CN1_Cursul 11.2

Proiectarea unui procesor didactic.

Dupa cum este cunoscut, procesorul reprezinta, in notatia PSM, ansamblul:

P = D — K

unde:

- P - procesor,
- D - unitate de executie (operator asupra datelor),
- K – unitate de comanda (controlor).

Completat cu memorie (M) si cu intrari/iesiri (T), procesorul P se constituie intr-un calculator (C).

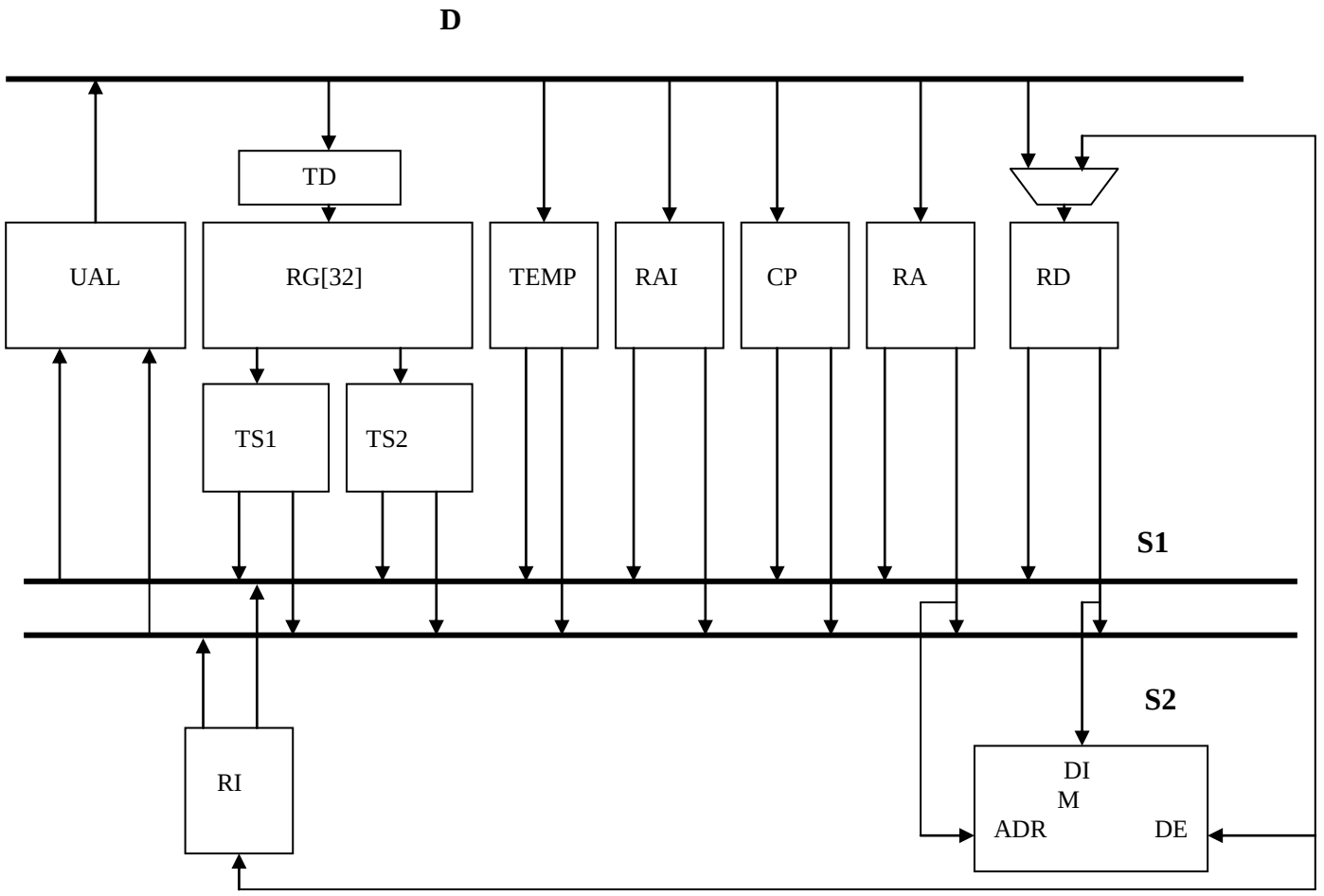
C = M— P— T

Specificatiile initiale ale procesorului didactic:

- Arhitectura centrata pe 32 registre generale RG;
- Lungimea cuvantului: 32 biti, cu posibilitati de manipulare a informatiei pe: octet/byte (8 biti), semicuvant (16 biti), cuvant (32 biti) si cuvant dublu (64 biti);
- Operare in binar, atat cu numere intregi cu semn (complementul fata de doi) si fara semn , cat si cu numere in virgula mobila (standardul IEEE 754) in formatele scurt si lung;

- Unitatea de executie poseda trei magistrale: doua sursa S1, S2 si una destinatie D;
- Registrele generale RG[32:32] sunt de tipul biport si dispun de doua registre tampon de iesire TS1, TS2 si un registru tampon de intrare TD, transparente pentru utilizator;
- RG[0] are continutul egal cu 0;
- Pentru operarea in virgula mobila se foloseste un set de 32 de registre F[32:32] cu dimensiunea de 32 de biti, care pot fi utilizate si sub forma de perechi (F0, F2,…,F30) in conjunctie cu operanzii de lungime dubla;
- Un registru special este prevazut pentru stocarea informatiei de stare, la operarea in virgula mobila: rezultatele comparatiilor, exceptiile etc.;
- Memoria este adresata in modul Big Endian, cu adrese de 32 de biti;
- Schimbul de date, intre memorie si RG sau F, se efectueaza prin instructiuni de tip Incarca/Memoreaza;
- Accesele care implica RG pot fi pe octet, semicuvant, cuvant;
- Accesele care implica F pot fi in simpla si dubla precizie;
- Accesele la memorie trebuie sa fie aliniate;
- Toate instructiunile au 32 de biti.
- Instructiunile au trei formate: R, I, J;
- Instructiunile se incadreaza in 4 clase: Incarca/Memoreaza/Transferuri de date, Operationale (legate de UAL), Ramificari/Salturi, Virgula Mobila

Organizarea procesorului pentru intregi:



Resursele Hardware ale procesorului :

- UAL (Unitatea Aritmetica Logica pentru intregi)
- RG[32:32]- registre generale biport;
- TS1, TS2, TD- registre tampon pentru registrele generale, transparente pentru programator;
- TEMP- registru temporar, transparent pentru programator;
- RAI – Registrul Adresei de Intrerupere (Registru Special);
- CP – Contorul de program;
- RA, RD Registrul de Adrese, Registrul de Date
- RI – Registrul Instructiunii;
- M – memoria principala

Operatiile UAL:

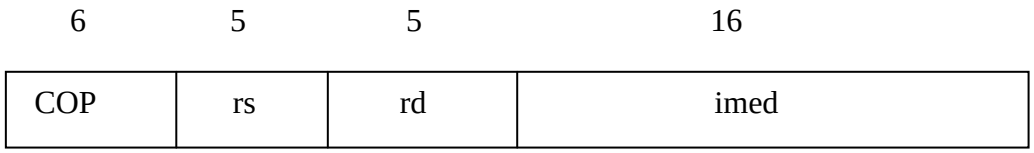
- operatii aritmetice/logice: S1+S2; S1 – S2; S1 $\wedge$ S2; S1 $\vee$ S2; S1 $\oplus$ S2;
- operatii de deplasare: S1 << S2 (deplasare logica stanga); S1 >> S2 (deplasare logica dreapta); S1 >>a S2 (deplasare aritmetica dreapta);
- constante: 0; 1.

Formatele Instructiunilor

Toate instructiunile au 32 de biti, dintre care 6 sunt folositi pentru codificarea operatiei.

Dupa format, instructiunile pot fi de trei tipuri:

1. **Instructiuni de tip I** (Imediat):



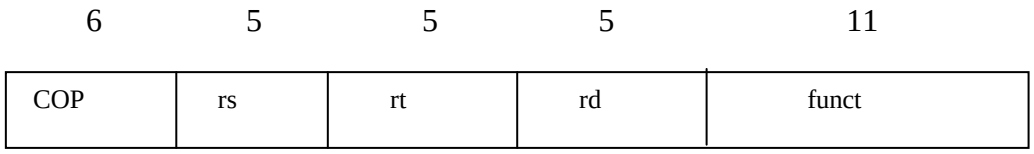
unde:

- COP reprezinta codul operatiei;
- rs, rd sunt adresele registrului sursa si registrului destinatie;
- imed constituie fie adresa, fie operand imediat

Instructiunile care se incadreaza in acest tip sunt urmatoarele:

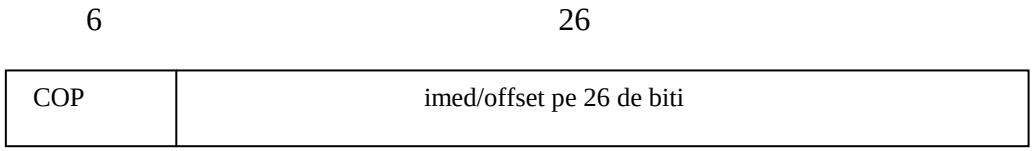
- Incarca/Memoreaza octet, semicuvant, cuvant;
- Immediate (cu operand imediat): RG[rd] ←RG[rs] @ imed, unde @ reprezinta un operator aritmetic/logic;
- Ramificare conditionata: rs – registru, rd – nefolosit;
- Salt la adresa specificata de registru, cu sau fara revenire (cu sau fara legatura): rt = 0, rd = destinatie, imed=0.

2. Instructiuni de tip R (Registru - Registru).



Operanzii se gasesc in registrele generale specificate de campurile rs si rt, iar rezultatul este depus in registrul specificat de campul rd. Campul funct reprezinta o extensie a codului de operatie, pentru a putea codifica mai multe instructiuni, decat ar permite un camp de cod de operatie de 6 biti. Acest camp codifica instructiunile operationale aritmetice si logice, instructiunile de scrie/citeste in /din registrul special RAI si instructiunile de deplasare: RG[rd] ← RG[rs] (funct) RG[rt].

3. Instructiunile de tip J.

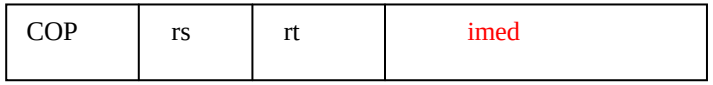


Campul imed/offset este utilizat pentru generarea adresei de salt.

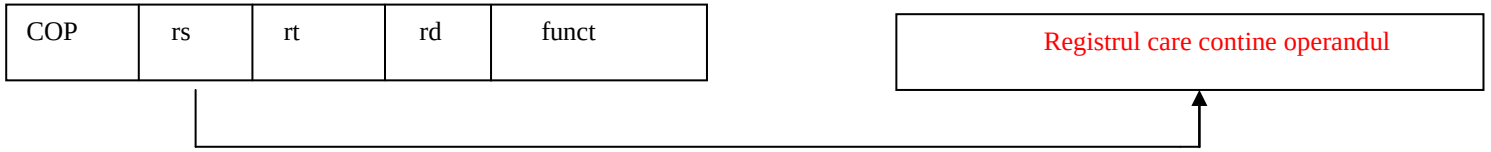
Instructiunile care intra in aceasta categorie sunt cele de salt simplu, salt cu legatura, revenire (RET) si intrerupere (TRAP).

Modurile de adresare prevazute in instructiuni.

1. **Imediat**. Operandul se afla in instructiune:

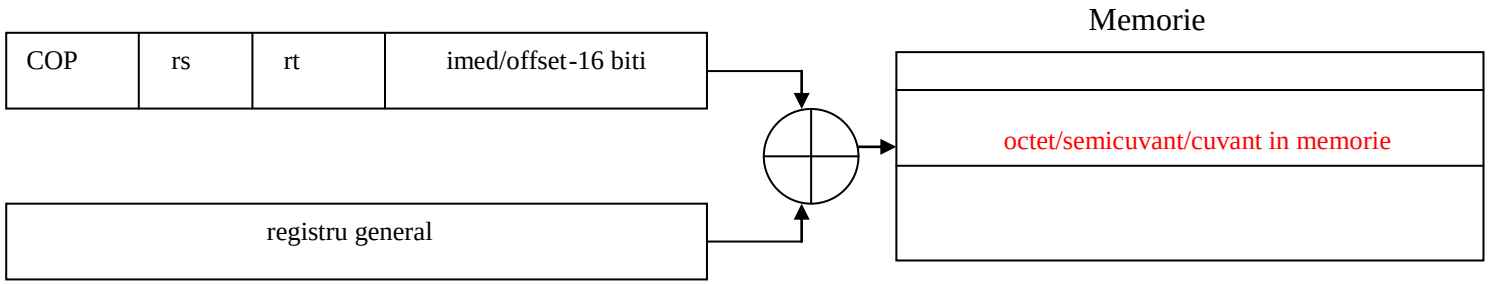


2. **Registru**. Operandul se afla intr-un registru specificat de campul rs.

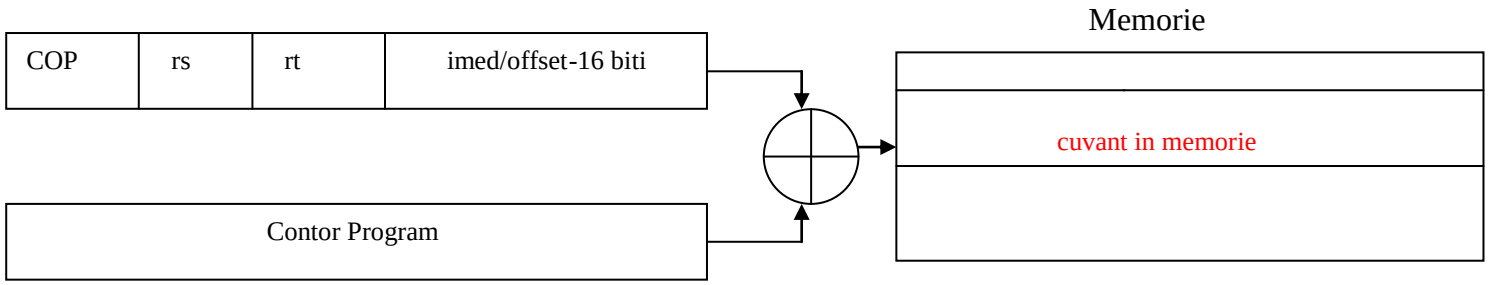


3. **Cu deplasare bazata*)** Continutul unui registru se aduna cu campul imediat pentru a forma adresa.

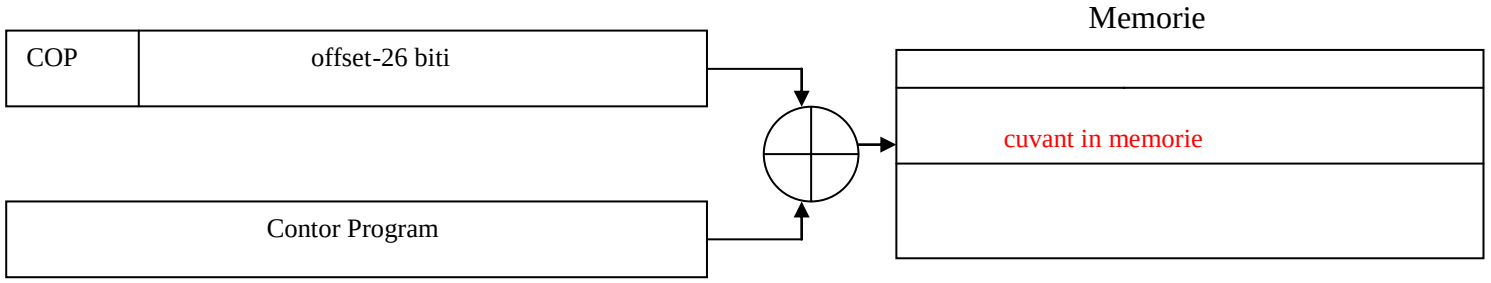
*) MIPS



4. Relativa la CP, cu offset de 16 biti (extensia semnului).



5. Relativa la CP, cu offset de 26 de biti (extensia semnului).



In principal sunt prevazute doua moduri de adresare: imediat (16 biti) si cu deplasare(16 biti), considerand semnul pentru imediat/deplasare. Adresarea cu deplasare presupune adunarea continutului lui RG[rs] la deplasare. Daca deplasarea este zero rezulta adresare prin registru. In cazul in care se foloseste RG[0] se obtine adresarea absoluta.

Prezentarea instructiunilor dupa operatiile efectuate

Dupa operatiile efectuate instructiunile procesorului didactic se pot incadra in urmatoarele tipuri:

- Incarca/Memoreaza si Transfer de Date ;
- Operatii UAL;
- Ramificari/Salturi;
- Operatii in VM.

1. Instructiuni Incarca/Memoreaza si Transfer de Date.

Aceste instructiuni transfera datele intre registre si memorie sau intre registrele pentru intregi RG si cele pentru VM sau registrele speciale. Pentru calculul adresei operandului instructiunile folosesc un registru si/sau deplasarea (imed) pe 16 biti, cu extensia bitului de semn, din rangul 16 al cuvantului instructiune. La memorarea pe octet sau semicuvant rangurile superioare ale cuvantului se completeaza prin extinderea rangului de semn al octetului sau semicuvantului. Incarcarile/Memorarile numerelor in VM respecta standardul IEEE754.

Exemple:

LB, LBU, SB Incarca octet, incarca octet fara semn, memoreaza octet.

LH, LHU, SH Incarca semicuvant, incarca semicuvant fara semn, memoreaza semicuvant.

LW, SW Incarca cuvant, memoreaza cuvant.

LF,LD,SF,SD Incarca cuvant simplu VM, incarca cuvant dublu VM, memoreaza cuvant simplu VM, memoreaza cuvant dublu VM

MOV I2S, MOV S2I Transfera de la/la RG la/de la registru special

MOVF, MOVD Copiaza un registru simplu VM (FP) sau registru dublu VM (DP) intr-un alt registru sau pereche de registre.

MOV FP2I, MOV I2FP Transfera un cuvant de 32 de biti de la/la registrele simple in VM (FP) la/de la registrele pentru intregi RG

Exemple de utilizare:

LW R1, 30(R2)	incarca cuvant	RG[1] $\leftarrow_{32}$ M[30 +RG[2]]
LW R1, 1000(R0)	incarca cuvant	RG[1] $\leftarrow_{32}$ M[1000+0]
LB R1, 40(R3)	incarca octet	RG[1] $\leftarrow_{32}$ (M[40 + RG[3]]$_0$)^{24} ## M[40 + RG[3]]
LBU R1, 40(R3)	incarca octet fara semn	RG[1] $\leftarrow_{32}$ 0^{24} ## M[40 + RG[3]]
LH R1, 40(R3)	incarca semicuvant	RG[1] $\leftarrow_{32}$ (M[40 + RG[3]]$_0$)^{16} ## M[40 + RG[3]] ## M[41 + RG[3]]
LF F0, 50(R3)	incarca simplu VM	F[0] $\leftarrow_{32}$ M[50 + RG[3]]
LD F0, 50(R2)	incarca dublu VM	F[0] ## F[1] $\leftarrow_{64}$ M[50 + RG[2]]
SW 500(R4), R3	memoreaza cuvant	M[500 + RG[4]] $\leftarrow_{32}$ RG[3]
SF 40(R3), F0	memoreaza simplu VM	M[500 + RG[4]] $\leftarrow_{32}$ F[0]
SD 40(R3), F0	memoreaza dublu VM	M[40 + RG[3]] $\leftarrow_{32}$ F[0]; M[44 + RG[3]] $\leftarrow_{32}$ F[1]
SH 502(R2), R3	memoreaza semicuvant	M[502 + RG[2]] $\leftarrow_{16}$ RG[3]$_{16..31}$
SB 41(R3), R2	memoreaza octet	M[41 + RG[3]] $\leftarrow_{8}$ RG[2]$_{24..31}$

2. Instructiuni care efectueaza operatii UAL.

In cadrul acestora se intalnesc instructiunile aritmetice/logice de tip R – R/R – I (unde I este operandul imediat pe 16 biti) si instructiunile de comparare. Operatiile aritmetice cu numere cu semn genereaza intreruperi in cazul depasirii.

In aceasta grupa a fost plasata si instructiunea LHI (Load High Immediate), care incarca jumatatea superioara a unui registru cu operandul imediat de 16 biti, in timp ce jumatatea inferioara a registrului se forteaza la zero. In acest mod se pot construi constante de 32 de biti, folosind doua instructiuni.

Instructiunile de comparare (SET) examineaza continuturile a doua registre sursa din punctul de vedere al relatiilor:

= ,≠, <, >, < =, = >. Daca conditia testata se indeplineste, se forteaza 1 in registrul destinatie, altfel se forteaza 0. Aceste instructiuni "seteaza" un registru conform rezultatului conditiei testate: SEQ, SNE, SLT, SGT, SLE, SGE. Ele au si forme immediate: SEQI, SNEI, SLTI, SGTI, SLEI, SGEI.

Exemple:

ADD, ADDI, ADDU, ADDUI	Adunare, Adunare cu operand imediatat (imediat – 16 biti), pentru operanzi cu si fara semn.
SUB, SUBI, SUBU, SUBUI	Scadere, Scadere cu operand imediat, pentru operanzi cu si fara semn.
MULT, MULTU, DIV, DIVU	Inmultire si Impartire, pentru operanzi cu si fara semn. Operanzii trebuie sa fie in registrele F Rezultatele au 32 de biti.
AND/OR, ANDI/ORI	Inmultire/Adunare logica, Inmultire/Adunare logica cu operand imediat
XOR, XORI	SAU-Exclusiv, SAU-Exclusiv cu operand Imediat
LHI	Incarca jumatatea superioara a registrului cu imediat
SLL, SRL, SRA, SLLI, SRLI, SRAI	Deplasare (S_), Deplasare cu operand imediat (S_I), stanga/dreapta - logic, dreapta–aritmetic
S__, S__I	Seteaza conditional, unde "__" poate fi: LT, GT, LE, GE, EQ, NE.

Exemple de utilizare:

ADD R1, R2, R3	Adunare	RG[R1] ←RG[R2] + RG[3]
ADDI R1, R2, #3	Adunare cu imediat	RG[1] ← RG[2] + 3
LHI R1, #42	Incarca jumatatea superioara imediat	RG[1] ← 42 ## 0^{16}
SLLI R1, R2, #5	Deplaseaza logic imediat	RG[1] ← RG[2] <<5
SLT R1, R2, R3	Seteaza daca mai mic	if (RG[2] < RG[3]) RG[1] ←1 else RG[1] ← 0

3. Instructiunile de Ramificare si Salt.

Controlul programului este asigurat prin instructiunile de ramificare si salt.

Ramificarile sunt conditionate, avand conditia specificata in instructiune. Se testeaza continutul registrului sursa pentru valoarea nonzero/zero, care poate fi o "data" sau rezultatul unei comparatii. Adresa de ramificare este specificata cu o deplasare de 16 biti, cu semn, care este adunata la CP + 4.

Salturile sunt de patru tipuri, diferentiate prin doua moduri de specificare a adresei destinatie si prin realizarea/nerealizarea legaturii.

Doua tipuri de salt folosesc o deplasare de 26 de biti, care se aduna la CP + 4. Alte doua salturi specifica registrul care contine adresa destinatie. In cadrul acestor tipuri saltul se poate face fara legatura sau cu legatura (pentru chemarea de proceduri), care plaseaza adresa de revenire (CP + 4) in RG[31].

Exemple:

BEQZ, BNEZ Ramificare daca RG este/nu este egal cu zero: (CP + 4) + Offset (16 biti).

BFPT, BFPF Testeaza bitul de comparare din registrul special de stare FP si ramifica/nu ramifica: (CP + 4) + Offset (16 biti).

J, JR Salt la (PC+4) + offset (26 biti) sau la adresa specificata de registrul tinta

JAL, JALR Salt la (PC+4) + offset (26 biti) sau la adresa specificata de registrul tinta, cu stocarea adresei de revenire PC + 4 in RG[31]

TRAP Transfera controlul sistemului de operare, la o adresa vector data.

RFE Revenire in programul utilizator, dintr-o situatie de exceptie.

Exemple de utilizare:

J nume	Salt	CP ←nume; (nume = (CP + 4) + $(RI)^6$ ##$RI_{16..31}$) ((CP + 4) – 2^{25}) ≤ nume < ((CP + 4) + 2^{25})
JAL nume	Salt cu legatura	R[31] ←CP + 4; CP ←nume; ((CP + 4) – 2^{25}) ≤ nume < ((CP + 4) + 2^{25})
JALR R2	Salt cu legatura registru	RG[31] ←CP + 4; CP ← RG[2]
JR R3	Salt registru	CP ← RG[3]
BEQZ R4, nume	Ramificare la zero	if (RG[4] == 0) CP ←nume; ((CP + 4) – 2^{15}) ≤ nume < ((CP + 4) + 2^{15})
BNEZ R4, nume	Ramificare la non zero	if (RG[4] != 0) CP ←nume; ((CP + 4) – 2^{15}) ≤ nume < ((CP + 4) + 2^{15})

4. **Instructiuni pentru virgula mobila.**

Instructiunile in virgula mobila utilizeaza registrele FP si specifica daca operatiile se efectueaza in precizie simpla sau dubla, utilizand sufixul F sau sufixul D.

Exemple:

ADDD, ADDF	Adunare in dubla, simpla precizie
SUBD, SUBF	Scadere in dubla, simpla precizie
MULTD, MULTF	Inmultire in dubla, simpla precizie
DIVD, DIVF	Impartire in dubla, simpla precizie
CVTF2D, CVTF2I	Conversii de tip: CVTx2y converteste de la tipul x la tipul y, unde x si y sunt I (intreg), D(VM- precizie simpla, F (VM – precizie dubla). Ambii operanzi sunt in registrele F (VM).
__D, __F	DP si SP compara : "__" = LT, GT, LE, GE, EQ, NE; si seteaza bitul corespunzator in registrul de stare FP

Eficienta procesorului.

Ecuatia care evalueaza timpul necesar executarii unui program:

Timp UCP = Nr. de Instructiuni pentru un program dat x CPI x Perioada ceasului,

unde: CPI reprezinta numarul mediu de cicluri (perioade de ceas) pe instructiune. CPI rezulta ca o medie a ciclurilor pentru fiecare instructiune inmultite cu frecventa de aparitie (%) a fiecarei instructiuni in programul dat.

Evaluarea frecventelor de aparitie ale instructiunilor unui procesor se poate realiza folosind programele SPECintxx si SPECfpxx (System Program Evaluation Cooperative), pentru intregi si pentru VM.

Ansamblul 5SPECint92 consta in urmatoarele programe:

- compress (1503 linii de cod C) efectueaza compresia de date pe un fisier, folosind codificarea Lempel-Ziv;
- eqntott (3376 linii de codC) traduce o Ecuatie Booleana intr-o tabela de adevar;
- espreso (13500 linii de codC) Minimizeaza functii Booleene,
- gcc(cc1) (89589 linii de cod C) Compilatorul GNU C care converteste fisiere preprocesate in cod de asamblare optimizat pentru Sun-3
- sc (8116 linii de cod C) efectueaza calcule intr-o tabela UNIX de tip Spreadsheet

Grupul 5SPECfp92 este constituit din programele:

- doduc (5334 linii de cod FORTRAN) realizeaza o simulare Monte Carlo pentru componenta unui reactor nuclear;
- ear (4483 linii de cod C) un model al urechii interne, care filtreaza si detecteaza diferite sunete si genereaza semnale vocale; foloseste precizia simpla;
- hydro2d (4461 linii de cod FORTRAN) calculeaza jeturile galactice, pe baza ecuatiilor Navier-Stokes – Astrofizica;
- mdljdp2 (4458 linii de cod FORTRAN) solutioneaza ecuatiile de miscare a 500 de atomi – Chimie;

- su2cor (2514 linii de cod FORTRAN) calculeaza masele particulelor elementare folosind teoria Quark-Gluon.

Frecventele de aparitie ale instructiunilor in:

	5SPEC92int	5SPEC92fp		5SPEC92int	5SPEC92fp
incarca	26%	1%	add FP	0%	8%
memoreaza	9%	1%	sub FP	0%	6%
add	14%	11%	mul FP	0%	13%
sub	0%	0%	div FP	0%	1%
mul	0%	0%	compara FP	0%	6%
div	0%	0%	move R-R FP	0%	2%
compara	13%	2%	altele FP	0%	2%
incarca imediat	3%	1%			
ramificare cond.	16%	8%			
ramificare necond.	1%	0%			
call	1%	1%			
return, salt ind.	1%	1%			
deplasare	4%	2%			
and	3%	0%			
or	5%	0%			
altele (xor, not)	1%	0%			
incarca FP	0%	23%			

memoreaza FP 0% 9%

O versiune mai recenta a SPECxx Benchmark o reprezinta SPEC95, care consta in 8 programe pentru operatii cu intregi si 10 programe pentru operatii in virgula mobila. Timpii de executie sunt mai intai normalizati prin impartirea timpului de executie corespunzator programului pe statia Sun SPARC 10/40 la timpul de executie pe masina data. Aceasta normalizare furnizeaza indicatorul SPEC ratio, care are avantajul ca este cu atat mai mare, cu cat performanta masinii testate este mai mare (SPEC ratio este invers proportional cu timpul de executie).

Ansamblul SPECint95 consta in urmatoarele 8 programe scrise in limbajul C:

- go: Inteligenta artificiala; Jocul GO
- m88ksim: Simulatorul pentru procesorul Motorola 88K; ruleaza program de test
- gcc: Compilatorul Gnu C, care genereaza cod SPARC
- compress: Compresie/decompresie fisiere in memorie
- li: Interpretor de Lisp
- ijpeg: Compresie/decompresie grafica
- perl: Manipuleaza siruri si numere prime in limbajul specializat de programare Perl
- vortex: Program de baze de date.

Ansamblul SPECfp95 consta in urmatoarele 10 programe scrise in limbajul Fortran 77:

- tomcatv: Program de generare a unei grile
- swim: Model pentru ape putin adanci cu o grila de 513 x 513
- su2cor: Fizica Cuantica; Simulare Monte Carlo
- hydro2d: Astrofizica; Ecuatiile Navier Stocks, din Hidrodinamica
- mgrid: Solver multigrila pentru un camp 3D, generat de un potential

- applu: Ecuatii cu derivate partiale Parabolice si Eliptice
- turb 3d: Simuleaza turbulenta omogena, isotropica intr-un cub
- apsi: Solutioneaza probleme privind temperatura, viteza vantului si raspandirea unui poluant
- fpppp: Chimie cuantica
- wave5: Fizica plasmei; simularea electromagnetica a particulelor.

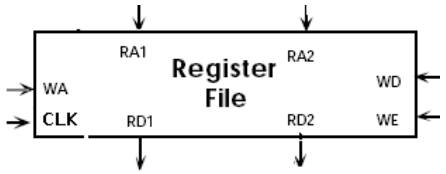
Problema:

1. Sa se calculeze pentru DLX valoarea efectiva a CPI, plecand de la datele din tabelul frecventelor de aparitie a instructiunilor in programele SPECint92, cat si de la valorile CPI-urilor pentru urmatoarele clase de instructiuni, date in tabelul de mai jos:

Instructiunea	Cicluri de ceas pe grupuri de instructiuni
Toate instructiunile UAL	1
Incarca/Memoreaza	1,4
Ramificatii conditionate efectuate (60%)	2,0
Ramificatii conditionale neefectuate (40%)	1,5
Salturi	1,2

CPI = 1 x ,47 + 1,4 x 0,35 + 2 x 0,096 + 1,5 x 0,064 + 1,2 x 0,02 = 1,272 (numarul mediu de cicluri de ceas pe instructiune)

2. Sa se descrie si sa se simuleze in Verilog un fisier de 32 registre generale de cate 32 de biti, RG[32:32], biport. Iesiri date: RD1, RD2; adrese descriere: RA1, RA2; intrare date: WD; adresa scriere date: WA; activare scriere: WE; intrare ceas CLK (operare pe front pozitiv). Se va proiecta logica care stabileste ca RG[0] contine constanta 0.




```
// 2-read, 1-write 32-location register file
module regfile(ra1,rd1,ra2,rd2,clk,werf,wa,wd);
input [4:0] ra1; // address for read port 1 (Reg[RA])
output [31:0] rd1; // read data for port 1
input [4:0] ra2; // address for read port 2 (Reg[RB], Reg[RC] for ST)
output [31:0] rd2; // read data for port 2
input clk;
input werf; // write enable, active high
input [4:0] wa; // address for write port (Reg[RC])
input [31:0] wd; // write data
reg [31:0] registers[31:0]; // the register file itself (local)
// read paths are combinational
// logic to ensure R31 reads as zero is in main datapath
assign rd1 = registers[ra1];
assign rd2 = registers[ra2];
// write port is active only when WERF is asserted
always @(posedge clk)
if (werf) registers[wa] <= wd;
endmodule
```