CN1_Cursul 11.3

Implementarea calculatorului didactic DLX

Cu exceptia instructiunilor in VM, toate instructiunile DLX pot fi descompuse in 5 pasi fundamentali, fiecare pas necesitand una sau mai multe perioade de ceas:

1. **Citire**: Citeste Instructiunea (**IF** -Instruction Fetch).
   RA ← CP; RI ← M[RA]
2. **Decodificare**: Decodifica instructiunea si acces la operanzii din RG (**RD** – Read)
   TS1 ← RG[S1]; TS2 ← RG[S2]; CP ← CP + 4;
3. **Executie**: UAL utilizeaza operanzii pregatiti la pasul anterior (**EX** – Execution)
   Pot avea loc trei tipuri de operatii:
   3.1. **Acces la memorie** (pentru citirea unui operand):
   RA ← TS1 + $(RI_{16})^{16}$ ## $RI_{16..31}$; RD ← RG[rd]
   3.2. **Operatii UAL** (operanzi in R-R sau in R-I):
   UALies ← TS1 op (TS2 sau $(RI_{16})^{16}$ ## $RI_{16..31}$)
   3.3. **Ramificare/Salt**
   UALies ← CP + $(RI_{16})^{16}$ ## $RI_{16..31}$; cond ← (TS1 comp 0)
   sau
   UALies ← CP + $(RI_{6})^{6}$ ## $RI_{6..31}$

- La iesirea UAL apare suma intre operandul imediat (16 sau 26 de biti) cu semn extins si contorul programului, ceea ce reprezinta adresa tinta a saltului;
- Pentru ramificarea conditionata, registrul care a fost citit in TS1, la pasul anterior, este comparat cu 0, pentru a se stabili conditia si a se lua decizia ramificare/nonramificare

4. **Acces la memorie/Ramificare** (**MEM -** Memory)
   4.1. Acces la memorie:
   RD ←M[RA] sau M[RA] ←RD
   4.2 Ramificare:
   daca(cond) CP ← UALies (ramificare)

Pentru instructiunea de salt J (Jump) conditia (cond) este intotdeauna adevarata

5. **Scrierea rezultatului** (**WB** – write back)
   RG[rd] ←UALies sau RD

In registrul destinatie se forteaza UALies sau operandul citit din memorie.

Controlul operatiilor

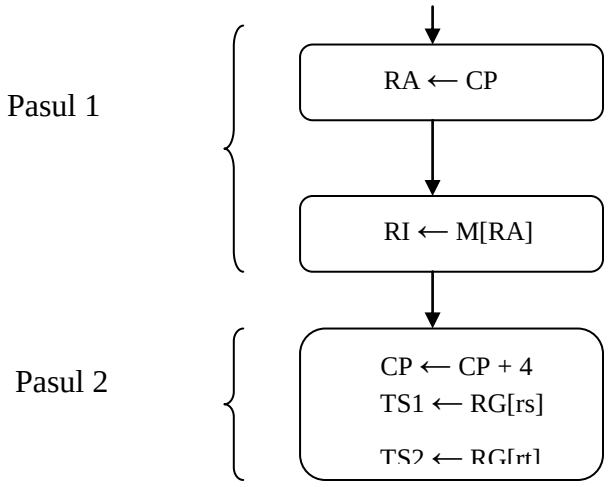
Dupa descrierea **UEx** si a instructiunilor se pune problema proiectarii **Ucd,** care se poate implementa sub forma *conventionala* sau *microprogramata*.

Controlul se poate specifica cu ajutorul unei diagrame finite de stari.

Fiecare stare necesita, de regula, o perioada de ceas.

Fiecare instructiune necesita mai multe perioade de ceas.

Diagrama de stari pentru primii doi pasi din secventa de derulare a unei instructiuni este data mai jos:



In etapa urmatoare se trece de la diagrama de stari la implementarea hardware.

Complexitatea UCd poate fi apreciata astfel:

Complexitatea UCd = nr. stari x *nr. semnale de intrare* x *nr. semnale de iesire*

unde:

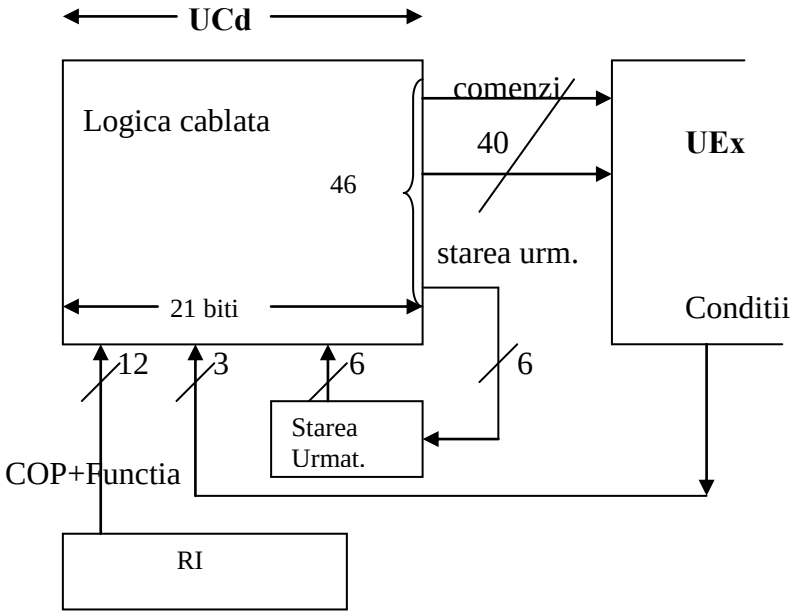
- *nr. stari* se refera la numarul starilor automatului de comanda;
- *nr. semnale de intrare* reprezinta numarul intrarilor testate de catre UCd;
- *nr. iesiri* constituie suma bitilor de comanda si a bitilor starii urmatoare.

Exemplu:

- *nr. stari* = 50, codificate rezulta 6 biti;
- *nr. semnale de intrare* (Codul de operatie + Functia = 16 biti, nr. conditii = 8 (codificat cu 3 biti) , nr. stari codificate = 6 biti), codificate rezulta 21 biti;
- *nr. iesiri* (decodificate) = 40 biti, pentru control si 6 biti pentru starea urmatoare, rezultand 46 de biti

In cazul in care s-ar utiliza o solutie bazata pe o memorie PROM, aceasta ar fi adresata pe 21 de biti (2^{21} cuvinte) si ar avea cuvinte de 46 de biti.

Capacitatea memoriei PROM = 2^{21}cuv. x 46 biti $\cong$ 10 Mb.



Utilizarea unei solutii bazate pe un aPLA ar reduce necesarul de 2^{21} cuvinte numai la 50 cuvinte, prin cresterea complexitatii logicii de decodificare a adresei si prin asignarea starilor.

La proiectarea detaliata a unui calculator se au in vedere:

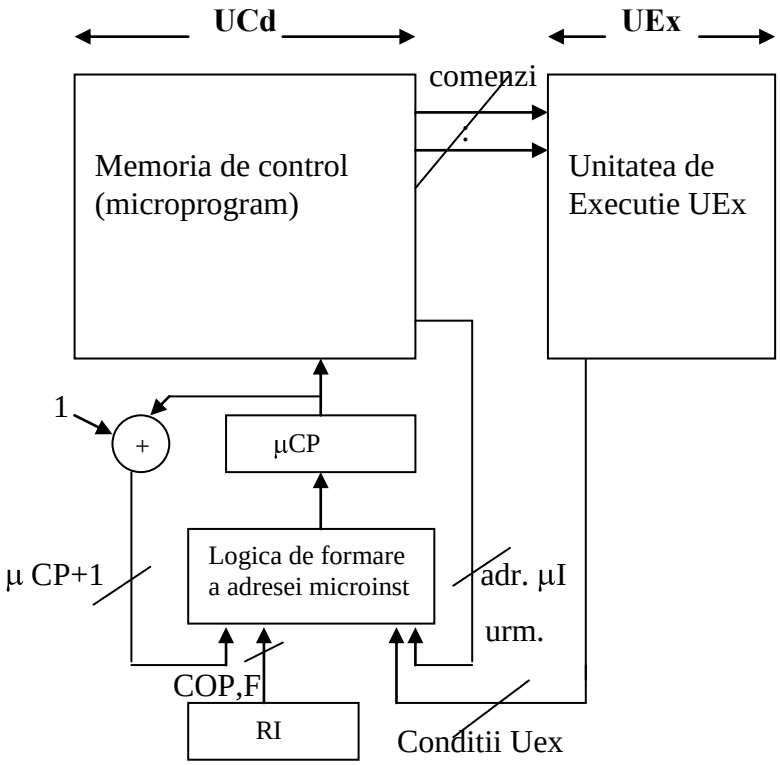
- minimizarea numarului de stari pe instructiune (numarul perioadelor de ceas);
- reducerea perioadei ceasului;
- reducerea cantitatii de hardware pentru implementarea UCd.

Astfel, se poate examina UEx pentru a comasa sau elimina stari. Un exemplu concret ar fi citirea instructiunii din memorie pe baza continutului lui CP: RD ← M[CP] eliminand operatia RA ← CP, ceea ce inseamna reducerea unei perioade de ceas pe instructiune.

Comanda microprogramata (Maurice Wilkes, Univ Cambridge, 1957)

UEx este capabila sa execute o lista de operatii elementare/ microoperatii implementate in hardware. Aceste microoperatii pot fi activate/evocate cu ajutorul unor semnale de comanda, aplicate in punctele de control ale unitatii. La un moment dat pot fi active simultan mai multe microoperatii, care se pot asambla sub forma unui cuvant numit microinstructiune

Implementarea unei instructiuni, pentru masina vazuta de catre programator/masina conventionala, presupune realizarea unei secvente de microinstructiuni denumita microprogram. Microprogramul care se stocheaza intr-o memorie rapida, de control, reprezinta un interpretor pentru lista de instructiuni ale masinii conventionale



Codificarea microinstructiunilor: orizontal, vertical, pe campuri. Codificarea pe campuri se bazeaza pe submultimile constituite din microoperatii care se exclud mutual:

Dest	OpUAL	Sursa1	Sursa2	Constanta.	Dif.	Cond.	Adresa de Salt in microprogram

Adresa microinstructiunii urmatoare se poate genera prin incrementarea lui μPC sau prin fortarea conditionata/neconditionata a unei adrese de salt. Intr-o microinstructiune se pot gasi mai multe adrese de salt.

Optimizarea proiectului are in vedere:

- reducerea capacitatii memoriei de control (nr. de cuvinte si nr. de biti pe cuvant);
- cresterea vitezei de operare etc.

Tehnicile de comanda conventionala si microprogramata sunt examinate prin impactul lor asupra:

- costului hardware-lui;
- perioadei ceasului;
- numarului de perioade de ceas pe instructiune;
- timpului necesar proiectarii/dezvoltarii

1. Metode pentru reducerea costului hardware-lui:

- codificarea semnalelor de comanda;
- formate multiple de microinstructiuni;
- utilizarea controlului cablat pentru partajarea microcodului.

2. Metode pentru reducerea numarului de cicluri pe instructiune:

- Utilizarea microcodului special CASE;
- Adaugarea controlului cablat;

1. Reducerea costului hardware-lui:

1.1. Codificarea semnalelor de comanda:

Gruparea pe campuri codificate a semnalelor care se exclud mutual.

Se considera urmatoarea codificare orizontala/pe linii a campurilor Dest, Sursa1/Sursa2:

Cod	Destinatie	Sursa1/Sursa2	Cod	Destinatie	Sursa1/Sursa2
0	--	TS1/TS2	5	RA	RD
1	TD	TEMP	6	RD	RI (6 biti-imediat)
2	TEMP	CP	7	--	RI (26 biti-imediat)
3	CP	RAI	8	--	Const.
4	RAI	RA 8			

Codificarea orizontala, cate un bit distinct pentru fiecare microperatie, conduce la: urmatorul numar de biti pentru cele trei campuri:

7 biti (Dest) + 9 biti (Sursa1) + 9 (Sursa2) = 25 biti

Codificarea orizontala a campurilor presupune:

$\log_2 7 + \log_2 9 + \log_2 9 = 2{,}8 + 3{,}2 + 3{,}2 \cong 3 + 4 + 4 = 11$ biti, ceea ce asigura o economie de 14 biti.

1.2. Formate multiple de instructiuni

Microinstructiunile pot fi codificate: orizontal, vertical, pe campuri (orizontal/vertical) si pot avea lungimi diferite:

Familia	Modelul	Capacitatea MC[kbiti]	Lungimea μI in biti
IBM360	30	4	50
	40	4	52
	50	2,75	85
	65	2,75	87
IBM370	135	24	16
	145	12	32
	155	6	69
	165	4	105
DEC VAX	730	16	24
	750	6	80
	780	4	96

1.3. Activarea prin μCod a unei scheme cablate/automat de comanda

Se accelereaza anumite operatii cu caracter critic.

2. Cresterea vitezei de operare.

2.1. Reducerea numarului de cicluri de ceas/instructiune folosind μCod suplimentar pentru instructiunile complexe.

VAX 8800 utilizeaza o zona mare a memoriei de comanda MC pentru diverse variante ale instructiunii de tip CALL

2.2 Reducerea numarului de cicluri de ceas/instructiune prin controlul cablat suplimentar

La VAX 11/780 sunt instructiuni de tip R-R si R-M, cu codificare comuna, care se executa in 5 cicluri de ceas. In cazul unei codificari separate instructiunile R-R s-ar executa intr-un singur ciclu de ceas, ca la VAX 8800.

Operarea cu memoria asincrona presupune testarea ciclica cu o microinstructiune a terminarii operatiei. In cazul in care μInstructiunea se pastreaza in registrul μRI, se castiga un ciclu de ceas, de la teminarea operatiei pana la trecerea la μInstructiunea urmatoare. La VAX 11/780 si VAX 8800 20% si respectiv 23% din ciclurile de ceas sunt cicluri de asteptare la memorie. Aceste cicluri de asteptare mai poarta numele de "Stall".

2.3 Reducerea numarului de cicluri de ceas prin paralelism.

Paralelismul se are in vedere la nivelul Uex, prin folosirea codificarii orizontale, care permite, in limitele asigurate de structura hardware si de paralelismul inerent algoritmului, operatii paralele. Aceasta situatie caracterizeaza calculatoarele mari, microprogramate.

Intreruperi

Sistemul de intreruperi al unui calculator are rolul de a suspenda programul in curs de executie in vederea tratarii unor evenimente de natura interna sau externa calculatorului.

Cauze de intrerupere:

- Cereri din partea dispozitiovelor de I/E;
- Invocarea Sistemului de Operare de catre utilizator;
- Trasarea/derularea programului, a instructiunilor pas cu pas;
- Introducerea unor "puncte de rupere" (break points) in program;
- Depasirea aritmetica in virgula fixa si in VM (superioara/inferioara);
- Eroarea de pagina la memoria virtuala;
- Accese nealiniate la memorie;
- Violarea protectiei memoriei;
- Folosirea unor coduri de operatie neimplementate;
- Erori hardware;
- Caderea tensiunii de alimentare etc.

Exemple de intervale in care se pot lua in considerare cererile de intrerupere la VAX 8800:

Eveniment	**Timpul intre evenimente**
- intreruperi provocate de I/E	2,7 ms
- intreruperi de la "contoarele interne"	10,0 ms;
- intreruperi software	1,5 ms;
- alte intreruperi	0,9 ms;
- intreruperi hardware	2,1 ms.

Caracteristicile si tipurile intreruperilor:

- *Sincrone/Asincrone.* Intreruperile sunt sincrone daca evenimentul care le provoaca apare ori de cate ori se executa programul dat, cu acelasi set de date si cu aceeasi alocare a memoriei. Cu exceptia intreruperilor generate de catre hardware si de catre dispozitivele de I/E, care sunt asincrone, celelalte tipuri de intrerupere sunt sincrone;
- *Cerute* de catre utilizator sau *Impuse* de evenimente independente de utilizator;

- *Mascabile sau Nemascabile* de catre utilizator.
- Generate si recunoscute *In cadrul* instructiunilor sau *Intre* instructiuni. Intreruperile recunoscute in cadrul instructiunilor impiedica derularea pana la final a instructiunii;
- Cu *Suspendare sau Terminare* a programului dupa aparitia si tratarea cauzei de intrerupere.

Cereri de intrerupere	Sincrone (S)/ Asincrone(A)	Cerute(C)/ Impuse(I)	Mascabile(M)/ Nemascabile (N)	In cadrul instr.(IC)/ Intre instr.(II)	Suspendat(S)/ Terminat (T)
I/E	A	I	N	II	S
Invoc SO	S	C	N	II	S
Trasare	S	C	M	II	S
Pct. rupere	S	C	M	II	S
Dep.arit.int.	S	I	M	IC	T
Dep.arit.VM	S	I	M	IC	S
Er.prg.	S	I	N	IC	S
Ac.neal.M.	S	I	M	IC	T
Viol.pr.M.	S	I	N	IC	T
COP nedef.	S	I	N	IC	T
Eroare hard.	A	I	N	IC	T
Alimentare	A	I	N	IC	T

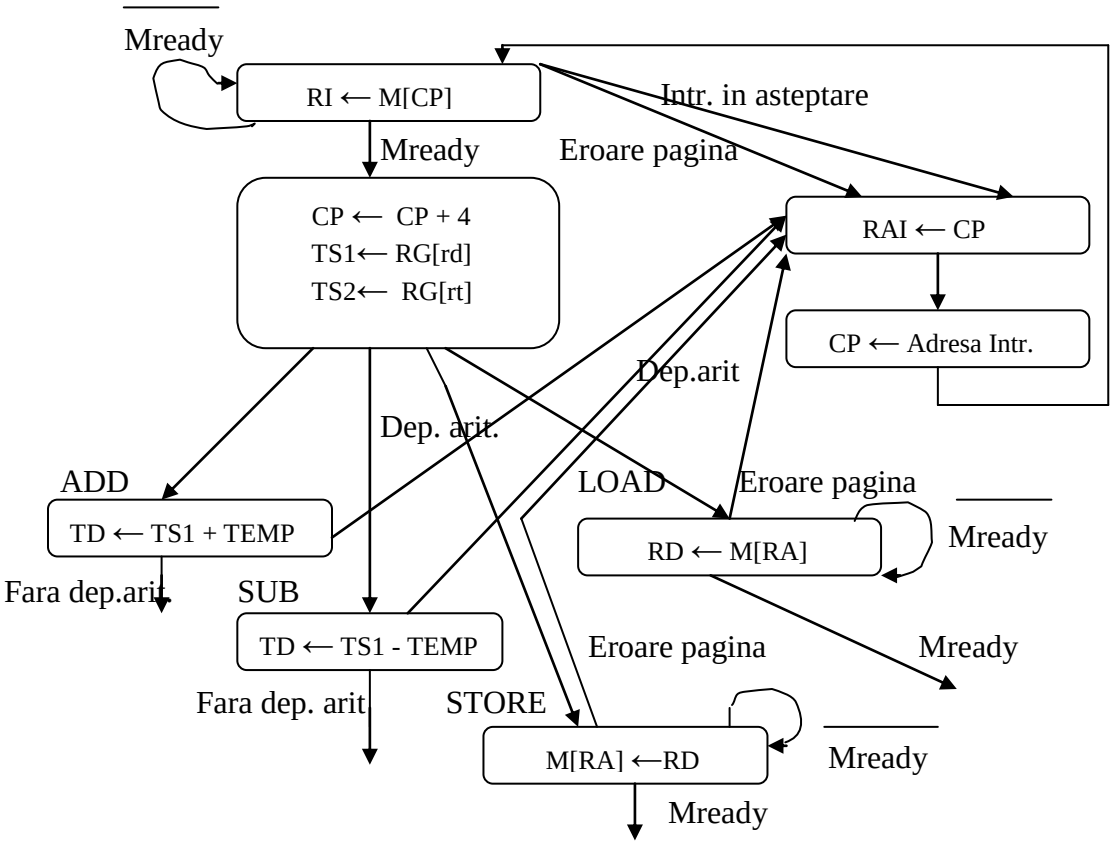
Testarea aparitiei unei cereri de intrerupere de catre UCd

Automatul finit care inplementeaza UCd poate verifica aparitia unei cereri de intrerupere la inceputul executiei, inaintea decodificarii sau la sfarsitul executiei instructiunii. Intreruperile, care apar pe parcursul derularii unei instructiuni, pot fi detectate, fie in starea curenta a automatului de comanda, fie in starea urmatoare.

La recunoasterea unei intreruperi, in mod automat, se realizeaza transferurile:

RAI ← CP; CP ← Adresa de Start a Rutinei de Tratare a Intreruperii.

Organigrama generala a operatiilor legate de aparitia si recunoasterea unei cereri de intrerupere:



Aspecte privind implementarea intreruperilor

Pentru a se putea relua executia programului, dupa tratarea intreruperii, hardware-ul trebuie astfel proiectat incat sa fie salvata intreaga stare a masinii, inclusiv informatia asupara evenimentului care a generat intreruperea.

La intreruperile care sunt recunoscute in cadrul derularii instructiunilor trebuie refacuta starea anterioara aparitiei evenimentului. Numai unele calculatoare “restartabile” poseda aceasta facilitate (IBM 370,DEC VAX s.a.), care memoreaza evolutia automatului de comanda si starea masini, pe durata fiecarei instructiuni.

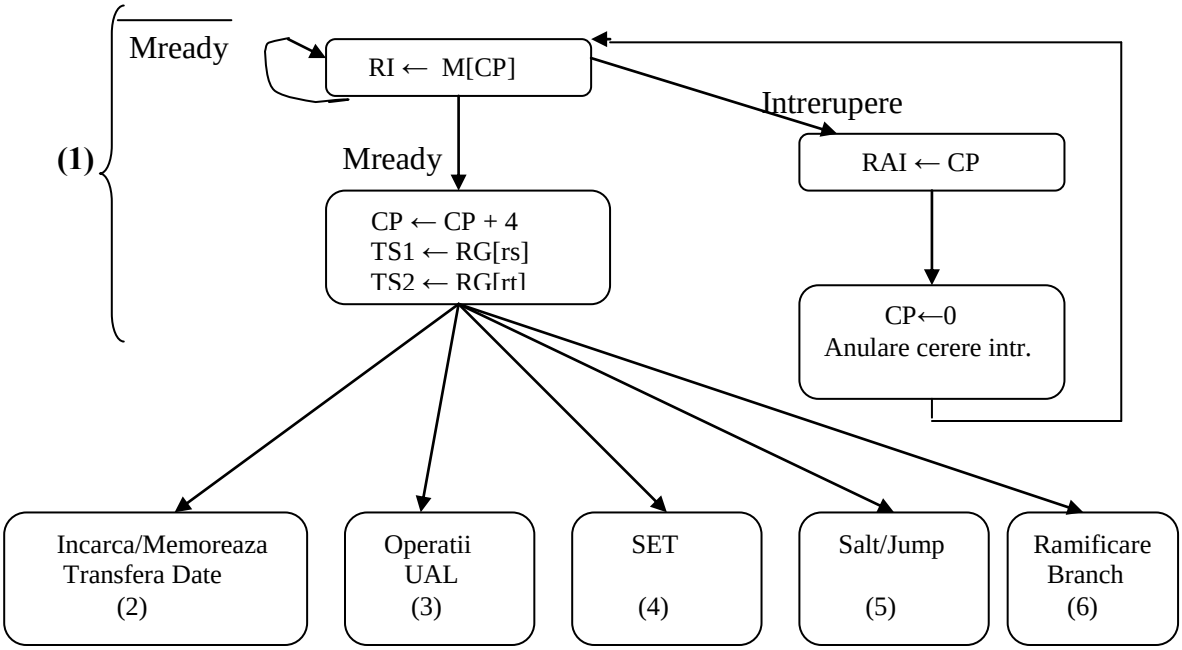
La instructiunile care prelucreaza siruri, cererile de intrerupere se testeaza dupa tratarea fiecarui octet, pentru a nu mari timpul de raspuns la aparitia unui eveniment.

Organigramele privind operarea Unitatii de Comanda.

In cele ce urmeaza vor fi avute in vedere numai instructiunile in virgula fixa, fara a se mai trata aspecte legate de intreruperi.

Pentru inceput se vor examina etapele: Citirea Instructiunii **(IF)** si Decodificare/Citire Registru **(RD)**. Inainte de citirea instructiunii se verifica existenta unei cereri de intrerupere. In caz afirmativ RAI ← CP si CP ← 0.

(1) Diagrama pentru pasii IF si RD



(2) Diagrama pentru pasii EXE si WB ai instructiunilor de Incarca/Memoreaza si Transfer de date (2):

Instructiunile examinate sunt urmatoarele:

- LB, LBU, LH, LHU, LW,
- SW,
- MOVI2S si MOVS2I.

Pentru grupul instructiunilor Incarca/Memoreaza, se va calcula mai intai adresa efectiva, care se va stoca in RA. In continuare instructiunile Incarca si Memoreaza se vor trata

separat. In cazul instructiunilor Incarca se va citi in RD operandul din memorie de la o adresa multiplu de 4. Apoi, in functie de codul de operatie si de ultimii doi biti ai adresei efective, se va extrage operandul din RD si va fi transferat in final la RG[rd].

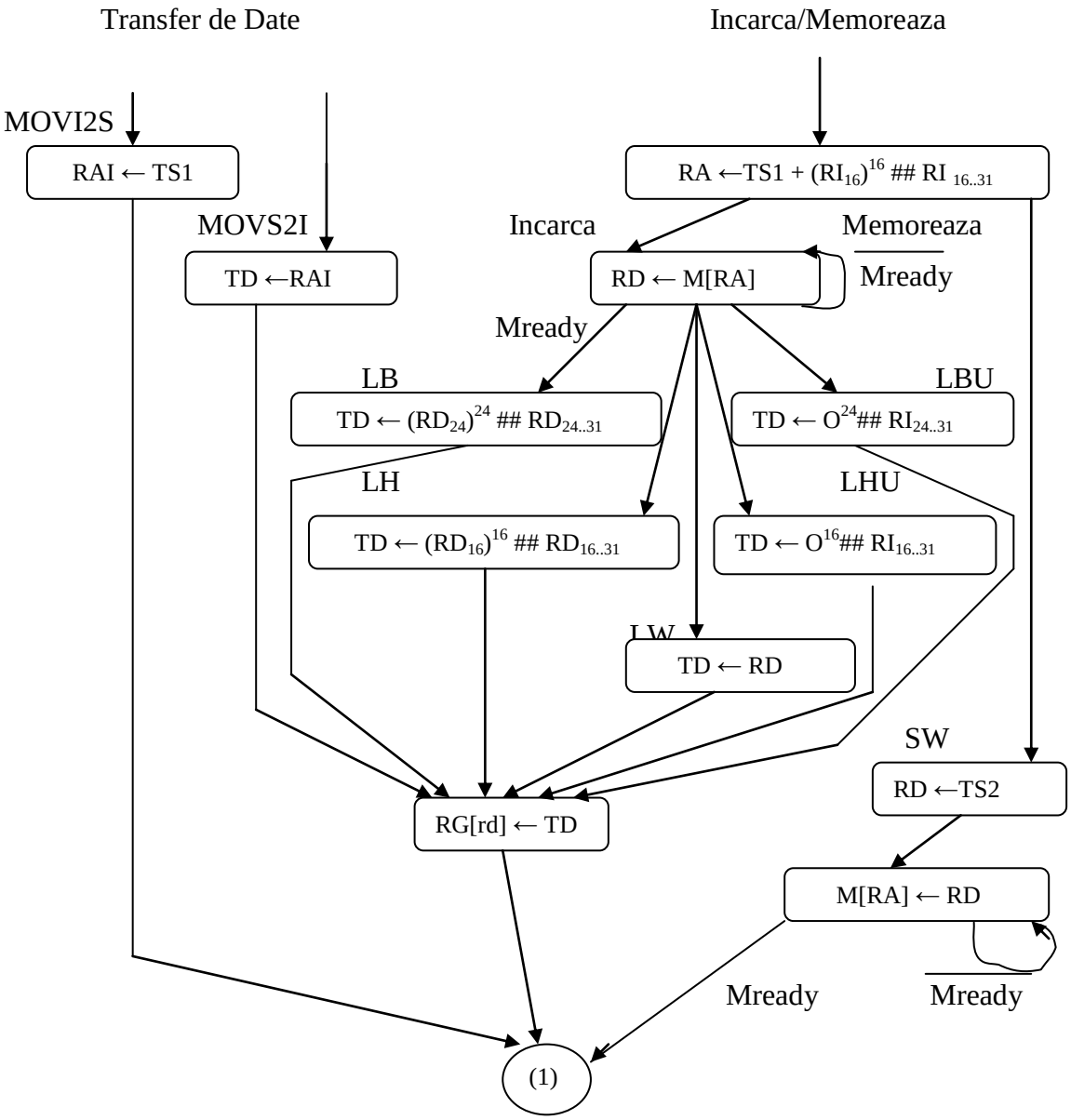
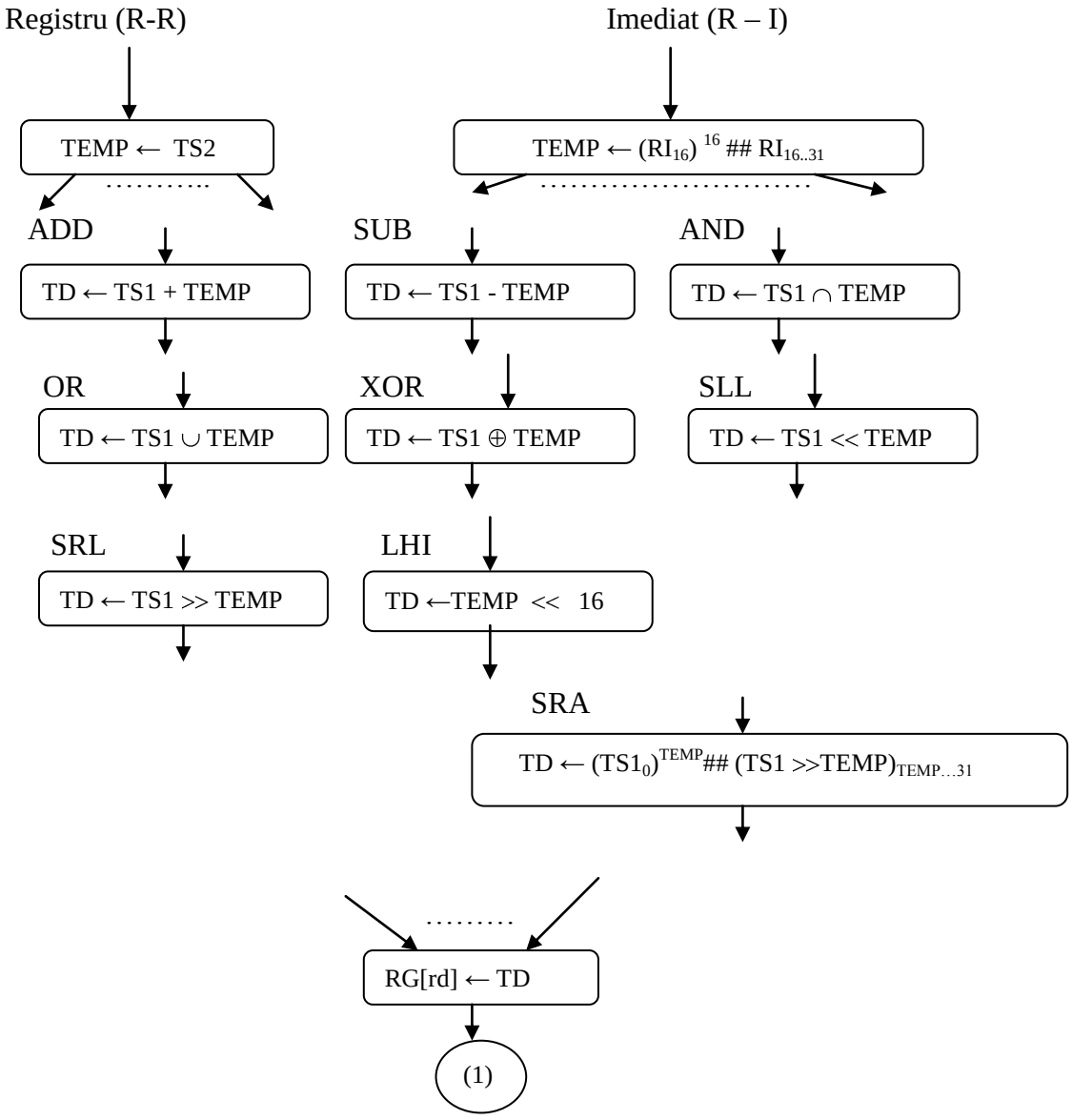


Diagrama de stari pentru pasii EXE si WB ai instructiunilor UAL (3):

Vor fi examinate instructiunile:

- ADD, ADDI, SUB, SUBI,
- AND, ANDI, OR, ORI, XOR, XORI,
- SLL, SLLI, SRL, SRLI, SRA, SRAI,
- LHI.

Pentru a simplifica implementarea hardware si a avea o singura secventa pentru instructiunile R-R si R-I, cel de-al doilea operand este adus in registrul TEMP

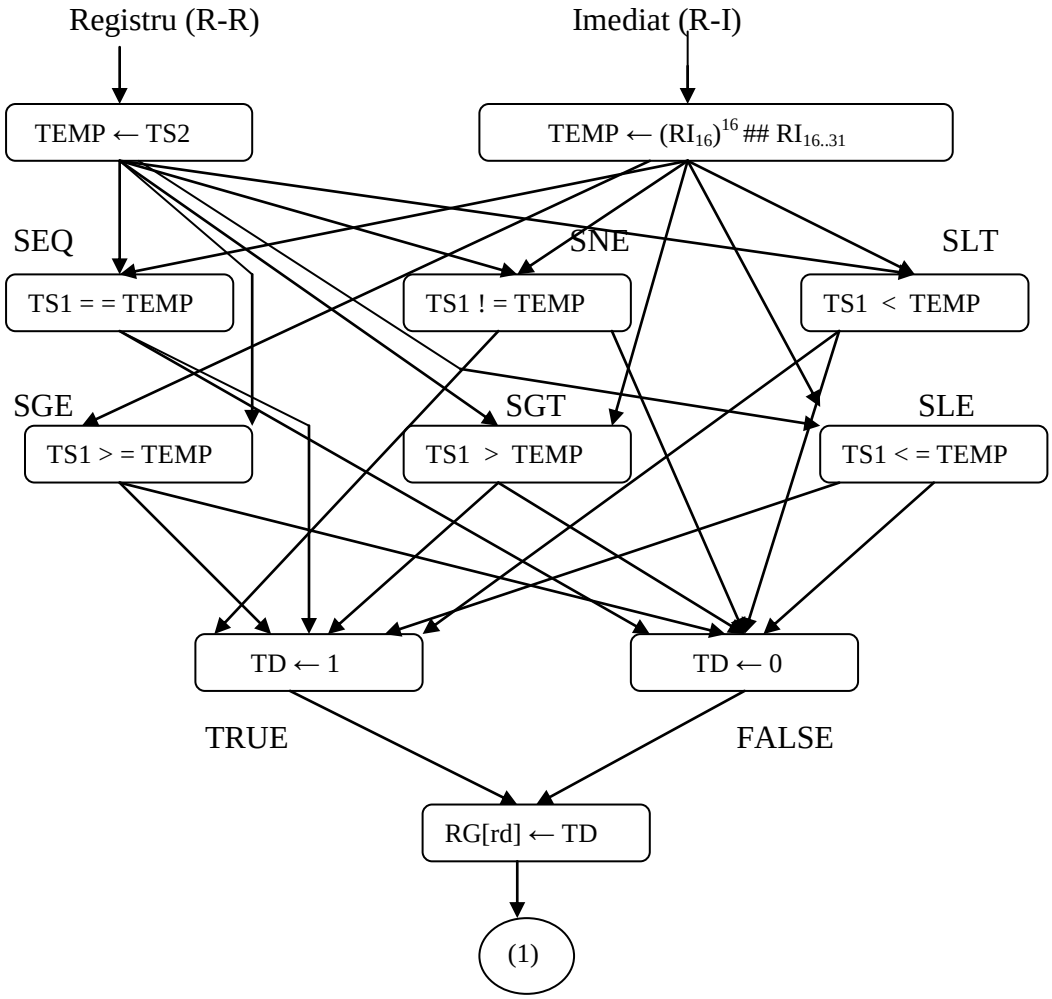


Diagramele pentru pasii EXE si WB ai instructiunilor SET (4):

Dupa plasarea in TEMP a continutului unui registru sau al unui operand imediat, cu semnul extins pe 16 biti, se efectueaza operatia de comparare, a continutului lui TEMP cu continutul lui TS1, specificata in codul de operatie. Rezultatul logic al comparatiei se plaseaza in TD si apoi in RG[rd].

Instructiunile implementate sunt:

- SEQ, SNE, SLT, SGE, SGT, SLE.



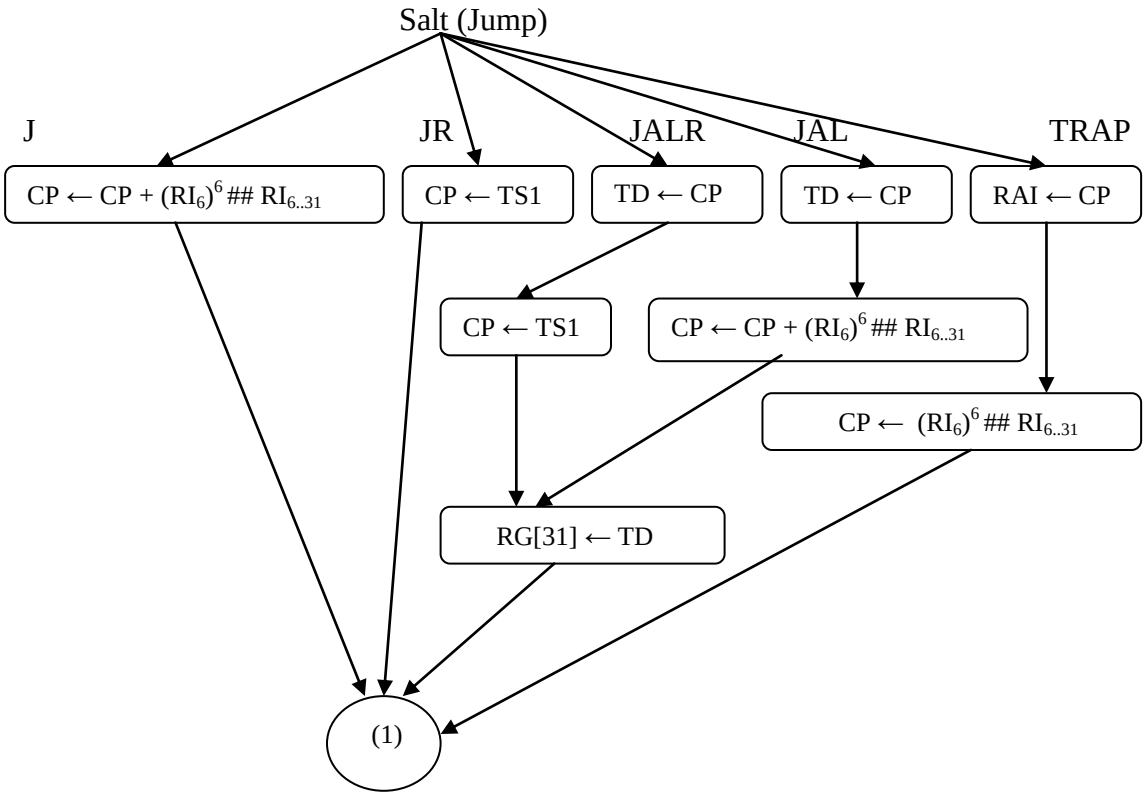
Diagramele pentru pasii EXE si WB ai instructiunilor de Salt (5):

Vor fi examinate instructiunile:

- JR, JALR,
- J, JR
- TRAP.

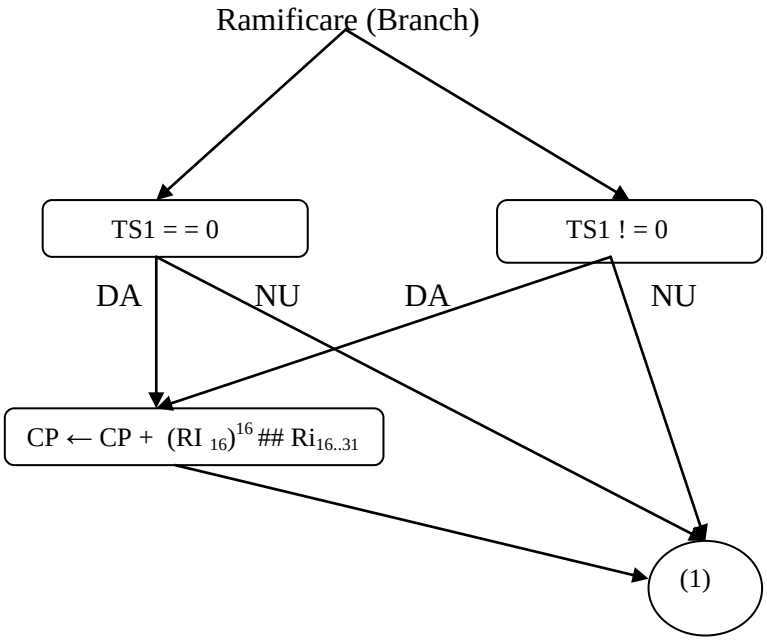
In cazul instructiunilor JAL si JALR adresa de revenire este plasata in TD, inaintea fortarii noii adrese in CP. In continuare TD este stocat in RG[31].

Instructiunea TRAP salveaza adresa in RAI. Offset-ul imediat are 26 de biti cu extensia semnului la stanga..



Diagramele pentru pasul EXE al Instructiunilor de Ramificare (Branch) (6):

Se vor implementa instructiunile BEQ si BNE, care testeaza continutul lui TS1 (in care a fost stocat continutul lui RG[rs]) daca este egal cu zero sau diferit de zero. In primul caz se efectueaza ramificarea la CP + Offset(16 biti), iar in al doilea caz secventa se continua normal, de la CP + 4.



Performantele comenzii cablate pentru procesorul DLX

Obiectivele urmarite se refera la:

- minimizarea CPI (numarul mediu de stari parcurse/perioade de ceas pe instructiune),
- cresterea frecventei ceasului (reducerea perioadei ceasului),
- reducerea cantitatii de hardware,
- reducerea timpului de dezvoltare.

Ca exemplu se considera o unitate de comanda cablata, care opereaza conform diagramelor de stare (1) – (6) si care executa instructiuni DLX - cod GCC,
pentru care se cunosc frecventele de aparitie in executie a instructiunilor DLX.

Plecand de la pasii din diagrama **(1)** si ignorand intreruperile se constata ca fiecare instructiune necesita *2 cicluri de ceas + timpul de asteptare la memorie.*

In continuare se examineaza in acelasi mod diagramele **(2)** – **(6)** si se obtine urmatoarea situatie generala:

Diagrama	Instructiuni	Nr. cicluri	Asteptare la Memorie
(1)	toate	2	1
(2)	Incarcare	4	1
	Memorare	3	1
(3)	UAL	3	0
(4)	SET	4	0
(5)	Salt simplu	1	0
	Salt cu legatura	3	0
(6)	Ramifcare (DA)	2	0
	Ramificare (NU)	1	0

Mai departe se va considera penalizarea impusa de asteptarea la memorie egala cu o perioada de ceas. De asemenea, timpul alocat unei stari a automatului UCd este tot de o perioada de ceas.

In conditiile de mai sus se poate intocmi urmatorul tabel:

Instr. DLX	Nr. minim de perioade de ceas	Penalizare de acces la memorie in perioade de ceas	Total perioade de ceas CPIi)
Incarcare	6	2	8
Memorare	5	2	7
UAL	5	1	6
SET	6	1	7
JMP	3	1	4
JMP&L	5	1	6
Br (True)	4	1	5
Br (False)	3	1	4

Numarul mediu de cicluri pe instructiune CPI se calculeaza astfel:

$$CPI = \sum_{i=1}^{n} (CPIi \text{ x } I/n)$$

unde:

- *I* constituie numarul de aparitii ale instructiunii *i*, in programul dat
- *n* reprezinta numarul total al instructiunilor executate din codul GCC,
- *CPIi* este numarul de cicluri de ceas pentru instructiunea *i*.

Folosind statistica frecventelor de aparitie (in %) a diferitelor instructiuni se obtine:

Instructiunea	Frecventa (*I*/*n*)	*CPIi*	*CPIi* x Frecventa
Incarcare	21%	8	1,68
Memorare	12%	7	0,84
UAL	37%	6	2,22
SET	6%	7	0,42
JMP	2%	4	0,08
MP&L	0%	6	0,00
Br (True)	12%	5	0,60
Br (False)	11%	4	0,44

Total CPI = 6,28 ≅ 6,3 , pentru cod GCC