,r0,0        ;R1 – initializat cu 0
        addi r4,r0,10       ;R4 – specifica sistemul de numeratie zecimal

;*** Citeste într-o bucla toti octetii pâna la caracterul null ***
Loop:
        lbu r3,0(r2)        ;transfer în R3 octetul curent de la ReadBuffer
```

```
seqi r5,r3,10      ;LF -> Exit. Setare registru R5 daca octetul
                   ;curent este terminatorul null
bnez r5,Finish     ;daca R5<>0 s-a încheiat detectarea octetilor
subi r3,r3,48      ;refacere în R3 a valorii întregi: scadere cod
                   ;Ascii numar 0
multu r1,r1,r4     ;shift zecimal (R1<- R1*10)
add r1,r1,r3       ;R1<- R1+R3
addi r2,r2,1       ;incrementare pointer – adresare urmatorul octet
                   ;de la adresa data de ReadBuffer
j Loop             ;reapel bucla

;*** Refacere continut registri ***
Finish:
lw r2,SaveR2       ;refacere registru R2 cu valoarea de la adresa
                   ;0x1094
lw r3,SaveR3       ;refacere registru R3 cu valoarea de la adresa
                   ;0x1098
lw r4,SaveR4       ;refacere registru R4 cu valoarea de la adresa
                   ;0x109C
lw r5,SaveR5       ;refacere registru R5 cu valoarea de la adresa
                   ;0x10A0
jr r31             ;revenire din subrutina în programul apelant
```

## 3. Probleme propuse spre rezolvare

I. Pornind de la exemplul prezentat anterior, comentati si apoi rulati programele de test **prim.s** si **gcm.s**.

II. Scrieti un program care citeste *n* numere de la tastatura prin intermediul modulului **Input.s** si calculeaza suma numerelor si le depune succesiv în memoria DLX la adresa 0x1500.

III. Numim CONFIGURATIE DE BAZA a procesorului RISC DLX, urmatoarea configuratie arhitecturala:

1 unitate executie numere întregi, latenta 1 ciclu;
1 unitate executie adunare flotant (FPP ADD), latenta 2 cicli;
1 unitate executie înmultire flotant (FPP MUL), latenta 5 cicli;
1 unitate executie împartire flotant (FPP DIV), latenta 19 cicli;
Nu e implementata tehnica "forwarding".

În acest context, în continuare, se cere:

1. Rata de procesare (IR, în instructiuni/ciclu), masurata pe benchmark-ul dat, pentru configuratia de baza DLX.

2. Cât devine rata de procesare (comparativ cu cazul precedent), daca se considera 2 respectiv 8 unitati de executie FPP ADD (în rest, sunt pastrate caracteristicile configuratiei de baza).

3. IR, daca se considera 2 respectiv 8 unitati de executie FPP MUL (în rest, configuratia de baza).

4. IR, daca se considera 2 respectiv 8 unitati de executie FPP DIV (în rest, configuratia de baza).

5. IR, daca fata de configuratia de baza se activeaza optiunea "forwarding".

6. IR, pentru variatia latentei unitatii de executie FPP ADD de la 1 la 5 cicli.

7. IR, pentru latenta unitatii FPP MUL de 1 respectiv 20 cicli.

8. IR, pentru latenta unitatii FPP DIV de 1 respectiv 32 cicli.

**Observatii**

1. Rezultatele obtinute la punctele III.2, III.3, III.4 sunt identice cu cel obtinut la punctul III.1. Explicatia o regasim daca privim în fereastra Statistica, si observam ca nu exista stagnari datorate hazardurilor structurale. Aceasta implica faptul ca, oricât de multe unitati de executie ar exista, pentru simularea optima a acestor benchmark-uri sunt necesare doar câte o unitate de executie în flotant de fiecare tip (ADD, MUL, DIV).

2. Tehnica de *forwarding* determina cresterea ratei de procesare în procente variabile (pâna la 22.97% pe **fact.s**), în functie de benchmark si functie de valorile parametrilor cu care sunt apelate programele de test. Cu cât aceste valori sunt mai mici, cu atât cresterea de performanta este mai accentuata (estimatie facuta pe acelasi benchmark - **fact.s**).

3. Este evidenta o diminuare a ratei de procesare odata cu cresterea latentei de executie a instructiunilor în virgula mobila. Varianta optima din acest punct de vedere este cea oferita de configuratia de baza.

**Bibliografie**

[1] **Gründbacher H. -** *Windows Deluxe Simulator - Tutorial*, University of Technology, Viena, 1992.

[2] **Negrescu L.** – *Introducere în limbajul C* - Editura MicroInformatica, Cluj Napoca., 1993