UTILIZAREA SIMULATORULUI SPIM

1. Scopul lucrarii

Lucrarea de fata prezinta o descriere detaliata a simulatorului SPIM. Volumul mare de informatie poate ascunde faptul ca SPIM este simplu si usor de folosit ca program. Este o lucrare de laborator rapida despre SPIM, cu scopul de a încarca, depana si rula programe MIPS simple.

2. Memento teoretic

SPIM/SAL este o portare pe WIN32S a SPIM, un simulator scris pentru uz didactic în Facultatea de Calculatoare a Universitatii din Wisconsin. WIN32S este o extensie pe 32 de biti a sistemului Windows 3.1. Interfata programabila cu aplicatia a WIN32S este un subset al WIN32, un API suportat de sistemul de operare Windows NT. Aplicatii ce folosesc WIN32S API, cum sunt SPIM/SAL, sunt compilate într-un format executabil numit *executabil portabil* (PE) care ruleaza fie pe Microsoft Windows 3.1. cu extensie WIN32S fie pe Microsoft Windows NT.

SPIM apare în doua versiuni: una simpla care se numeste *spim*. Ea ruleaza pe orice tip de terminal. Se tasteaza comanda la terminal iar *spim* o executa. Versiunea superioara se numeste *xpim* si ruleaza pe un sistem X - window. *Xpim* este mult mai usor de învatat si folosit deoarece comenzile sunt întotdeauna vizibile pe ecran si afiseaza permanent continutul registrilor, dar necesita un display mapat pe bit.

SPIM S20 este un simulator care ruleaza programe pentru calculatoare RISC cu procesor MIPS R2000 / R3000. SPIM poate citi si executa imediat fisiere scrise în limbaj de asamblare sau executabile MIPS. El contine un debugger si asigura câteva servicii specifice sistemelor de operare. SPIM este

mult mai încet decât un calculator real (aproximativ 100 de ori). Totusi, costul scazut si larga aplicabilitate nu poate fi egalata de hardware-ul adevarat.

Se pune deci întrebarea: "De ce folosim un simulator când multi utilizatori au statii de lucru ce contin cip-uri MIPS care sunt semnificativ mai rapide decât SPIM ?"

Un motiv ar fi ca aceste statii nu sunt universal folositoare. Un altul ar fi progresul spre calculatoare noi si rapide, facând aceste masini demodate. Tendinta curenta este de a face calculatoarele mai rapide prin executia concurenta a mai multor instructiuni. Aceasta face arhitecturile mai dificil de înteles si programat. În plus, simulatoarele pot asigura un mediu mai bun de programare decât o masina existenta, deoarece ele pot detecta mai multe erori si pot asigura mai multe caracteristici decât un calculator actual. Deci, simulatoarele sunt unelte folositoare în studierea calculatoarelor si programelor care ruleaza pe ele. Deoarece sunt implementate software si nu hardware, simulatoarele pot fi usor modificate adaugând instructiuni noi, construind sisteme noi cum ar fi cele multiprocesor, sau se pot colecta si analiza date.

Implicit, SPIM simuleaza masina virtuala cu un bogat set de instructiuni. Totusi, el poate simula si hardware-ul simplu. În continuare vom descrie masina virtuala si vom mentiona pe scurt caracteristicile care nu apartin harware-ului existent. Pentru aceasta, vom urma conventiile programatorilor în limbaje de asamblare MIPS, care este folosit în mod obisnuit de catre masina extinsa.

Desi SPIM este un simulator fidel calculatoarelor MIPS, unele lucruri nu sunt identice cu cele ale unui calculator adevarat. Cea mai evidenta diferenta consta în sincronizarea instructiunilor si sistemul de memorie. SPIM nu simuleaza memoria cache sau latenta memoriei, si nici nu reflecta exact întârzierile datorate operatiilor în virgula mobila sau instructiunilor de înmultire si împartire.

O caracteristica pe care o regasim si la masinile reale este aceea ca o pseudoinstructiune expandeaza în mai multe instructiuni masina. Când rulam pas cu pas programul sau examinam memoria, instructiunile pe care le vedem

sunt diferite de cele ale programului sursa. Corespondenta dintre cele doua seturi de instructiuni este simpla întrucât SPIM nu reorganizeaza instructiunile pentru a umple "delay slot-ul" (vezi "*Arhitectura microprocesoarelor MIPS R2000/R3000*").

Referitor la ordinea octetilor dintr-un cuvânt, SPIM opereaza atât cu ordinea "*big-endian*" cât si "*little-endian*". Ordinea octetilor în cadru SPIM este identica cu ordinea de pe masina pe care ruleaza simulatorul. De exemplu, pe o statie DEC 3100, SPIM foloseste "little-endian", în timp ce pe un MacIntosh, HP Bobcat sau Sun SPARC, ordinea folosita este "big-endian".

3. Desfasurarea lucrarii

3.1. Optiuni în linia de comanda

Ambele versiuni de SPIM (*spim* si *xspim*) accepta urmatoarele optiuni în linia de comanda.

- *bare* –
  simuleaza o masina MIPS simpla, fara pseudoinstructiuni sau moduri de adresare suplimentare asigurate de asamblor. Implica *quiet*.
- *asm* –
  simuleaza masina virtuala MIPS furnizata de asamblor. Optiunea *asm* este implicita.
- *notrap* –
  nu se încarca handler-ul de exceptie standard si codul de start-up. Acest handler trateaza exceptiile. Când are loc o exceptie, SPIM executa salt la locatia 80000080H, care contine codul de deservire a exceptiei. Codul de start-up apeleaza rutina *main*. Fara o rutina de start-up, SPIM începe executia de la instructiunea etichetata cu *__start*.
- *trap* –

încarca handler-ul standard de exceptie si codul de start-up. Aceasta optiune este implicita.

- *noquiet –*

  afiseaza un mesaj când apare o exceptie. Este implicita.

- *quiet –*

  nu afiseaza mesaj la exceptie.

- *nomapped_IO –*

  dezactiveaza maparea memoriei.

- *mapped_IO –*

  valideaza facilitatea de mapare a memoriei într-un dispozitiv de intrare/ iesire. Programele care folosesc apeluri sistem SPIM pentru a citi de la un terminal nu pot, de asemenea sa foloseasca maparea memoriei.

- *file –*

  încarca si executa codul scris în asamblare al fisierului.

- *execute –*

  încarca si executa codul în fisiere executabile MIPS *a.out.* Aceasta comanda este disponibila doar când SPIM ruleaza pe un sistem cu procesor MIPS. Programul nu poate invoca nici un serviciu al sistemului de operare (în cazul unei intrari sau iesiri), întrucât SPIM nu simuleaza trap-urile (exceptiile) nucleului MIPS.

- *s seg size –*

  seteaza dimensiunea initiala a segmentului de memorie *seg* la valoarea de *size* octeti. Segmentele de memorie se numesc: **text, data, stack, ktext si kdata.** Segmentul **text** contine instructiunile din program. Segmentul **data** pastreaza datele programului. **Stack** este segmentul de stiva din timpul rularii programului. Pe lânga executia programului, SPIM executa si cod sistem care trateaza întreruperile si exceptiile. Acest cod se afla într-o zona separata a spatiului de adresa numita kernel (nucleu). Segmentul **ktext** pastreaza instructiunile acestui cod

iar **kdata** datele sale. Nu exista **kstack** întrucât codul sistem foloseste aceeasi stiva ca si programul utilizator. De exemplu, linia de comanda urmatoare:

- s data 2000000 - semnifica faptul ca segmentul de date utilizator începe la adresa 0x10010000 si are dimensiunea initiala de 2000000 octeti.

- *l seg size* –

  stabileste limita superioara a segmentului de memorie *seg*, care poate fi de *size* octeti. Segmentele de memorie care pot creste sunt: data, stack si kdata.

3.2. Interfata cu terminalul

Versiunea simpla a SPIM este apelata cu *spim*. Nu este necesar un display mapat pe bit si poate fi rulat de pe orice terminal. Desi *spim* poate fi mai dificil de învatat, el opereaza exact ca *xspim* si asigura aceeasi functionalitate. Interfata cu terminalul se bazeaza pe comenzile:

- *exit* –

  opreste simulatorul.

- *read "file"* –

  citeste fisierul "file" scris în limbaj de asamblare MIPS. Daca fisierul a fost deja citit în SPIM, sistemul trebuie reinitializat sau simbolurile globale vor trebui multiplu definite.

- *load "file"* –

  comanda sinonima cu read.

- *execute "a.out"* –

  citeste executabile MIPS în simulator. Comanda este disponibila numai daca SPIM ruleaza pe un sistem ce contine doar procesor MIPS.

- *run <addr> –*

  porneste rularea programului. Daca argumentul optional <addr> este prezent, programul începe de la aceasta adresa, altfel de la simbolul global __start, care este codul de start implicit.

- *step <N> –*

  ruleaza programul în grupuri de N instructiuni (implicit 1). Afiseaza instructiunile care se executa.

- *continue –*

  continua executia programului fara oprire.

- *print $N –*

  afiseaza continutul registrului N.

- *print $fN –*

  afiseaza continutul registrului flotant N.

- *print addr –*

  afiseaza continutul adresei de memorie addr.

- *print _sym –*

  afiseaza tabela de simboluri, adresa de simboluri globale (nu locale).

- *reinitialize –*

  sterge (anuleaza) continutul registrilor si memoria.

- *breakpoint addr –*

  stabileste un punct de întrerupere (breakpoint) la adresa *addr*. Aceasta poate fi fie o adresa de memorie fie o eticheta.

- *delete addr –*

  sterge toate punctele de întrerupere de la adresa *addr*.

- *list –*

  listeaza toate punctele de întrerupere.

- **.** –

  restul liniei este o instructiune în asamblare care este stocata în memorie.

- <nl> –

linie noua; care determina reexecutarea comenzii anterioare.

- ? –

afiseaza o cerere de help.

Majoritatea comenzilor pot fi abreviate la prefixul: ex, re, l, ru, s, p. Comenzile de genul reinitialize necesita un prefix mai lung.

3.3. Apeluri sistem

SPIM asigura un mic set de servicii ale sistemului de operare prin instructiuni (syscall) – apeluri sistem. Pentru a apela un serviciu, programul încarca codul apelului în registrul v_0 si argumentele în registrele a_0 ... a_3 (sau f_{12} pentru valori în flotant). Apelurile sistem care returneaza valori pun rezultatele în registrele v_0 sau f_0 pentru operatii în flotant.

Serviciul	Codul Apelului Sistem	Argumentele	Rezultatul
Print_int	1	a_0 = integer	
Print_float	2	f_{12} = float	
Print_double	3	f_{12} = double	
Print_string	4	a_0 = string	
Read_int	5		Integer (în v_0)
Read_float	6		Float (în f_0)
Read_double	7		Double (în f_0)
Read_string	8	a_0 = buffer, a_1 = lungimea	
Sbrk	9	a_0 = cantitate	Adresa (în v_0)
Exit	10		

Tabelul 1. Servicii Sistem

Sbrk întoarce un pointer la un bloc de memorie ce contine n octeti, iar *exit* opreste rularea programului.

Programele care folosesc aceste apeluri sistem pentru a citi de la terminal nu trebuie sa foloseasca dispozitive de mapare a memoriei.

Exemplul 1.

Pentru exemplificare vom scrie codul necesar afisarii unui mesaj de tipul:

"solutia corecta = 10".

```
        .data
str:
        .asciiz "solutia corecta ="

        .text
        li      $v0, 4          # codul apelului sistem pentru afisare de string
        la      $a0, str        # adresa sirului de tiparit
        syscall                 # afiseaza sirul

        li      $v0, 1          # codul apelului sistem pentru afisare de întreg
        li      $a0, 10         # valoarea întreaga de tiparit
        syscall                 # afiseaza valoarea
```

În momentul rularii secventei de instructiuni vom observa ca instructiunea *li $v0, 4* – care este de fapt o pseudoinstructiune, este înlocuita de instructiunea *ori $2, $0, 4*; instructiunea de încarcare imediata în registru se obtine dintr-o operatie de SAU între registrul 0, care este întotdeauna 0, si valoarea respectiva.

Adresa etichetei *str*, unde începe sirul de date este 10010000h. Reamintim ca, segmentul de date începe la adresa 10000000h, iar compilatorul MIPS memoreaza variabilele globale în primii 64 Kocteti, deoarece aceste variabile sunt mai frecvent accesate decât alte date globale.

Exemplul 2.

Urmatoarea secventa va afisa pe ecran mesajul: “Dati un sir de caractere: ” si asteapta tastarea unui string de la tastatura.

```
        .data
d3:
        .asciiz "Dati un sir de caractere: "
d4:
        .space 50

        .text
        la $a0, d3          # adresa sirului de tiparit
        li $v0, 4           # codul apelului sistem pentru afisare de string
        syscall             # afiseaza sirul

        la $a0, d4          # adresa unde se va memora sirul
        li $a1, 50          # lungimea maxima a sirului
        li $v0, 8           # codul apelului sistem pentru citire string de la tastatura
        syscall             # se citeste sirul
        done                # se revine din program
```

3.4. Pseudoinstructiuni folosite în limbaj de asamblare MIPS

Prezentam în continuare câteva din pseudoinstructiunile pe care le vom întâlni în lucrarile de laborator ulterioare. Pseudoinstructiunile sunt asemanatoare cu instructiuni din limbajul C, cum ar fi cele de afisare caractere, valori întregi, siruri de caractere, citire de la tastatura a valori de diverse tipuri (**int, char, string**).

Afisare string pe consola:

puts *label* - are ca argument eticheta de la care se va afisa sirul de caractere

Afisare caracter pe consola:

putc *vchar* - are ca argument caracterul care se va afisa pe consola

Afisare numar întreg pe consola:

puti $a0 - are ca argument registrul al carui continut este numarul care se va afisa pe consola

Citire caracter de la tastatura:

getc $a0 - are ca argument registrul în care se va încarca codul ASCII al caracterului citit

Citire întreg de la tastatura:

geti $a0 - are ca argument registrul în care se va încarca valoarea întreaga citita

Terminare program:

done - nu are argumente si determina încheierea programului

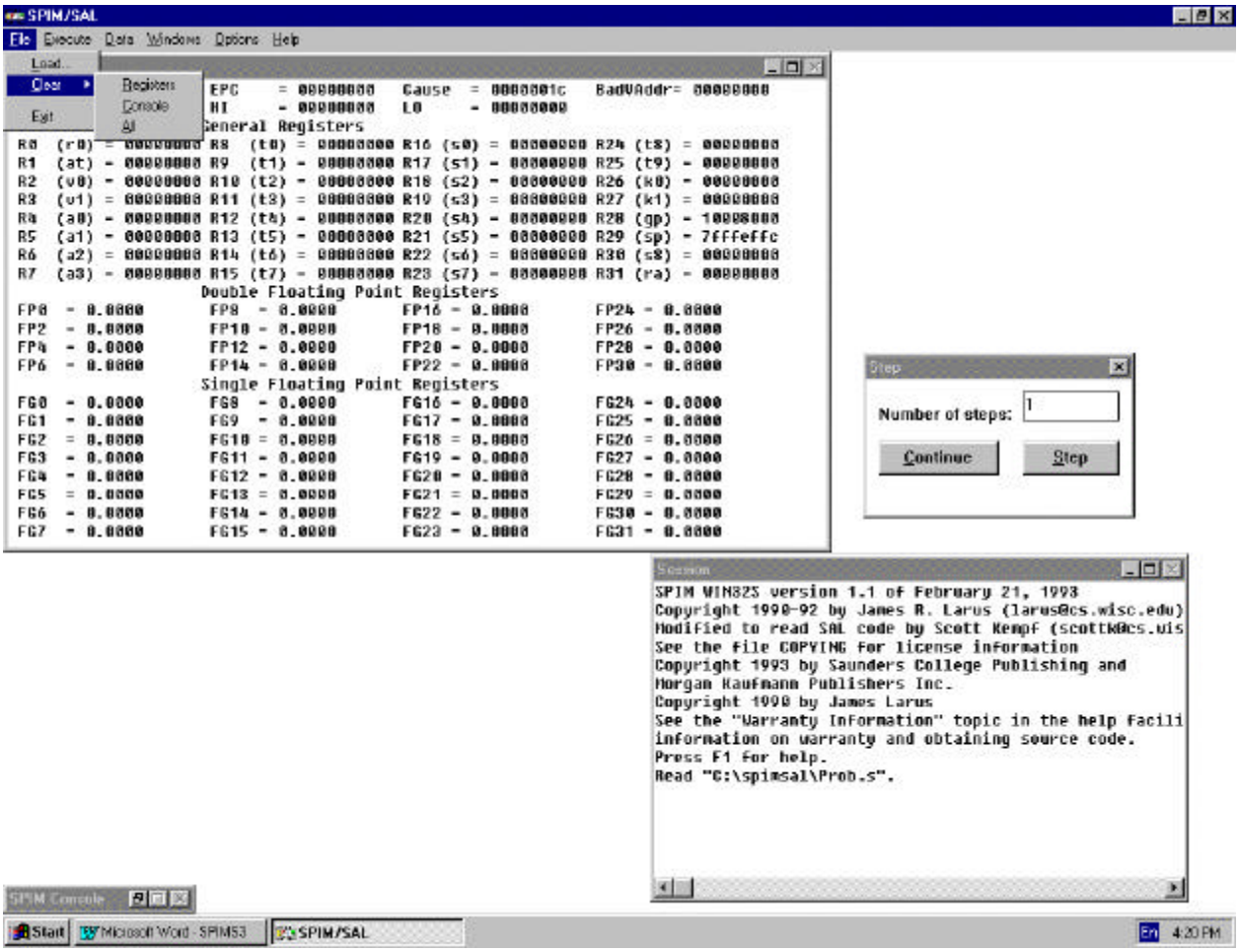
Apel sistem:

syscall - nu are argumente si determina un apel sistem de citire de la tastatura/ afisare pe consola sau terminare program

Celelalte pseudoinstructiuni aritmetico-logice, cu referire la memorie, de salt (conditionat sau nu), de transfer au fost prezentate în lucrarea "*Arhitectura microprocesoarelor MIPS R2000/R3000*".

3.5. Ghid de utilizare al simulatorului

Când pornim simulatorul pe ecran apare fereastra urmatoare.



Pasul urmator ar fi încarcarea unui fisier care urmeaza a fi executat. Se selecteaza optiunea *Load* din meniul *File*, care va declansa aparitia ferestrei de selectie a fisierelor. SPIM/SAL accepta fisiere cu extensia ".s", care sunt fisiere în limbaj de asamblare MIPS. Fisierul selectat este analizat sintactic, iar tipurile de erori depistate sunt scrise în fereastra Sesiune. Încarcarea fisierelor se poate face totodata si din linia de comanda.

Pentru reluarea unui program sau încarcarea unui nou fisier e necesara reinitializarea starilor masinii si a registrilor. Se selecteaza optiunea *Clear* din meniul *File*. Exista trei posibilitati: stergerea ferestrei consola, a registrilor sau a memoriei.

Executia programului poate fi realizata în mod continuu sau pas cu pas. Pentru aceasta a doua varianta, se selecteaza *Step* din meniul *Execute*. Pe ecran apare o cutie de dialog care permite selectia adresei de start si numarul de pasi în care se executa. Cutia de dialog ramâne deschisa pâna la închiderea ei. Concomitent instructiunile sunt reflectate în fereastra Sesiune, atât instructiunile cât si adresa lor.

Managementul operatiilor de breakpoint se face prin selectia optiunii *Breakpoint* din meniul *Execute*. O cutie de dialog va apare pe ecran din care se va selecta operatia si adresa tinta pentru operatia de breakpoint. În cazul selectiei unei adrese la care nu exista instructiuni se genereaza un mesaj de eroare în fereastra Sesiune. Când este întâlnit un punct de întrerupere în executia programului, programul se opreste si apare o cutie de dialog care permite una din optiunile: de continuare sau oprire a programului.

Stabilirea unei date se face alegând optiunea *Set* din meniul *Data*. O cutie de dialog va aparea pe ecran si prin intermediul ei se va specifica adresa unde vrem sa scriem si data pe care o setam la adresa respectiva.

Afisarea unei date în fereastra Sesiune se face prin selectarea optiunii *Print* din meniul *Data*. O cutie de dialog va permite alegerea domeniului de adrese.

Din meniul *Options* se poate alege modul de lucru simplu al asamblorului MIPS - *Bare* si *Quiet* pentru a nu afisa mesaje la exceptie. Prin selectia optiunii *Save* Settings din meniul *Options* se salveaza aceste moduri de functionare ale simulatorului SPIM în fisierul de initializare al Windows-ului - *win.ini*.

SPIM/SAL ofera permanent cinci ferestre: fereastra principala, fereastra Sesiune, fereastra Consola, fereastra Registru si fereastra Memorie. Fiecare are anumite caracteristici:

Fereastra principala este o fereastra simpla care contine un meniu. Ea poate fi redimensionata, minimizata, maximizata si închisa.

Fereastra Sesiune este o fereastra copil care obisnuieste sa afiseze mesaje sistem. Este singura fereastra copil care este initial vizibila. Ea poate fi maximizata, minimizata, redimensionata, ascunsa sau derulata. Exista un buffer

de dimensiune fixa asociat ferestrei sesiune. Cele mai recent scrise mesaje din fereastra sunt pierdute la fel ca si mesajele sosite dupa ce capacitatea buffer-ului s-a atins.

Fereastra Memorie este o fereastra copil care obisnuieste sa afiseze continutul memoriei. Exista o sectiune separata pentru fiecare segment. Ea poate fi maximizata, minimizata, redimensionata, ascunsa si derulata.

Fereastra Registru este o fereastra copil care afiseaza valorile registrelor microprocesorului. Are aceleasi atribute cu fereastra Memorie.

Fereastra Consola este o fereastra de dimensiune fixa care este folosita la receptionarea iesirilor programului si asigura intrarea pentru programe. Poate fi minimizata si ascunsa dar nu maximizata, redimensionata sau derulata. Programele interactioneaza cu fereastra Consola prin apeluri sistem.

Cele patru ferestre (principala, Sesiune, Memorie, Registru) pot coexista pe parcursul rularii unui program. Oricare din acestea poate fi adusa în prim plan, printr-un click pe bara *Caption* a ferestrei respective, celelalte trecând în plan secundar. Acest lucru nu se poate realiza si cu fereastra Consola, ea ramânând în prim plan atât timp cât ea este activa si nu este minimizata.

Bibliografie

[1] **Hennessy J., Patterson D.** - *Computer Organization and Design*: *The Hardware / Software Interface,* Morgan Kaufmann Publishers, San Francisco, California, 1994

[2] **Kane G., Heinrich J.** – *MIPS RISC Architecture*, Prentice Hall, 1992