ARHITECTURA MICROPROCESOARELOR MIPS R2000/R3000

1. Scopul lucrarii

Lucrarea de fata urmareste familiarizarea cu o arhitectura de procesor scalar RISC. Astfel, vom studia modurile de adresare, sintaxa asamblor, setul de instructiuni, partajarea memoriei s.a. la procesorul MIPS R2000 - un exemplu mai recent de masina load/store, acesta putând fi considerat reprezentantul unei masini RISC complete.

2. Memento teoretic

Obiectivul procesorului MIPS este înalta performanta obtinuta prin pipelining, o implementare hardware usoara si compatibilitatea cu compilatoare optimizate. Aceste scopuri conduc la instructiuni simple, moduri simple de adresare, formate de instructiuni de lungime fixa si multi registrii.

Arhitectura calculatoarelor MIPS este simpla si obisnuita, putând fi usor înteleasa si învatata. Procesorul contine 32 registrii de uz general si un set de instructiuni bine proiectat care-l face o tinta favorabila pentru generarea codului într-un compilator. Pe lânga unitatea de procesare pentru întregi, procesorul MIPS contine si o colectie de coprocesoare care îndeplinesc sarcini auxiliare sau opereaza asupra altor tipuri de date, cum ar fi numere în flotant.

Arhitectura MIPS, ca majoritatea calculatoarelor RISC, este dificil de programat direct datorita întârzierilor introduse de branch-uri (instructiuni de ramificatie) si load-uri (instructiuni cu referire la memorie) precum si datorita restrictiilor introduse de modurile de adresare. Impedimentul este acceptabil întrucât aceste calculatoare sunt destinate a fi programate în limbaje de nivel înalt, si astfel, ele prezinta o interfata potrivita pentru programatorii de compilatoare

mai degraba decât pentru cei ce scriu în limbaje de asamblare. Pentru a atinge o performanta ridicata, compilatoarele MIPS trebuie sa foloseasca registrii în mod eficient.

O buna parte a complexitatii programelor rezulta din instructiunile care implica întârzieri. Un branch necesita doi cicli de întârziere, necesari stabilirii conditiei de salt si calculului de adresa tinta. În al doilea ciclu, instructiunea imediat urmatoare branch-ului se executa. Aceasta instructiune poate fi una utila care s-ar executa în mod normal înainte de branch sau poate fi un simplu *nop* (no operation). Similar, un load întârziat necesita doi cicli, astfel ca instructiunea imediat urmatoare load-ului nu va putea folosi valoarea din memorie.

MIPS ascunde complexitatea amintita mai sus, având implementat propriul asamblor (o masina virtuala). Acest calculator virtual nu are load-uri si branch-uri întârziate, în plus, are un set mai bogat de instructiuni decât hardware-ul actual. Asamblorul rearanjeaza instructiunile pentru a umple "delay slot-ul" - întârzierea pe care în mod normal ar fi introdus-o instructiunile întârziate (branch si load). De asemenea, sunt simulate macroinstructiuni sau *pseudoinstructiuni*, prin generarea unor secvente scurte de instructiuni actuale (de baza).

Proiectantii MIPS au furnizat doua moduri de acces al operanzilor. Primul este de a face mai rapid accesul la constante mici si al doilea de a face branch-urile mai eficiente. Compromisul realizat de proiectanti este de a pastra toate instructiunile de aceeasi lungime, necesitând diferite tipuri de formate de instructiuni pentru instructiuni diverse.

3. Desfasurarea lucrarii

3.1. Schema bloc si registrii procesorului MIPS R2000

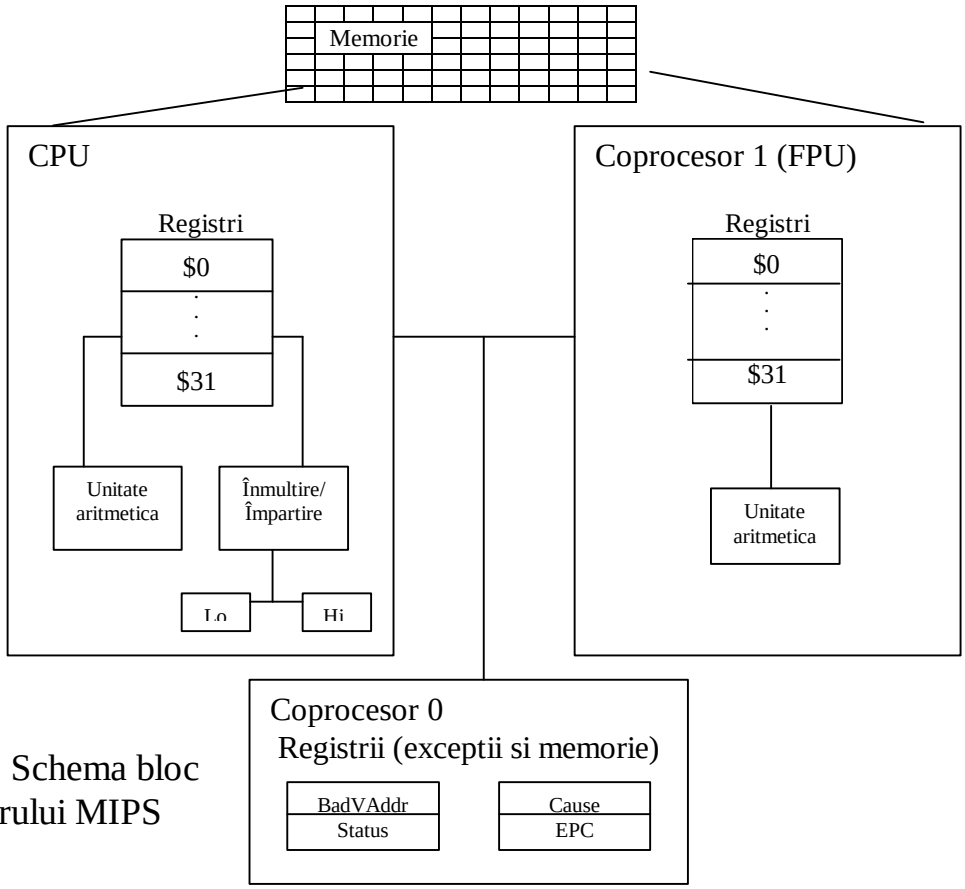


Figura 1. Schema bloc a procesorului MIPS R2000

Unitatea centrala de procesare a MIPS contine 32 registrii generali numerotati de la 0 la 31. Registrul n este desemnat ca $n. Registrul $0 este întotdeauna cablat la valoarea 0. MIPS a stabilit un set de conventii despre cum ar trebui folositi registrii. Aceste conventii sunt principii calauzitoare, care nu sunt fortate de catre hardware. Totusi, un program care violeaza aceste principii nu va functiona perfect cu alt software.

Numele Registrilor	Numarul Registrilor	Utilizarea Registrilor
Zero	0	Constanta 0.
at, k_0, k_1	1, 26, 27	Rezervati sistemului de operare; nu trebuie folositi de catre programele utilizator sau compilatoare.

v_0, v_1	2, 3	Pastreaza rezultatele returnate de functii în urma apelurilor sistem sau (în v_0) - codul apelurilor sistem.
$a_0 \div a_3$	4 ÷ 7	Transmiterea primelor patru argumente rutinei apelate (restul argumentelor sunt puse pe stiva).
$t_0 \div t_9$	8 ÷ 15, 24, 25	Registrii de salvare ai rutinei apelante, memoreaza temporar valori care nu trebuie pastrate de-a lungul apelului.
$s_0 \div s_9$	16 ÷ 23	Registrii de salvare ai rutinei apelate, memoreaza valorile ce trebuie pastrate la iesirea din proceduri.
Gp	28	Pointer global care indica spre mijlocul unui bloc de 64K octeti în segmentul de date statice, ce pastreaza constante si variabile globale.
sp	29	Indicator de stiva, care pointeaza pe prima locatie libera din stiva.
fp	30	Pointer de cadru
ra	31	Pastreaza adresa de revenire dintr-o procedura dupa apelul instructiunii jal.

Tabelul 1. Registrii MIPS si conventii de utilizare

În plus, coprocesorul 0 contine registrii care sunt folositi pentru tratarea exceptiilor, pe care îi prezentam în tabelul 2.

Numele Registrului	**Numãr**	**Utilizare**
BadVAddr	8	Adresa de memorie la care apare adresa virtuala care a produs exceptia
Status	12	Contine bitii de validare a întreruperii

Cause	13	Tipul exceptiei
EPC	14	Adresa instructiunii care a cauzat exceptia

Tabelul 2. Registrii coprocesorului 0

Acesti registrii sunt o parte din setul de registrii ai coprocesorului 0 si pot fi accesati prin instructiuni de genul: *lcw0, mfc0, mtc0* si *swc0*.

3.2. Ordinea octetilor si modurile de adresare

Procesoarele pot numerota octetii din interiorul unui cuvânt de 32 de biti astfel încât octetul cu numarul cel mai mic este fie cel mai din stânga fie cel mai din dreapta. Conventia folosita de catre o masina se numeste *ordinea octetilor*. Procesoarele MIPS pot opera fie cu ordinea octetilor *big-endian*

Byte #			
0	1	2	3

sau

little-endian

Byte #			
3	2	1	0

Instructiunile de calcul opereaza doar cu valorile din registre. O masina simpla asigura doar un singur mod de adresare a memoriei, si anume modul indexat: **c(rx)**, care foloseste suma imediata dintre constanta **c** si registrul **rx** ca o adresa. Masinile virtuale asigura urmatoarele moduri de adresare pentru instructiunile load/store (vezi tabelul 2).

FORMAT	CALCULUL ADRESEI
(Registru)	Continutul unui registru
Imm	Valoare imediata
Imm(registru)	valoare imediata + continutul registrului
Symbol	Adresa simbolului
Symbol imm	adresa simbolului + sau – valoarea imediata
Symbol imm(registru)	adresa simbolului + sau - (valoarea imediata + continutul registrului)

Tabelul 3. Modurile de Adresare la MIPS R2000/R3000

Majoritatea instructiunilor load si store opereaza doar cu date aliniate. O cantitate este aliniata daca adresa sa de memorie este un multiplu a marimii sale în octeti. De aceea, un obiect de jumatate de cuvânt trebuie sa fie memorat la adrese pare iar obiecte pe un cuvânt trebuie memorate la adrese care sunt multiplu de 4. Totusi, MIPS asigura unele instructiuni pentru manipularea datelor nealiniate (*lwl*, *lwr*, *swl* si *swr*).

3.3. Sintaxa asamblor

Limbajul de asamblare este un limbaj de programare, principala sa deosebire fata de limbajele de nivel înalt, cum sunt BASIC, PASCAL si C, fiind aceea ca el ofera doar câteva tipuri simple de comenzi si date. Limbajele de asamblare nu specifica tipul valorilor pastrate în variabile, lasând programatorul sa le aplice operatiile potrivite.

Comentariile în fisiere de asamblare încep cu simbolul #. Orice urmeaza acestui caracter pâna la sfârsitul liniei este ignorat. Identificatorii sunt o secventa de caractere alfanumerice, linie de subliniere (underbars), si punct, si sa nu înceapa cu un numar. Opcode-ul instructiunilor sunt cuvinte rezervate care nu pot

fi folosite ca identificatori. Etichetele sunt declarate prin asezarea lor la începutul unei linii urmate de caracterul ‘:’.

Exemplu:

```
        .data
item:   .word 1
        .text
        .globl main         # Must be global
main:   lw $t0, item
```

Numerele sunt implicit în baza 10. Daca sunt precedate de caracterul 0x, ele sunt interpretate ca hexazecimal. Deci, 256 si 0x100 semnifica aceeasi valoare.

Sirurile sunt încadrate de ghilimele ‘”’. Caracterele speciale din siruri urmeaza conventia limbajului C. Astfel:

- linie noua \n
- tab \t
- ghilimele \”

Prezentam, în continuare, câteva din directivele de asamblare ale MIPS.

.align n - alineaza datele urmatoare într-un domeniu de 2^n octeti. De exemplu,

.align 2 alineaza valorile urmatoare într-un domeniu limitat de 1 cuvânt (word).

.align 0 dezactiveaza automat alinierea datorata directivelor *.half*, *.word*, *.float* si *.double* pâna la urmatoarea directiva *.data* sau *.kdata*.

.ascii str - memoreaza sirul *str* în memorie, dar nu si terminatorul null.

.asciiz str - memoreaza sirul *str* în memorie împreuna cu terminatorul null.

.byte b_1, ..., b_n - memoreaza n valori în octeti succesivi în memorie.

.data <addr> - numere (articole) ulterioare sunt memorate în segmentul de date. Daca argumentul optional *addr* este prezent, valorile ulterioare sunt memorate începând de la adresa *addr*.

.double d_1, ..., d_n - memoreaza cele n numere în flotant dubla precizie în locatii succesive de memorie.

.extern sym size - declara data memorata la *sym* de dimensiunea *size* si este un simbol global. Aceasta directiva valideaza asamblorul sa memoreze datele într-o fractiune a segmentului de date care este accesata eficient prin registrul $gp.

.float f_1, ..., f_n - memoreaza cele n numere în flotant simpla precizie în locatii succesive de memorie.

.globl sym - declara simbolul *sym* ca global si poate fi referit din alte fisiere.

.half h_1, ...,h_n - memoreaza n locatii de 16 biti în semicuvinte succesive în memorie.

.kdata <addr> - date succesive sunt memorate în segmentul de date al kernel-ului. Daca argumentul optional *addr* este prezent, datele ulterioare sunt memorate începând cu adresa addr.

.ktext <addr> - date succesive sunt memorate în segmentul de text al kernel-ului. Daca argumentul optional *addr* este prezent, datele ulterioare sunt memorate începând cu adresa addr.

.set noat .set at - a doua directiva revalideaza avertismentele. Întrucât instructiunile false expandeaza în cod care foloseste registrul $1, programatorii trebuie sa fie foarte atenti când parasesc valorile din registru.

.space n - aloca n octeti de spatiu în segmentul curent.

.text <addr> - valori ulterioare sunt puse în segmentul de text. Daca argumentul optional *addr* este prezent, atunci valorile sunt memorate începând cu aceasta adresa.

.word w_1, ..., w_n - memoreaza cuvinte de 32 de biti în locatii succesive de memorie.

3.4. Formatul instructiunilor si setul de instructiuni al procesorului MIPS R2000

Prezentam în continuare, atât instructiuni implementate în hardware-ul real al MIPS cât si pseudoinstructiuni oferite de asamblorul MIPS. Cele doua tipuri de instructiuni se disting usor. Instructiunile reale sunt descrise de câmpuri reprezentate binar. De exemplu:

add R_d, R_s, R_t

0	R_s	R_t	R_d	0	0x20
6	5	5	5	5	6

adunare cu depasire (overflow)

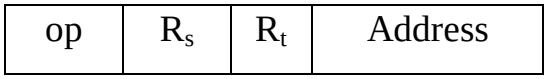
Instructiunea *add* consta din 6 câmpuri. Dimensiunea fiecarui câmp în biti este indicata de numerele de sub câmp. Aceasta instructiune începe cu 6 biti de 0. Urmeaza codificarea registrului sursa pe 5 biti, registru tampon si registrul destinatie tot pe 5 biti. Un alt câmp general este Imm_{16}, care este o constanta imediata pe 16 biti. Formatul de mai sus se numeste R-tip si este o particularizare a formatului general urmator:

op	R_s	R_t	R_d	shamt	funct
6	5	5	5	5	6

Câmpurile au urmatoarea semnificatie:

- *Op* – operatia instructiunii
- R_s – primul registru sursa, operand al instructiunii
- R_t – al doilea registru sursa
- R_d – registrul destinatie; pastreaza rezultatul operatiei
- *Shamt* – cantitate de shiftare
- *Func* - functia prin care se selecteaza varianta operatiei din câmpul *op*.

Un al doilea tip de format de instructiuni este I-tip si este folosit de catre instructiunile de transfer. Câmpurile formatului sunt urmatoarele:



Desi, multiplele formate complica hardware-ul, prin pastrarea unor formate similare se poate reduce aceasta complexitate. De exemplu, primele trei câmpuri ale formatelor prezentate mai sus (R-tip si I-tip) sunt identice, iar dimensiunea celui de-al patru-lea câmp al formatului I-tip este egal cu suma ultimelor trei câmpuri ale R-tip. Formatele se disting prin valorile din primul câmp.

Un al treilea format pentru instructiunile MIPS se numeste J-tip, care consta dintr-un câmp operatie pe 6 biti si unul de adresa de 26 de biti.

Pseudoinstructiunile urmeaza aproximativ aceeasi conventie, dar omit informatia de codificare a instructiunilor. Astfel, S_{rc2} este fie registru fie constanta imediata. Asamblorul va transforma forma generala a unei instructiuni (Ex. *add*) în una intermediara (*addi*) daca cel de-al doilea parametru este o constanta imediata.

3.4.1. Instructiuni aritmetice si logice

abs R_{dest}, R_{src} Valoare absoluta

Pune valoarea absoluta a întregului aflat în registrul sursa în registrul destinatie.

add $\mathbf{R_{dest}}$, $\mathbf{R_{src1}}$, $\mathbf{S_{rc2}}$ Adunare cu depasire (overflow)

addi **R_{dest}, R_{src1}, Imm**	Adunare imediata cu depasire
addu **R_{dest}, R_{src1}, S_{rc2}**	Adunare fara depasire
addiu **R_{dest}, R_{src1}, Imm**	Adunare imediata fara depasire

Calculeaza suma întregilor din registrii R_{src1} si S_{rc2} sau Imm, si depune rezultatul în R_{dest}.

and **R_{dest}, R_{src1}, S_{rc2}**	SI logic
andi **R_{dest}, R_{src1}, Imm**	SI logic cu o constanta imediata

Pune în registrul R_{dest} rezultatul obtinut în urma operatiei SI logic dintre întregii din registrii R_{src1} si S_{rc2}.

div **R_{src1}, RS_{rc2}**	Împartire cu depasire
divu **R_{src1}, RS_{rc2}**	Împartire fara depasire

Împarte continutul celor doua registre. Câtul este depus în registrul LO si restul în registrul HI. De remarcat ca, daca un operand este negativ, restul este nespecificat într-o arhitectura MIPS si depinde de conventia masinii pe care este rulat programul.

div R_{dest}, R_{src1},S_{rc2}	Împartire cu depasire
divu R_{dest}, R_{src1},S_{rc2}	Împartire fara depasire

Pune câtul împartirii întregi dintre R_{src1} si S_{rc2} în registrul destinatie R_{dest}.

mul R_{dest}, R_{src1},S_{rc2}	Înmultire fara depasire
mulo R_{dest}, R_{src1},S_{rc2}	Înmultire cu depasire
mulou R_{dest}, R_{src1},S_{rc2}	Înmultire fara semn cu depasire

Pune în registrul R_{dest} rezultatul obtinut în urma înmultirii dintre întregii din registrii R_{src1} si S_{rc2}.

***mult* $\mathbf{R_{src1}}$, $\mathbf{R_{src2}}$** Înmultire

***multu* $\mathbf{R_{src1}}$, $\mathbf{R_{src2}}$** Înmultire fara semn

Înmulteste continutul celor doua registre. Partea mai putin semnificativa a rezultatului e depusa în registrul LO, iar cea semnificativa în registrul HI.

neg R_{dest}, R_{source} Valoarea negativa cu depasire

negu R_{dest}, R_{source} Valoarea negativa fara depasire

Valoarea negativa a întregului din registrul R_{source} este depusa în registrul R_{dest}.

nor R_{dest}, R_{src1}, S_{rc2} NOR

Rezultatul operatiei NOR logic a întregilor din registrele R_{src1} si S_{rc2}, este depus în registrul R_{dest}.

***not* $\mathbf{R_{dest}}$, $\mathbf{R_{source}}$** NOT

Pune negatia logica a întregului din registrul R_{source} în registrul R_{dest}.

***or* $\mathbf{R_{dest}}$, $\mathbf{R_{src1}}$, $\mathbf{S_{rc2}}$** OR

***ori* $\mathbf{R_{dest}}$, $\mathbf{R_{src1}}$, Imm** OR imediat

Rezultatul operatiei SAU logic a întregilor din registrele R_{src1} si S_{rc2} (sau Imm) este depus în registrul R_{dest}.

rem R_{dest}, R_{src1}, S_{rc2}	Rest
remu R_{dest}, R_{src1}, S_{rc2}	Rest fara semn

Pune restul împartirii dintre valoarea întreaga aflata în registrul R_{src1} la cea din S_{rc2}, în registrul R_{dest}. Daca un operand este negativ, restul este nespecificat pentru o arhitectura MIPS si depinde de conventiile masinii pe care ruleaza programul.

rol R_{dest}, R_{src1}, S_{rc2}	Roteste la stânga
ror R_{dest}, R_{src1}, S_{rc2}	Roteste la dreapta

Roteste la stânga sau dreapta continutul registrului R_{src1}, cu distanta indicata de registrul S_{rc2} si pune rezultatul în registrul destinatie.

sll R_{dest}, R_{src1}, S_{rc2}	Deplaseaza logic la stânga
sllv R_{dest}, R_{src1}, S_{rc2}	Deplaseaza logic la stânga variabil
sra R_{dest}, R_{src1}, S_{rc2}	Deplaseaza aritmetic la dreapta
srav R_{dest}, R_{src1}, S_{rc2}	Deplaseaza aritmetic la dreapta variabil
srl R_{dest}, R_{src1}, S_{rc2}	Deplaseaza logic la dreapta
srlv R_{dest}, R_{src1}, S_{rc2}	Deplaseaza logic la dreapta variabil

Deplaseaza la stânga sau dreapta continutul registrului R_{src1}, cu distanta indicata de registrul S_{rc2} si pune rezultatul în registrul destinatie.

sub $\mathbf{R_{dest}}$, $\mathbf{R_{src1}}$, $\mathbf{S_{rc2}}$	Scadere cu depasire
subu $\mathbf{R_{dest}}$, $\mathbf{R_{src1}}$, $\mathbf{S_{rc2}}$	Scadere fara depasire

Diferenta dintre întregii din registrele R_{src1} si S_{rc2} este depusa în registrul destinatie.

xor R_{dest}, R_{src1}, S_{rc2}	XOR
xori R_{dest}, R_{src1}, Imm	XOR imediat

Calculeaza XOR logic dintre întregii din registrii R_{src1} si S_{rc2}(sau Imm), rezultatul fiind depus în R_{dest}.

3.4.2. Instructiuni cu referire la memorie si manipulare a constantelor

***la* R_{dest}, address**	Încarcare adresa

Încarca adresa calculata, valoarea adresei nu continutul locatiei, în registrul R_{dest}.

lb R_{dest}, address	Încarcare octet
lbu R_{dest}, address	Încarcare octet fara semn

Încarca octetul de la adresa *address* în registrul destinatie (si extensia de semn).

ld R_{dest}, address	Încarcare dublu cuvânt

Încarca 8 octeti de la adresa *address* în registrii R_{dest} si R_{dest}+1.

lh R_{dest}, address	Încarcare semicuvânt
lhu R_{dest}, address	Încarcare semicuvânt fara semn

Încarca 2 octeti (un semicuvânt de 32 biti) de la adresa *address* în registrul destinatie precum si extensia de semn.

***lw* $\mathbf{R_{dest}}$, address** Încarcare cuvânt

Încarca un cuvânt de 4 octeti de la adresa *address* în registrul R_{dest}.

lwcz R_{dest}, address Încarcare cuvânt în coprocesor

Încarca un cuvânt de 4 octeti de la adresa *address* în unul din registrii coprocesorului z(0-3).

lwl R_{dest}, address Încarcare din stânga a cuvântului
lwr R_{dest}, address Încarcare din dreapta a cuvântului

Încarca în registrul destinatie octetii din stânga sau dreapta ai unui cuvânt în cazul unei adrese posibil nealiniate.

sb R_{source}, address Memorare octet

Memoreaza octetul mai putin semnificativ al registrului sursa la adresa specificata de *address*.

sd R_{source}, address Memorare dublu cuvânt

Memoreaza 8 octeti (continutul registrilor R_{source} si R_{source} +1) la adresa indicata de *address*.

sh R_{source}, address Memorare semicuvânt

Memoreaza primii doi octeti mai putin semnificativi al registrului sursa la adresa specificata de *address*.

***sw* $\mathbf{R_{source}}$, address**	Memorare cuvânt

Memoreaza cuvântul din registrul sursa la adresa *address*.

swcz R_{source}, address	Memorare cuvânt din coprocesor

Memoreaza cuvântul din R_{source} al coprocesorului z la adresa *address*.

swl R_{source}, address	Memorare portiune stânga din cuvânt
swr R_{source}, address	Memorare portiune dreapta din cuvânt

Memoreaza octetii din stânga sau dreapta ai registrului sursa în cazul unei posibile nealinieri a adresei.

ulh R_{dest}, address	Încarcare semicuvânt de la adresa nealiniata
ulhu R_{dest}, address	Încarcare semicuvânt fara semn de la adresa nealiniata

Încarcare semicuvânt si extensie semn de la adrese posibil nealiniate în registrul destinatie.

ulw R_{dest}, address	Încarcare cuvânt de la adresa nealiniata

Încarcare cuvânt de la adrese posibil nealiniate în registrul destinatie.

ush R_{source}, address — Memorare semicuvânt la adresa nealiniata

Memoreaza semicuvântul mai putin semnificativ din registrul sursa la o adresa posibil nealiniata.

usw R_{source}, address — Memorare cuvânt la adresa nealiniata

Memoreaza cuvântul din registrul sursa la o adresa posibil nealiniata.

li **R_{dest}, Imm** — Încarcare imediata

Transfera constanta imediata în registrul destinatie.

li.d FR_{dest}, float — Încarcare constanta imediata dubla precizie

Transfera un numar în virgula mobila dubla precizie în registrii flotanti FR_{dest} si FR_{dest} +1.

li.s FR_{dest}, float — Încarcare constanta imediata simpla precizie

Transfera un numar în flotant simpla precizie în registrul flotant $FR_{dest.}$

lui **R_{dest}, integer** — Încarcare constanta întreaga

Încarca semicuvântul mai putin semnificativ al unui întreg în semicuvântul semnificativ al registrului destinatie. Cei mai putin semnificativi biti ai registrului sunt 0.

3.4.3. Instructiuni de întreruperi si exceptii

rfe Revenire din exceptie

Reface registrul de stare al programului.

syscall Apel sistem

Registrul $\$v_0$ contine numarul apelului.

break n Exceptia n

Cauzeaza exceptia n. Exceptia 1 este rezervata debugger-ului.

nop Nici o operatie

Nu executa nimic.

3.4.4. Instructiuni de comparatie

În toate instructiunile de mai jos, S_{rc2} poate fi fie registru fie valoare imediata.

seq R_{dest}, R_{src1}, S_{rc2} Seteaza flag în caz de egalitate

sge R_{dest}, R_{src1}, S_{rc2} Seteaza flag pe relatia de mai mare

sgeu R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mare fara semn
sgt R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mare strict
sgtu R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mare strict fara semn
sle R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mic sau egal
sleu R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mic sau egal fara semn
slt R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mic strict
slti R_{dest}, R_{src1}, Imm	Seteaza flag pe relatia de mai mic strict decât valoarea imediata
sltu R_{dest}, R_{src1}, S_{rc2}	Seteaza flag pe relatia de mai mic decât valoare fara semn
sltiu R_{dest}, R_{src1}, Imm	Seteaza flag pe relatia de mai mic decât valoare imediata fara semn
sne R_{dest}, R_{src1}, S_{rc2}	Seteaza flag în caz de inegalitate

Seteaza registrul destinatie la valoarea 1 daca operanzii R_{src1} si S_{rc2} (sau Imm) sunt în relatia specificata si 0 altfel. Comparatiile pot fi cu sau fara semn.

3.4.5. Instructiuni de salt si ramificatie

În toate instructiunile de mai jos, S_{rc2} poate fi fie registru fie valoare imediata (întreg). Instructiunile de salt folosesc un câmp offset pe 16 biti cu semn. Deci pot exista salturi de 2^{15}-1instructiuni înainte si 2^{15} înapoi. Instructiunea *jump* contine un câmp de adresa de 26 de biti. Offset-ul instructiunilor nu este precizat,

el fiind înlocuit cu eticheta. Acest lucru se întâmpla în majoritatea limbajelor de asamblare deoarece distanta dintre instructiuni este dificil de calculat când pseudoinstructiunile expandeaza în câteva instructiuni reale.

b label	Instructiunea branch

Se executa salt neconditionat la eticheta *label*.

bczt label	Branch pe conditie adevarata în coprocesor
bczf label	Branch pe conditie falsa în coprocesor

Se executa salt conditionat la eticheta *label* daca conditia din registrul coprocesorului este adevarata (sau falsa).

beq R_{src1}, S_{rc2}, label	Branch pe egalitate
bge R_{src1}, S_{rc2}, label	Branch pe conditie de mai mare sau egal
bgeu R_{src1}, S_{rc2}, label	Branch pe conditie de mai mare sau egal, comparatie fara semn
bgt R_{src1}, S_{rc2}, label	Branch pe conditie de mai mare strict
bgtu R_{src1}, S_{rc2}, label	Branch pe conditie de mai mare strict, comparatie fara semn
ble R_{src1}, S_{rc2}, label	Branch pe conditie de mai mic sau egal
bleu R_{src1}, S_{rc2}, label	Branch pe conditie de mai mic sau egal, comparatie fara semn
blt R_{src1}, S_{rc2}, label	Branch pe conditie de mai mic strict

bltu R_{src1}, S_{rc2}, label	Branch pe conditie de mai mic, comparatie fara semn
bne R_{src1}, S_{rc2}, label	Branch pe neegalitate

Salt conditionat la eticheta *label* daca continuturile celor doua registre sunt în relatia respectiva. Comparatia poate fi cu semn sau fara semn.

beqz **R_{source}, label**	Branch pe egalitate cu zero
bgez R_{source}, label	Branch pe conditie de mai mare sau egal cu zero
bgezal R_{source}, label	Branch pe conditie de mai mare sau egal cu zero si salvare legatura
bltzal R_{source}, label	Branch pe conditie de mai mic strict decât zero si salvare legatura
bgtz R_{source}, label	Branch pe conditie de mai mare strict decât zero
blez **R_{source}, label**	Branch pe mai mic sau egal decât zero
bltz R_{source}, label	Branch pe conditie de mai mic strict decât zero
bnez R_{source}, label	Branch pe neegalitate cu zero

Salt conditionat la eticheta *label* daca continutul registrului sursa îndeplineste conditiile respective. În plus unele instructiuni salveaza adresa urmatoarei instructiuni în registrul 31.

j **label**	Jump

Salt neconditionat la eticheta.

***jal* label** Jump si salvare legatura

***jalr* R_{source}** Jump si salvare legatura prin registru

Salt neconditionat la eticheta sau la adresa specificata de registrul sursa. Adresa urmatoarei instructiuni e salvata în registrul 31. Deci un CALL/RET se transforma în JAL/JR $31.

***jr* R_{source}** Jump la adresa din registru

Salt neconditionat la adresa specificata de registrul sursa.

3.4.6. Instructiuni de transfer

***move* R_{dest}, R_{source}** Transfer

Transfera continutul registrului sursa în registrul destinatie.

Rezultatul obtinut în urma executiei instructiunilor de înmultire si împartire este depus în doi registrii suplimentari, HI si LO. Aceste instructiuni transfera valori în si din registrii. Instructiunile de înmultire, împartire si rest, descrise anterior (vezi 3.4.1.), sunt pseudoinstructiuni care fac sa apara ca si cum unitatile de executie respective opereaza cu registrii generali si detecteaza conditii de eroare cum ar fi împartire cu 0 sau depasire de domeniu.

***mfhi* R_{dest}** Transfera din HI

***mflo* R_{dest}** Transfera din LO

Muta continutul registrului HI sau LO în registrul destinatie.

mthi $\mathbf{R_{dest}}$	Transfera în HI
mtlo $\mathbf{R_{dest}}$	Transfera în LO

Muta continutul registrului HI sau LO în registrul destinatie.

Fiecare coprocesor are propriul sau set de registre. Urmatoarele instructiuni transfera valori între aceste registre si registrele procesorului.

mfcz R_{dest}, $C_{opsource}$	Transfer din coprocesor z

Muta continutul registrului $C_{opsource}$ al coprocesorului z în registrul destinatie R_{dest}.

mfc1.d R_{dest}, FR_{src1}	Transfer dublu cuvânt flotant din coprocesorul 1

Transfera continutul registrelor flotante FR_{src1} si $FR_{src1}+1$ în registrele procesorului R_{dest} si $R_{dest}+1$.

mtcz R_{source}, C_{opdest}	Transfer în coprocesor z

Muta continutul registrului R_{source} al procesorului în registrul C_{opdest} al coprocesorului z.

Nota:

Instructiunile scrise cu litere aldine se vor întâlni mai des în practica, mai ales în lucrarile de laborator "*Utilizarea simulatorului SPIM*" si "*Investigatii arhitecturale utilizând simulatorul SPIM*".

3.4.7. Instructiuni în virgula mobila

Calculatorul MIPS are un coprocesor (notat cu 1) care opereaza în virgula mobila simpla precizie (valori pe 32 biti) si dubla precizie (valori pe 64 biti). Acest coprocesor are proprii sai registrii, care sunt numerotati $\$f_0$ - $\$f_{31}$. Deoarece aceste registre sunt doar pe 32 de biti, sunt necesare doua din ele pentru a pastra rezultatul în dubla precizie. Pentru a simplifica lucrurile, operatiile în virgula mobila folosesc doar registre numerotate par - chiar si instructiunile care opereaza în simpla precizie. Valorile sunt transferate în si din aceste registre pe cuvinte de 32 biti prin instructiunile: *lwc1*, *swc1*, *mtc1* si *mfc1* (descrise anterior) si *l.s*, *l.d*, *s.s* si *s.d* pseudoinstructiuni descrise în continuare. Flagul setat de catre operatiile de comparatie este citit de procesor prin instructiunile *bc1t* si *bc1f*. În toate instructiunile de mai jos FR_{dest}, FR_{src1}, FR_{src2} si FR_{source} sunt registrii flotanti.

abs.d FR_{dest}, FR_{source}	Valoare absoluta registru flotant dubla precizie
abs.s FR_{dest}, FR_{source}	Valoare absoluta registru flotant simpla precizie

Calculeaza valoarea absoluta a registrului sursa flotant simpla (sau dubla) precizie si pune rezultatul în registrul destinatie.

add.d FR_{dest}, FR_{src1}, FR_{src2}	Adunare în dubla precizie
add.s FR_{dest}, FR_{src1}, FR_{src2}	Adunare în simpla precizie

Calculeaza suma celor doi registrii sursa în simpla sau dubla precizie si pune rezultatul în registrul destinatie.

c.eq.d FR_{src1}, FR_{src2}	Comparatie de egalitate în dubla precizie
c.eq.s FR_{src1}, FR_{src2}	Comparatie de egalitate în simpla precizie

Se seteaza flagul conditie în flotant pe true daca continutul registrilor sursa este identic.

c.le.d FR_{src1}, FR_{src2}	Comparatie pe relatie de mai mic sau egal în dubla precizie
c.le.s FR_{src1}, FR_{src2}	Comparatie pe relatie de mai mic sau egal în simpla precizie

Se seteaza flagul conditie în flotant pe true daca FR_{src1} este mai mic sau egal decât FR_{src2}.

c.lt.d FR_{src1}, FR_{src2}	Comparatie pe relatie de strict mai mic în dubla precizie
c.lt.s FR_{src1}, FR_{src2}	Comparatie pe relatie de strict mai mic în simpla precizie

Se seteaza flagul conditie în flotant pe true daca FR_{src1} este mai mic strict decât FR_{src2}.

cvt.d.s FR_{dest}, FR_{source}	Conversie din simpla în dubla precizie
cvt.d.w FR_{dest}, FR_{source}	Conversie din întreg în dubla precizie

Converteste un numar în simpla precizie (sau un întreg) aflat în registrul FR_{source} în dubla precizie si pune rezultatul în registrul FR_{dest}.

cvt.s.d FR_{dest}, FR_{source}	Conversie din dubla în simpla precizie
cvt.s.w FR_{dest}, FR_{source}	Conversie din întreg în simpla precizie

Converteste un numar în dubla precizie (sau un întreg) aflat în registrul FR_{source} în simpla precizie si pune rezultatul în registrul FR_{dest}.

cvt.w.d FR_{dest}, FR_{source}	Conversie din dubla precizie în întreg
cvt.w.s FR_{dest}, FR_{source}	Conversie din simpla precizie în întreg

Converteste un numar în dubla (sau simpla) precizie aflat în registrul FR_{source} în întreg si pune rezultatul în registrul FR_{dest}.

div.d FR_{dest}, FR_{src1}, FR_{src2}	Împartire în dubla precizie
div.s FR_{dest}, FR_{src1}, FR_{src2}	Împartire în simpla precizie

Calculeaza câtul împartirii celor doi registrii FR_{src1} si FR_{src2} si pune rezultatul în registrul FR_{dest}.

l.d FR_{dest}, address	Încarcare valoare dubla precizie de la adresa
l.s FR_{dest}, address	Încarcare valoare simpla precizie de la adresa

Încarca o valoare în dubla (sau simpla) precizie de la adresa specificata.

mov.d FR_{dest}, FR_{source}	Transfera valoare dubla precizie
mov.s FR_{dest}, FR_{source}	Transfera valoare simpla precizie

Transfera o valoare dubla (sau simpla) precizie din registrul sursa în cel destinatie.

mul.d FR_{dest}, FR_{src1}, FR_{src2}	Înmultire în dubla precizie
mul.s FR_{dest}, FR_{src1}, FR_{src2}	Înmultire în simpla precizie

Calculeaza produsul celor doi registrii FR_{src1} si FR_{src2} si pune rezultatul în registrul FR_{dest}.

neg.d FR_{dest}, FR_{source}	Neaga valoare dubla precizie
neg.s FR_{dest}, FR_{source}	Neaga valoare simpla precizie

Neaga valoarea registrului sursa si o depune în registrul destinatie.

s.d FR_{dest}, address	Memorare valoare dubla precizie la adresa
s.s FR_{dest}, address	Memorare valoare simpla precizie la adresa

Scrie o valoare în dubla (sau simpla) precizie la adresa specificata.

sub.d FR_{dest}, FR_{src1}, FR_{src2}	Scadere în dubla precizie
sub.s FR_{dest}, FR_{src1}, FR_{src2}	Scadere în simpla precizie

Calculeaza diferenta celor doi registrii FR_{src1} si FR_{src2} si pune rezultatul în registrul FR_{dest}.

3.5. Utilizarea memoriei si conventii de apel

Organizarea memoriei în sistemele MIPS este convetionala. Spatiul de adresa al unui program este compus din trei parti. Prima zona din spatiul de adresa utilizator (începe la 0x400000) este segmentul text, care memoreaza instructiunile programului. Deasupra segmentului de text este segmentul de date (începând de la adresa 0x1000000) si este împartita în doua parti. Zona de date statica contine obiecte a caror marime si adresa sunt cunoscute de catre compilator si link-editor. Imediat, deasupra acestei zone se afla datele dinamice. La alocarea spatiului dinamic de memorie pentru un program (prin **malloc** si apelul sistem **sbrk**) marginea superioara a segmentului de date se deplaseaza în sus. La marginea superioara a spatiului de adresa (0x7fffffff) este stiva de program, care creste în jos, spre segmentul de date.

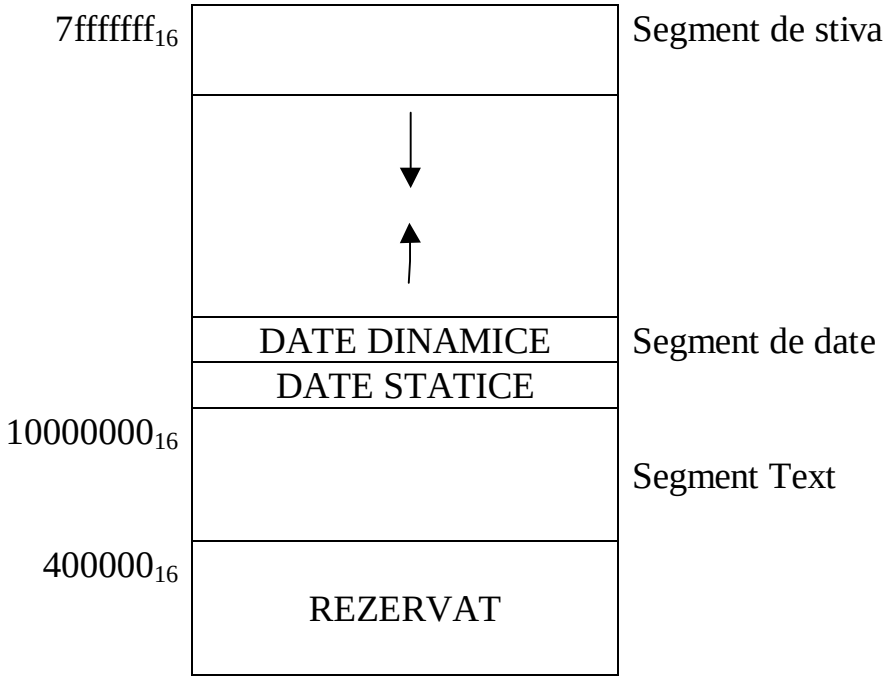


Figura 2. Harta de memorie a microprocesorului MIPS R2000

Conventiile de apel descrise folosesc **gcc**, nu compilatorul nativ al MIPS, care foloseste o conventie mai complexa si usor mai rapida. Diferenta dintre cele doua compilatoare este ca, de obicei, compilatorul MIPS nu foloseste un indicator de cadru, astfel încât acest registru este disponibil ca un alt registru de salvare $\$s_8$.

Un *cadru* consta în memoria dintre indicatorul de cadru ($fp), care pointeaza la cuvântul imediat urmator ultimului argument transmis pe stiva, si indicatorul de stiva ($sp), care pointeaza la primul cuvânt liber pe stiva. Tipic pentru sistemele UNIX, stiva creste în jos de la adrese de memorie mai mari, astfel încât indicatorul de cadru este deasupra indicatorului de stiva.

Pentru a efectua un apel sunt necesari urmatorii pasi:

a. Transmiterea argumentelor. Prin conventie, primele patru argumente sunt transferate în registrii $\$a_0$ - $\$a_3$ (chiar unele compilatoare mai simple simplifica aceasta conventie si transfera toate argumentele pe stiva). Argumentele ramase sunt puse pe stiva.
b. Salveaza registrii $\$t_0$ - $\$t_9$.
c. Executa instructiunea *jal*.

În subrutina apelata sunt necesari urmatorii pasi:

a. Se calculeaza cadrul de stiva prin scaderea marimii cadrului din indicatorul de stiva.
b. Salveaza registrii $\$s_0$ - $\$s_7$ în cadru. Registrul $fp este întotdeauna salvat, iar $ra e necesar a fi salvat doar când subrutina apeleaza alte subrutine.
c. Stabileste indicatorul de cadru prin adaugarea dimensiunii cadrului de stiva la adresa din $sp.

În final, la revenirea din subrutina, rezultatul unei functii este plasat în registrul $\$v_0$ si se executa urmatorii pasi:

a. Restaureaza toti registrii care au fost salvati la intrarea în subrutina.

b. Reface cadrul de stiva prin scaderea dimensiunii cadrului din $sp.

c. Se revine la adresa specificata de registrul $ra.

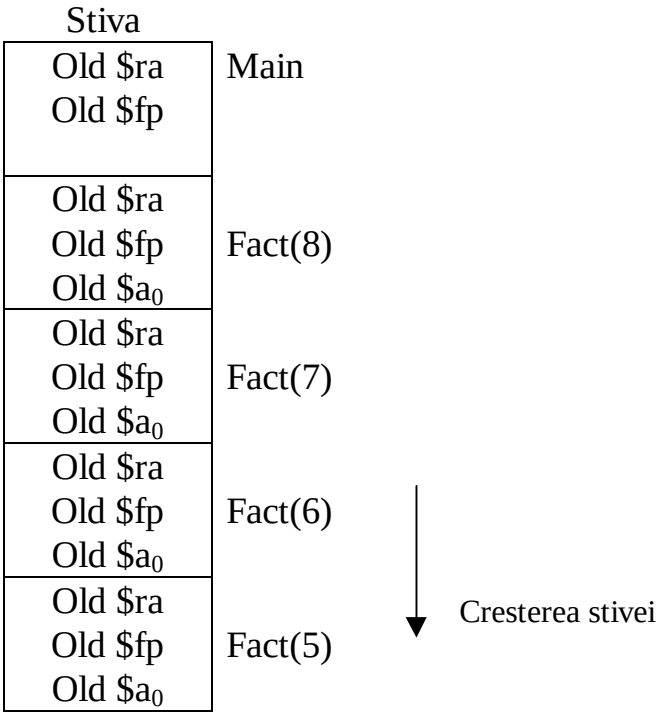


Figura 3. Imaginea stivei pentru calculul valorii fact(8)

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

[3] **Vintan L.** – *Metode de evaluare si optimizare în arhitecturile paralele de tip I.L.P.*, Editura Universitatii "Lucian Blaga" Sibiu, Sibiu, 1997.