## 4.4 Unitatea de comanda

Unitatea de comanda controleaza functionarea tuturor modulelor sistemului de calcul, pe baza comenzilor primite de la utilizator prin intermediul programului aflat in curs de executie. Fiecare unitate de comanda, de fapt procesor, are limbajul sau specific de programare la nivelul cel mai de jos, reprezentat de limbajul de asamblare. Limbajul de asamblare specific unui calculator este reprezentat prin setul sau de instructiuni. In general, fiecare instructiune contine urmatoarele informatii: codificarea operatiei de efectuat de catre instructiunea respectiva (codul instructiunii) si unul sau mai multi specificatori de operanzi, operanzi care se pot afla in registrele procesorului, in memorie sau la nivelul porturilor de intrare / iesire. Numarul de biti utilizati pentru codificarea instructiunilor determina numarul maxim de instructiuni care pot fi astfel codificate. Seturile de instructiuni ale calculatoarelor sunt de doua mari categorii:

-seturi de instructiuni de lungime fixa, in care fiecare instructiune ocupa un numar fix de cuvinte in memoria calculatorului;

-seturi de instructiuni de lungime variabila, in care instructiunile pot ocupa un numar variabil de cuvinte in memorie.

**Set de instructiuni**

In continuare, se va studia pe scurt un set general de instructiuni, care pot fi regasite la majoritatea procesoarelor. Aceste instructiuni se pot clasifica in urmatoarele categorii:

-instructiuni pentru transfer de date;

-instructiuni aritmetice;

-instructiuni logice;

-instructiuni pentru transfer de control (de ramificatie).

Se considera ca instructiunile prezentate in continuare pot avea ca operanzi registre ale procesorului, locatii de memorie sau porturi de intrare / iesire. Multe dintre instructiunile unui procesor se executa in stransa legatura cu indicatorii de conditie, care, o parte dintre acestia, au fost discutati la unitatea aritmetica logica. Unele instructiuni pozitioneaza indicatorii de conditie, alte instructiuni testeaza acesti indicatori. Si in cadrul acestui set general, se vor avea in vedere indicatorii de conditie T (transport), Z (zero), S (semn), P (paritate) si D (depasire). Procesoarele actuale contin in plus si alti indicatori.

*Instructiuni pentru transferul de date.* Principalele instructiuni pentru transfer de date, impreuna cu operatiile realizate sunt prezentate in continuare:

**mov dest, sursa** dest ← sursa //*transfera operandul sursa la destinatie*
*//aici sunt incluse si operatiile de citire si scriere cu*
*// memoria si transferurile intre registre*

**xchg dest, sursa** dest ↔ sursa //*interschimba sursa cu destinatia*

**in reg, port** reg ← port *//citire de la un port de intrare*

**out port, reg** port ← reg *//scriere la un port de iesire*

**push reg**
IS ← IS – 1 //IS *indicatorul varfului stivei*
mem(IS) ← reg //*salvarea in varful stivei a registrului*

**pop reg**
reg ← mem(IS) *//extragerea varfului stivei si*
IS ← IS + 1 *//incarcarea informatiei in registru*

**push ind**
IS ← IS – 1 //*salvarea in varful stivei a*
mem(IS) ← ind // *registrului indicatorilor de conditie*

**pop ind**
ind ← mem(IS) *//extragerea varfului stivei si*
IS ← IS + 1 *//incarcarea in registrul indicatorilor*

Instructiunile pentru transferul de date nu modifica indicatorii de conditie, cu exceptia evident a instructiunii *pop ind*, care in mod explicit incarca in indicatorii de conditie o anumita informatie extrasa din varful stivei.

*Instructiuni aritmetice.* De remarcat faptul ca toate instructiunile aritmetice pozitioneaza indicatorii de conditie in functie de rezultatele obtinute in unitatea aritmetica logica.

**add dest,sursa** dest ← dest+sursa *//aduna sursa cu destinatia*
*// memorand rezultatul peste destinatie*

**adc dest,sursa** dest ← dest+sursa+T *//analog, dar in plus aduna*
*//indicatorul de transport in rangul c.m.p.s.*
*//util, pentru adunari pe lungime mare de biti*

**inc sursa** sursa ← sursa+1 *//incrementare*

**sub dest, sursa** dest ← dest–sursa *//scadere*

**sbb dest,sursa** dest ← dest–sursa–T *//scadere cu imprumut*

**dec sursa** sursa ← sursa–1 *//decrementare*

**cmp dest,sursa** dest–sursa *//scade cei doi operanzi, fara memorarea //rezultatului, dar cu pozitionarea indicatorilor de //conditie (comparatie). Daca dupa instructiune,*
*//*T=1 => *dest<sursa*
*//*T=0 => *dest≥sursa*
*//*Z=1 => *dest=sursa*
*//*Z=0 => *dest≠*sursa
*//*Z=0 *si* T=0 => *dest>sursa*

*Instructiuni logice.* Aceste instructiuni executa operatii logice asupra operanzilor. Toate instructiunile pozitioneaza indicatorii de conditie.

**not sursa** sursa ← sursa# *//negatia sau complementarea fata de 1*

**or dest,sursa** dest ← dest SAU sursa *//operatia SAU logic*

**and dest,sursa** dest ← dest SI sursa *//operatia SI logic*

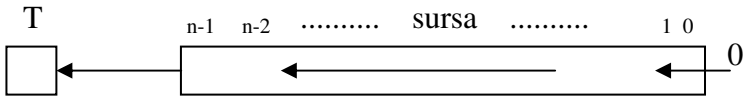
**xor dest,sursa** dest ← dest SAU-EX sursa *//operatia SAU-Excusiv*

**test dest,sursa** dest SI sursa *//fara memorarea rezultatului dar cu //pozitionarea indicatorilor de conditie*
*//util, pentru testarea unui bit (camp de biti). Exemplu:*
*//se doreste aflarea bitului ? din operandul dest*

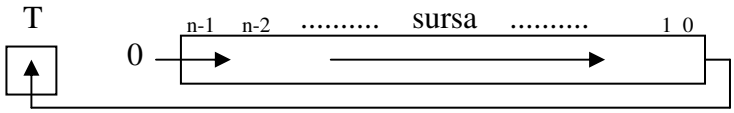
| | | | | | | |
|---|---|---|---|---|---|---|
| dest: | XX ..... XX | ? | XX ..... XX |
| sursa: | 00 ..... 00 | 1 | 00 ..... 00 |
| xor dest,sursa: | 00 ..... 00 | ? | 00 ..... 00 |

*//daca Z=1 => ?=0; daca Z=0 => ?=1*

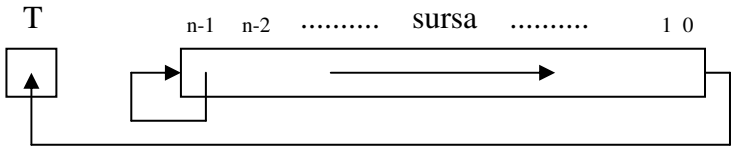
**shl/sal sursa** T,sursa ← $sursa_{n-1:0}$,0 *//deplasare logica/aritm stanga*



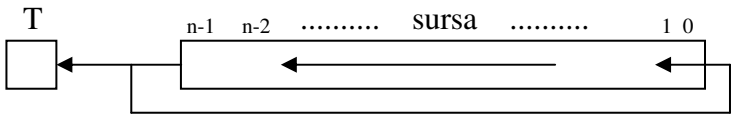

**shr sursa** T,sursa ← $sursa_0$,0,$sursa_{n-1:1}$ *//deplasare logica dr.*



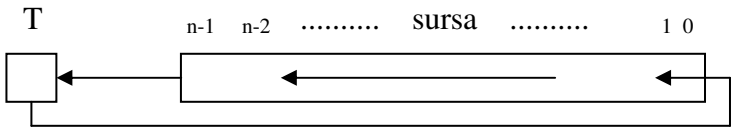

**sar sursa** T,sursa ← $sursa_0$,$sursa_{n-1}$,$sursa_{n-1:1}$ *//deplasare aritm dr.*



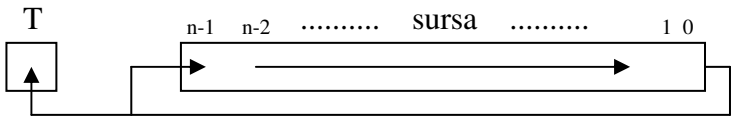

**rol sursa** T,sursa ← $sursa_{n-1:0}$,$sursa_{n-1}$ *//rotatie logica stanga*



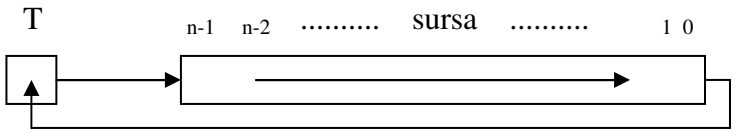

**ral sursa** T,sursa ← $sursa_{n-1:0}$,T *//rotatie aritmetica stanga*




**ror sursa** T,sursa ← $sursa_0$,$sursa_0$,$sursa_{n-1:1}$ *//rotatie logica dr*




**rar sursa** T,sursa ← $sursa_0$,T,$sursa_{n-1:1}$ *//rotatie aritm dreapta*




*Instructiuni pentru transfer control (de ramificatie).*

**jmp adr** CP ← adr *//salt neconditionat, in contorul de program*
*//se incarca adresa de salt*

**jz adr** daca Z=1 atunci CP ← adr *//salt daca Z=1*
altfel CP ← CP+2 *//altfel se incrementeaza CP cu lungimea*
*//instructiunii, aici se presupune lungimea este de 2 cuvinte*

**jnz adr** daca Z=0 atunci CP ← adr *//salt daca Z=0*
altfel CP ← CP+2 *//altfel se incrementeaza CP cu lungimea*
*//instructiunii, aici se presupune lungimea este de 2 cuvinte*

..... *//analog instructiuni de salt conditionat pentru ceilalti*
*//indicatori de conditie*

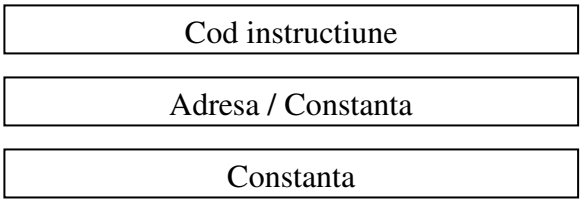
**call adr** IS ← IS – 1 *//apel de procedura, se depune in stiva adresa*
mem(IS) ← CP+2 *// de revenire, iar in CP se incarca*
CP ← adr *//adresa procedurii*

**ret** CP ← mem(IS) *//revenire din procedura, in CP se incarca*
IS ← IS+1 *// adresa de revenire din varful stivei*

**hlt** *//oprirea executiei programului si trecerea in starea HALT*

### Moduri de adresare

Modul de adresare este modalitatea (mecanismul) prin care, pe baza informatiilor continute de instructiunea curenta se poate calcula adresa efectiva pentru accesarea operandului (operanzilor), respectiv pentru stocarea rezultatului. Principalele moduri de adresare, intalnite la majoritatea procesoarelor sunt prezentate pe scurt in continuare. Se presupune un set general de instructiuni, de lungime variabila, fiecare instructiune fiind reprezentata pe 1, 2 sau 3 cuvinte, in care, intotdeauna primul cuvant este codul instructiunii (codifica instructiunea sau operatia), al doilea cuvant, daca exista reprezinta o adresa de memorie sau o valoare constanta, iar al treilea cuvant, daca exista, reprezinta o valoare constanta.



Astfel, se presupune ca un singur operand poate fi in memorie, cand eventual poate reprezenta si destinatia rezultatului.

*Adresarea directa.* Adresa efectiva a operandului este specificata in instructiune, prin al doilea cuvant al instructiunii (fig.4.4.1). Accesul la operand se face conform urmatoarei scheme:

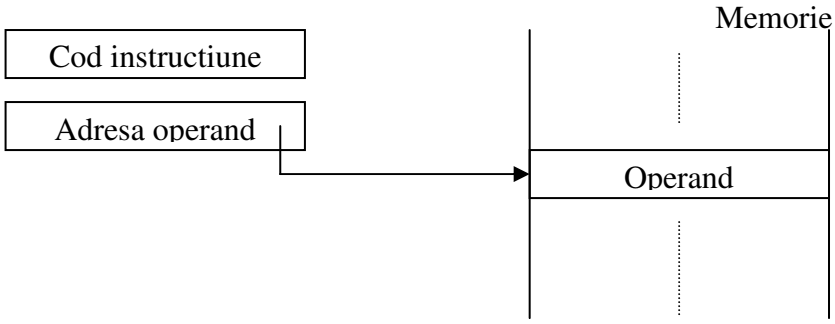



*Fig.4.4.1 Adresarea directa.*

Exemplu de utilizare: mov reg_dest,adresa.

*Adresarea indirecta.* Al doilea cuvant al instructiunii contine adresa adresei operandului, aflat in memorie (fig.4.4.2). Accesul la operand:

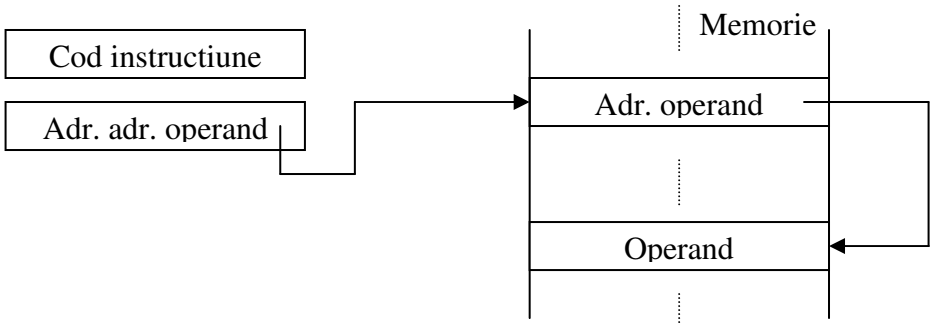



*Fig.4.4.2 Adresarea indirecta.*

Exemplu de utilizare: mov reg_dest,[adresa].

*Adresarea indirecta prin registru.* Un camp din codul instructiunii specifica un registru, care contine adresa operandului aflat in memorie (fig.4.4.3). Schema de acces la operand:

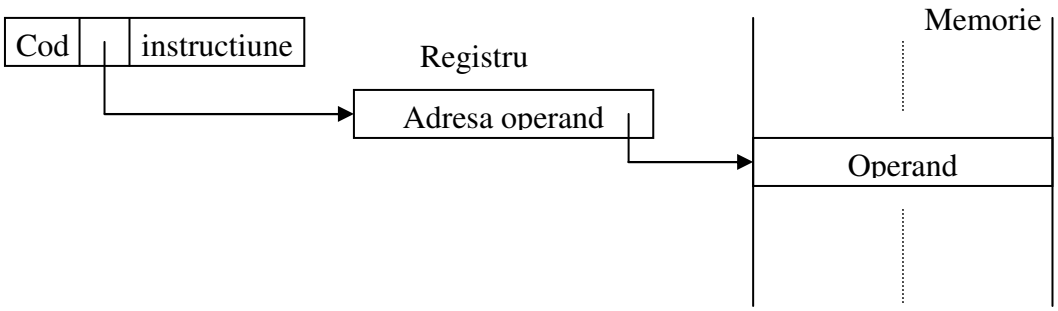



*Fig.4.4.3 Adresarea indirecta prin registru.*

Exemplu de utilizare: mov reg_dest,[reg].

*Adresarea indirecta prin registru cu autoincrementare / autodecrementare.* Acest mod de adresare (fig.4.4.4) este asemanator cu cel precedent, insa registrul utilizat in calcularea adresei efective este incrementat (dupa calculul adresei efective) sau decrementat (inainte de calculul adresei efective). Schema de acces la operand:

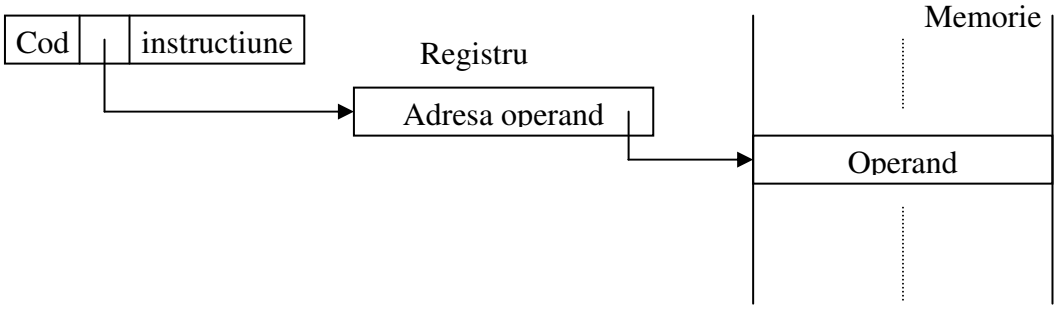



*Fig.4.4.4 Adresarea indirecta prin registru cu autoincrementare / autodecrementare*

Acest mod de adresare este util pentru accesarea componentelor unui vector aflate in locatii succesive din memorie.

Exemplu de utilizare: mov reg_dest,[reg+] sau mov reg_dest,[reg–].

*Adresarea bazata.* Adresa efectiva se obtine din insumarea continutului unui registru de baza al procesorului cu un deplasament de adresa, aflat in al doilea cuvant al instructiunii (fig.4.4.5). Schema de acces la operand:

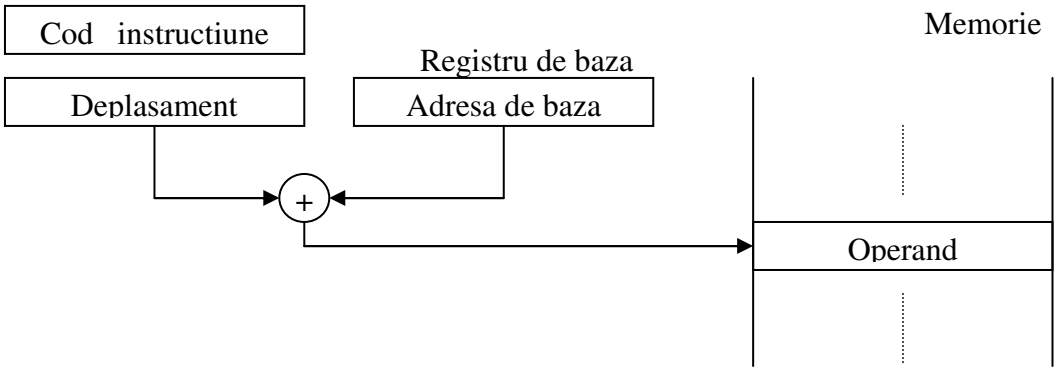



*Fig.4.4.5 Adresarea bazata.*

Exemplu de utilizare: mov reg_dest,[reg_baza+deplasament].

*Adresarea indexata.* Adresa efectiva se obtine din insumarea continutului unui registru index al procesorului cu un deplasament de adresa, aflat in al doilea cuvant al instructiunii (fig.4.4.6). Accesul la operand se realizeaza conform schemei urmatoare:

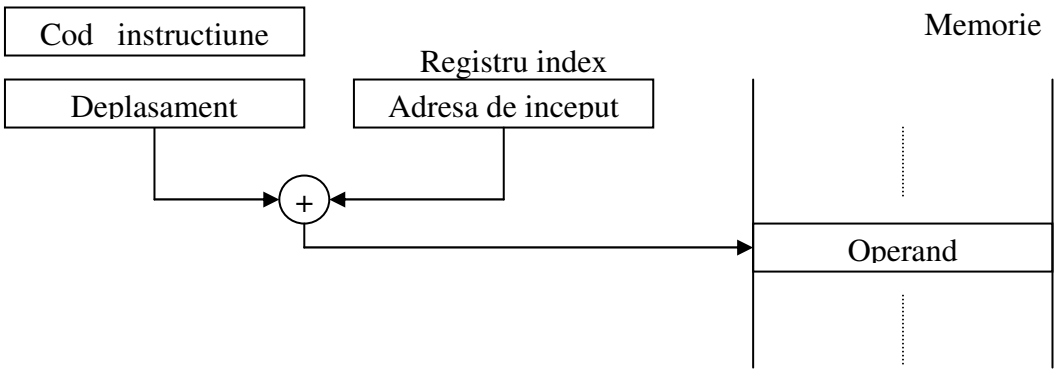



*Fig.4.4.6 Adresarea indexata.*

Principala diferenta fata de modul precedent de adresare este ca la adresarea indexata deplasamentul este reprezentat pe lungimea de biti corespunzatoare spatiului total de memorie, in timp ce la adresarea bazata deplasamentul este reprezentat pe o lungime mai mica de biti, ceea ce permite o reprezentare mai scurta a instructiunilor ce folosesc acest mod de adresare.
Exemplu de utilizare: mov reg_dest,[reg_index+deplasament].

*Adresarea bazata-indexata.* Adresa efectiva se obtine din insumarea continutului unui registru de baza al procesorului, cu un registru index si un

deplasament de adresa, aflat in al doilea cuvant al instructiunii (fig.4.4.7). Schema de acces la operand:

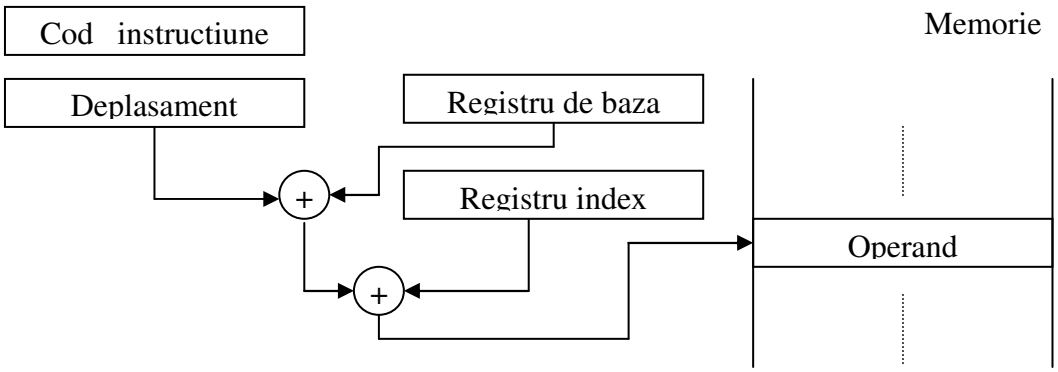



*Fig.4.4.7 Adresarea bazata-indexata.*

Exemplu de utilizare: mov reg_dest,[reg_baza+reg_index+deplasament].

*Adresarea la registru.* Un camp din codul instructiunii specifica un registru, care contine operandul (fig.4.4.8). Schema de acces la operand:

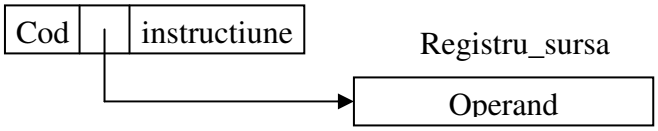



*Fig.4.4.8 Adresarea la registru.*

Exemplu de utilizare: mov reg_dest,reg_sursa.

*Adresarea imediata.* Operandul este specificat in instructiune, prin al doilea cuvant al instructiunii (fig.4.4.9). Accesul la operand se face conform urmatoarei scheme:

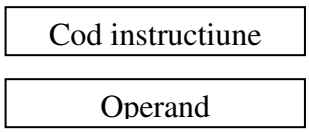



*Fig.4.4.9 Adresarea imediata.*

Exemplu de utilizare: mov reg_dest,valoare_constanta.

**Structura unitatii centrale de prelucrare**

In continuare se va avea in vedere un sistem de calcul elementar, pentru care se va studia structura unitatii centrale de prelucrare si modul de executie a instructiunilor. Schema bloc a unitatii centrale de prelucrare este prezentata in figura urmatoare (fig.4.4.10):

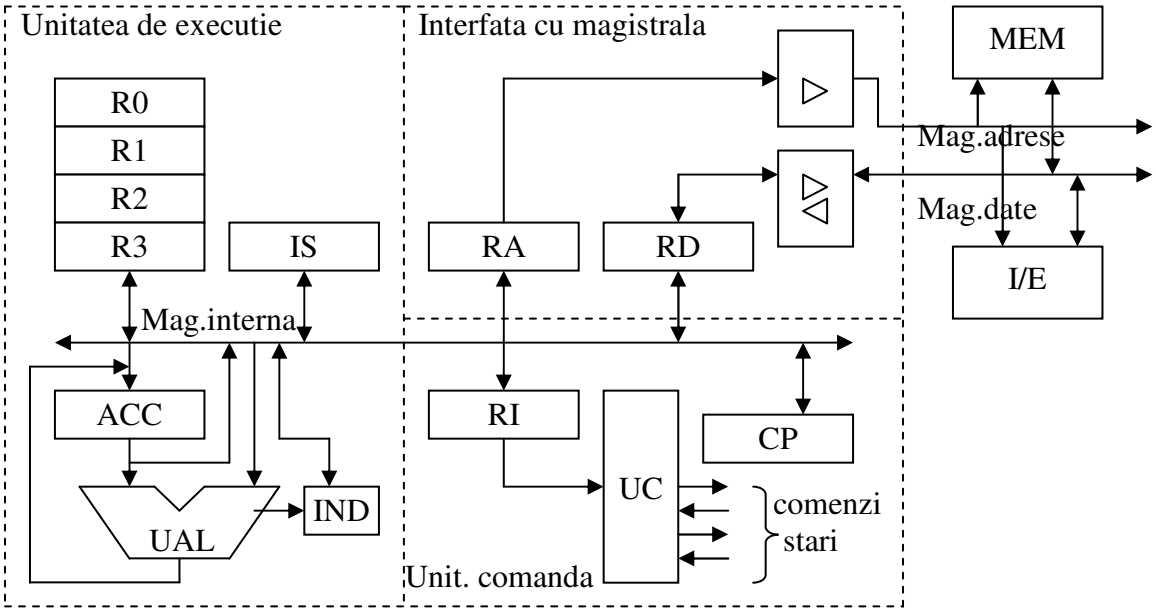



*Fig.4.4.10 Structura unitatii centrale de prelucrare.*

Resursele prezentate in schema bloc sunt urmatoarele:

R0-R3 sunt registre generale de lucru, accesibile prin program.

IS este indicatorul varfului stivei. Stiva este organizata in memorie, sensul de crestere a acesteia fiind catre adrese mici. Intotdeauna IS contine adresa de memorie la care s-a depus ultima valoare in stiva. Astfel, la o depunere in stiva, IS este decrementat si se inscrie valoare la adresa continuta de IS, iar la o extragere din stiva, se citeste cuvantul din memorie de la adresa data de IS, iar apoi IS este incrementat.

RA este registrul de adrese, inaccesibil programatorului prin instructiuni. In registrul RA se incarca o adresa de memorie sau de port de intrare / iesire. Continutul RA este plasat pe magistrala externa de adrese in timpul operatiilor de citire sau scriere cu memoria sau cu un port de intrare / iesire. Conectarea registrului la magistrala externa se face prin intermediul unor circuite tampon, unidirectionale.

RD este registrul de date, de asemenea transparent programatorului. In acest regsitru este incarcat un cuvant de date citit sau scris in cadrul unei operatii cu memoria sau cu un port de intrare / iesire. Conectarea registrului

la magistrala externa de date se face prin intermediul unor circuite tampon bidirectionale.

UAL este unitatea aritmetica logica care efectueaza toate operatiile aritmetice si logice.

ACC este un registru accumulator, la nivelul UAL pentru pastrarea unui operand. Al doilea operand este furnizat direct de pe magistrala interna a procesorului. ACC nu poate fi accesat direct de catre programator.

IND reprezinta registrul indicatorilor de conditie, pozitionati in functie de rezultatele obtinute in UAL.

UC este unitatea de comanda care citeste instructiunile programului si, testand o serie de semnale de stare, lanseaza o succesiune de semnale de comanda care sa permita executia instructiunilor.

RI este registrul de instructiune, in care se incarca primul cuvant al instructiunii, codul instructiunii. Continutul registrului este analizat de unitatea de comanda, care recunoaste instructiunea si o executa.

CP este contorul de program. La inceputul executiei instructiunii curente, CP contine adresa de memorie la care se gaseste primul cuvant al instructiunii (codul). In timpul executiei instructiunii curente continutul CP este actualizat, fie prin incrementare cu 1, 2 sau 3 (in functie de lungimea instructiunii curente) in cazul unei secvente liniare de instructiuni, fie prin incarcare cu o valoare citita sau calculata in cazul instructiunilor de ramificatie.

MEM este memoria interna a calculatorului.

I/E reprezinta dispozitivele de intrare / iesire impreuna cu interfetele lor.

In cadrul procesorului s-au demarcat prin linie intrerupta trei module importante: unitatea de executie, unitatea de interfata cu magistrala si unitatea de comanda.

**Executia instructiunilor**

Executia unei instructiuni necesita patru faze:

-faza de citire a instructiunii din memorie (faza de fetch);

-faza de decodificare a instructiunii (de recunoastere de catre unitatea de comanda a instructiunii);

-faza de citire operanzi (daca instructiunea respectiva necesita operanzi);

-faza de executie propriu-zisa (se executa operatia ceruta de instructiune).

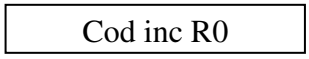
In continuare se vor considera cateva exemple de instructiuni reprezentative pentru categoriile unui set de instructiuni si se va analiza modul lor de executie.

*Instructiuni fara adresa efectiva*. O astfel de instructiune nu contine adresa de operand aflat in memorie. Se reprezinta pe un cuvant (codul instructiunii) sau pe doua cuvinte (primul cuvant codul, iar al doilea cuvant reprezinta o constanta in cazul adresarii imediate).

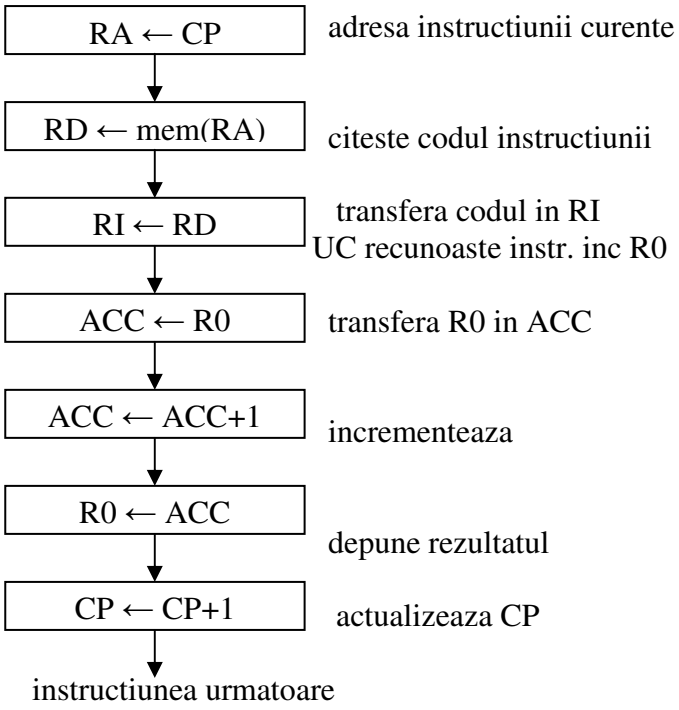
Se va considera ca exemplu instructiunea:

**inc R0**

Reprezentarea instructiunii:




Executia acestei instructiuni este prezentata in organigrama urmatoare:

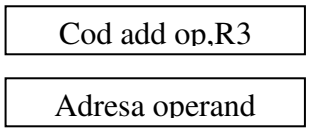



Toate transferurile intre resursele interne ale procesorului se realizeaza prin intermediul magistralei interne.

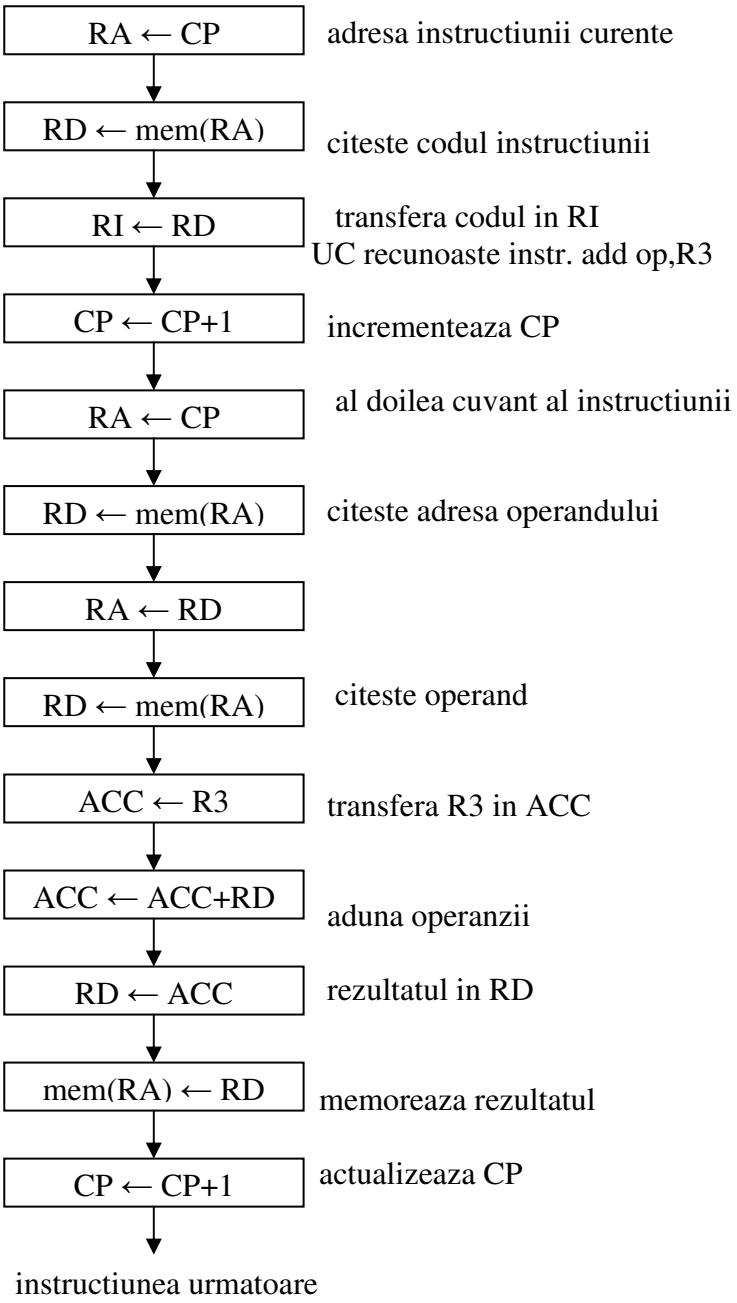
*Instructiuni cu o adresa efectiva*. O astfel de instructiune contine informatii de adresa pentru un operand aflat in memorie. Se reprezinta pe doua cuvinte (codul instructiunii si informatiile de adresa) sau pe trei cuvinte (codul, informatii de adresa si o constanta). Se va considera ca exemplu instructiunea:

**add operand,R3**

unde operand se afla in memorie. Se presupune utilizarea adresarii directe, deci adresa operandului se gaseste in instructiune, fiind reprezentata prin al doilea cuvant al instructiunii. Reprezentarea instructiunii:




Executia acestei instructiuni este prezentata in organigrama urmatoare:



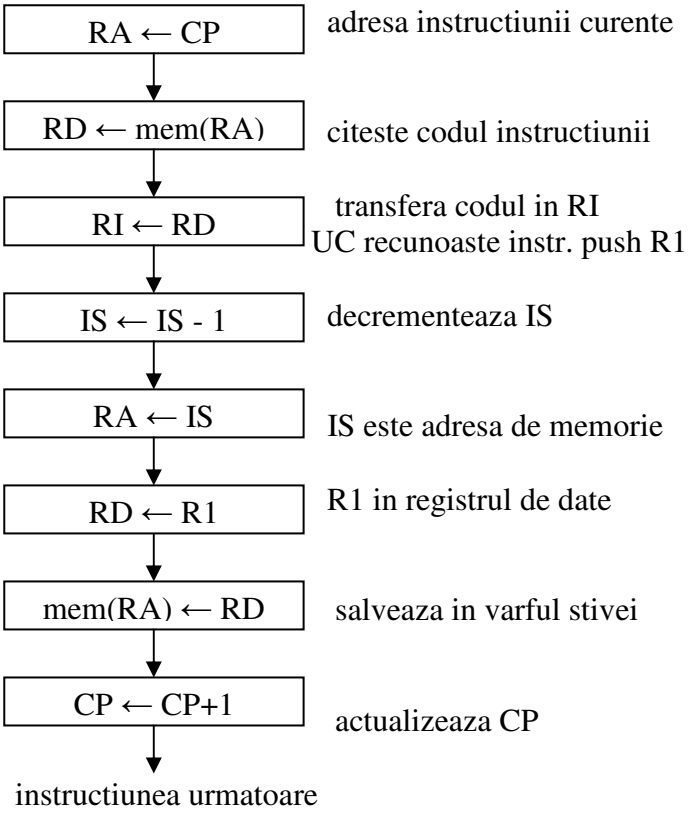
De asemenea, toate transferurile intre resursele interne ale procesorului se realizeaza prin intermediul magistralei interne.

*Instructiuni de depunere in stiva.* Se considera instructiunea:
**push R1**

Reprezentarea instructiunii pe un singur cuvant:

| Cod push R1 |
|---|

Executia instructiunii:

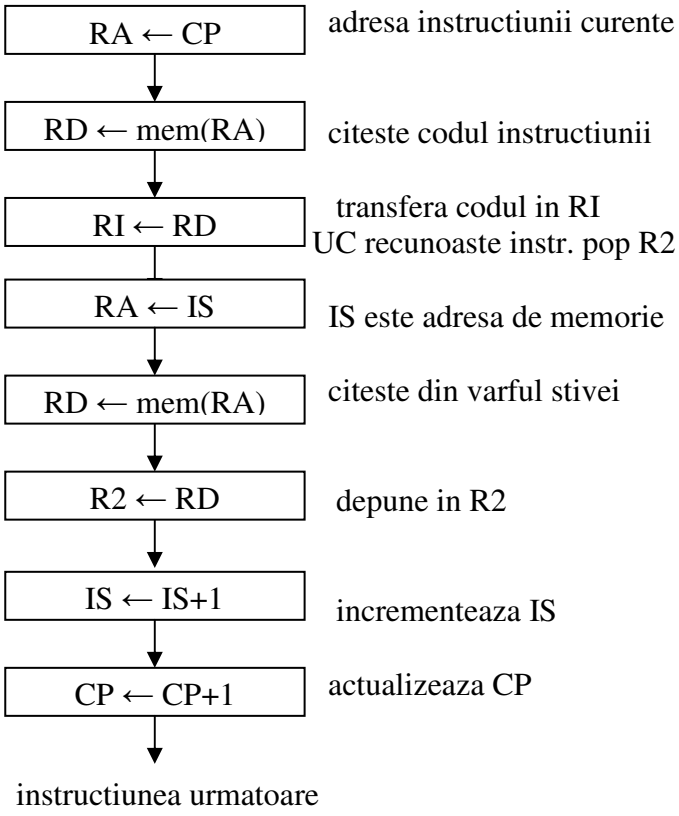



*Instructiuni de extragere din stiva.* Se considera instructiunea:
**pop R2**

Reprezentarea instructiunii pe un singur cuvant:

| Cod pop R2 |
|---|

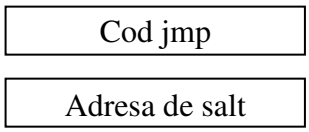
Executia instructiunii:



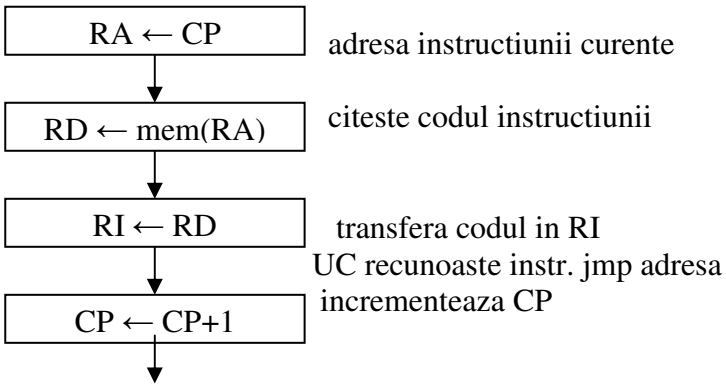

*Instructiuni de salt neconditionat*. O astfel de instructiune contine adresa de salt. Se reprezinta pe doua cuvinte (codul instructiunii si adresa) Se va considera ca exemplu instructiunea:

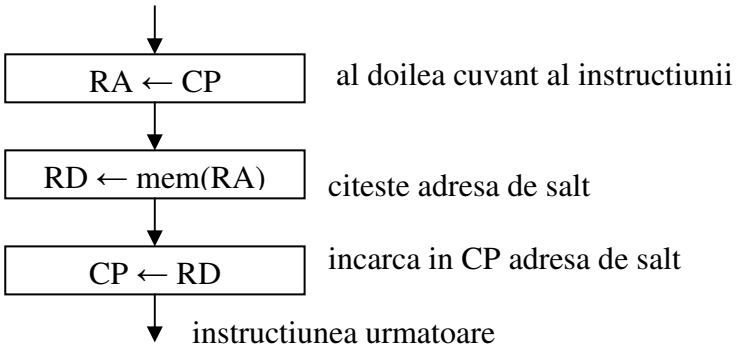
**jmp adresa**

Reprezentarea instructiunii:




Executia acestei instructiuni este prezentata in organigrama urmatoare:

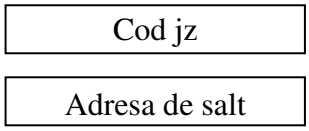




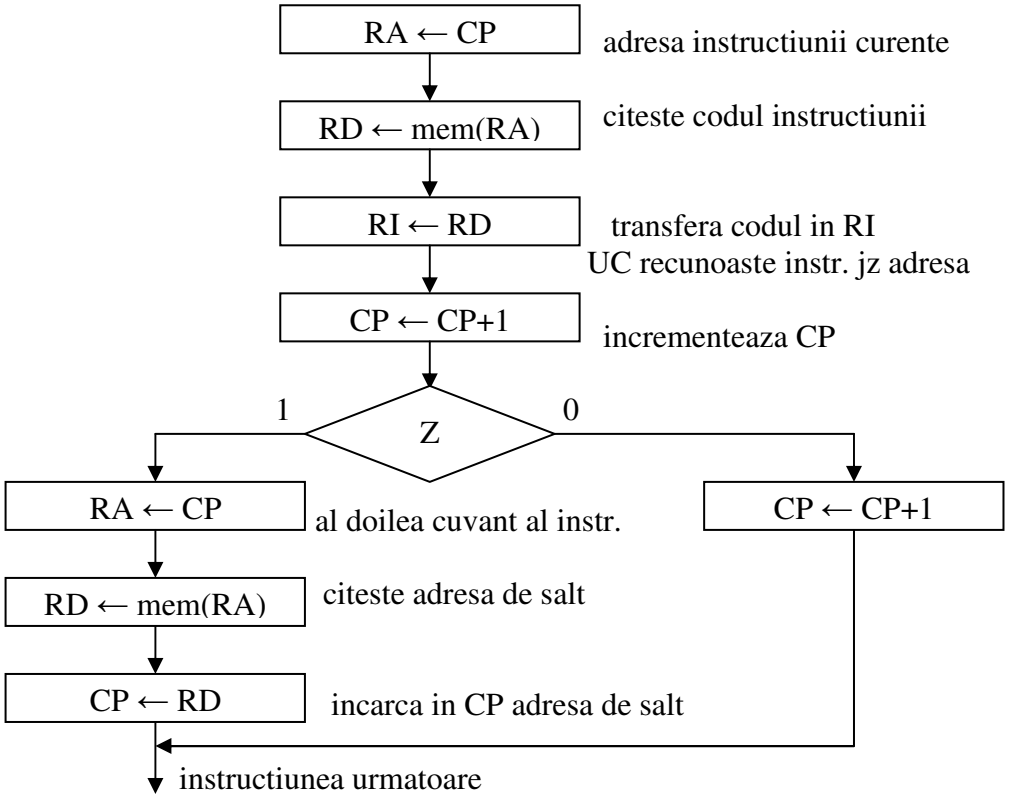

*Instructiuni de salt conditionat*. O astfel de instructiune contine de asemenea adresa de salt. Se reprezinta pe doua cuvinte (codul instructiunii si adresa). Se considera ca exemplu instructiunea:

**jz adresa**

Reprezentarea instructiunii:



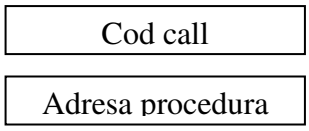

Executia instructiunii:



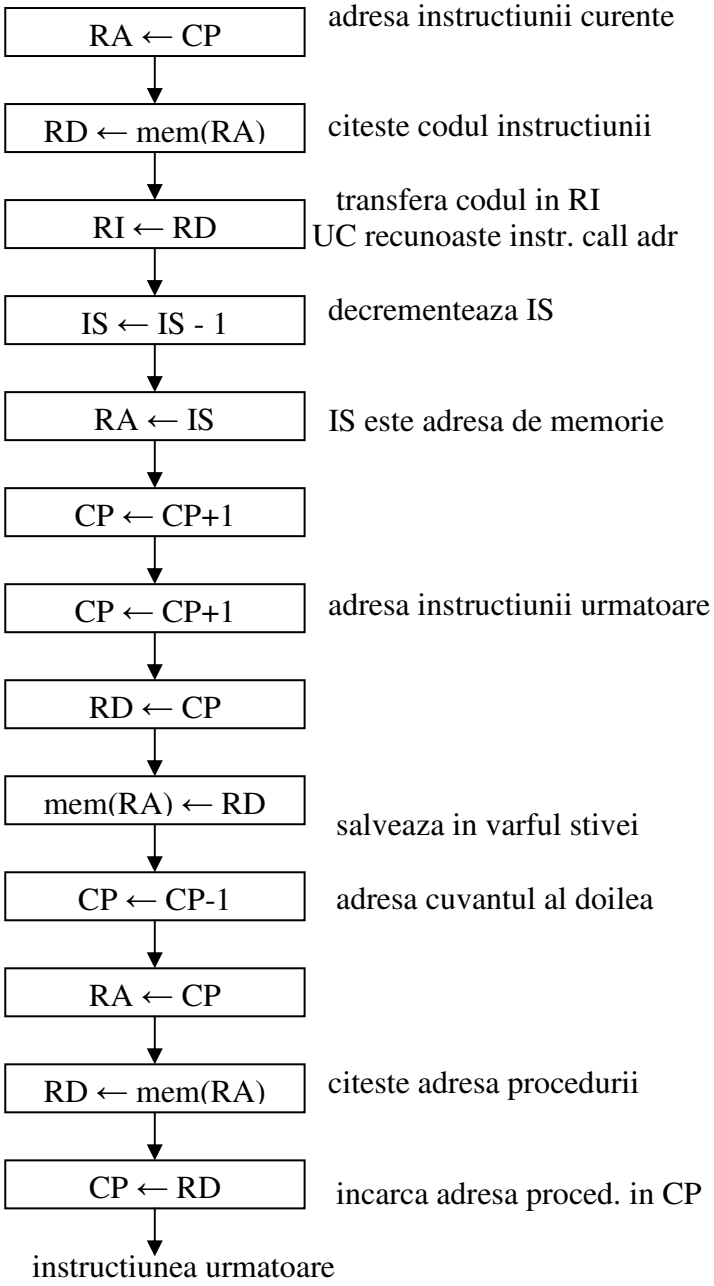
*Instructiuni de apel de procedura.* Se considera instructiunea:

**Call adresa**

Aceasta instructiune apeleaza o procedura de la adresa specificata, depunand in stiva adresa de revenire si incarcand in registrul CP adresa primei instructiuni din procedura. Reprezentarea instructiunii se face pe doua cuvinte:



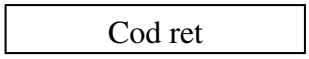

Executia instructiunii:



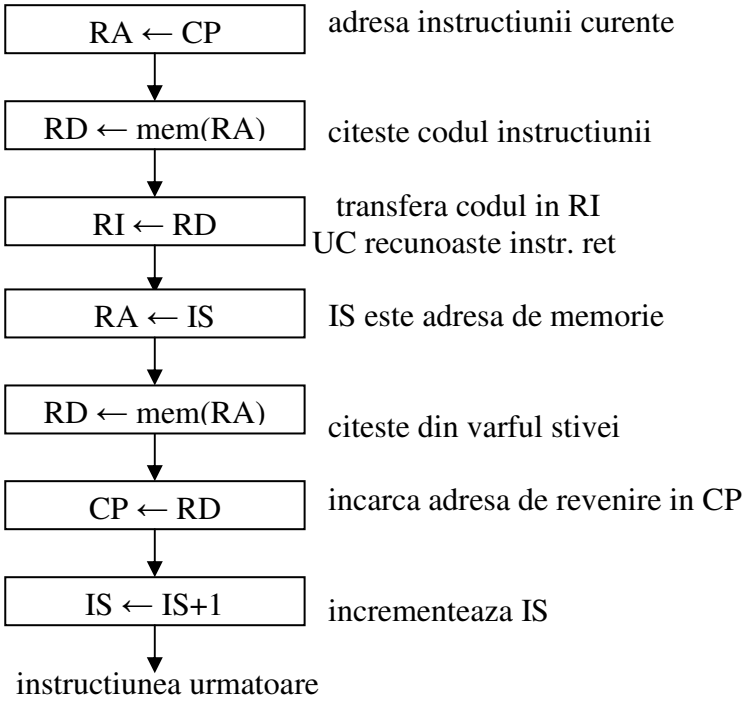
*Instructiuea de revenire din procedura.* Se considera instructiunea:

**ret**

Se reprezinta pe un singur cuvant:

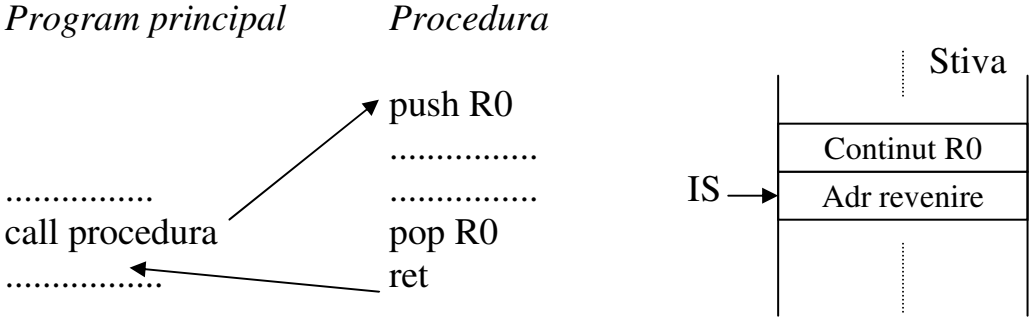



Aceasta instructiune asigura revenirea dintr-o procedura. In varful stivei se gaseste adresa de revenire, care va fi extrasa si incarcata in registrul CP. Executia instructiunii:




Este necesar sa se acorde o deosebita atentie instructiunilor de lucru cu stiva. In continuare sunt date doua exemple:

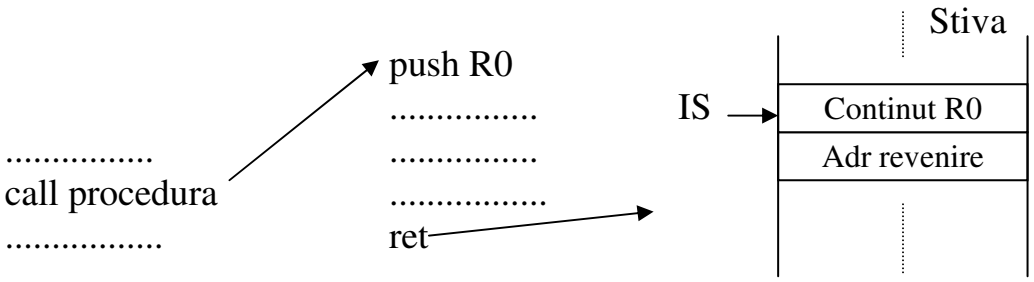
1) Exemplu de utilizare corecta.




La apelul procedurii din programul principal, se incarca in stiva adresa de revenire (adresa instructiunii urmatoare de dupa instructiunea

**call**). Prima instructiune din procedura este **push**, care depune continutul lui R0 in varful stivei. Presupunem ca celelalte instructiuni nu modifica continutul stivei. In incheierea procedurii se executa o instructiune **pop**, care reface continutul lui R0, iar inainte de executia instructiunii **ret**, in varful stivei se gaseste chiar adresa de revenire (locatia indicata de IS in desenul de mai sus). Astfel se realizeaza revenirea corecta in programul care a apelat procedura.

2) Exemplu de utilizare incorecta.




Asemanator cu exemplul precedent, la apelul procedurii din programul principal, se incarca in stiva adresa de revenire (adresa instructiunii urmatoare de dupa instructiunea **call**). Prima instructiune din procedura este **push**, care depune continutul lui R0 in varful stivei. Din nou, presupunem ca celelalte instructiuni nu modifica continutul stivei. In incheierea procedurii nu se mai executa o instructiune **pop**, iar inainte de executia instructiunii **ret**, in varful stivei se gaseste vechiul continut al registrului R0 (locatia indicata de IS in desenul de mai sus). Astfel se ajunge la o adresa oarecare in memorie si se pierde controlul programului.