ARHITECTURA SISTEMELOR DE CALCUL

NOTE DE CURS

Cuprins

1. Reprezentarea datelor in calculator

1.1 Reprezentarea numerelor naturale

Pentru inceput vom reaminti anumite rezultate din aritmetica.

Teorema 1 (Teorema impartirii cu rest)

Pentru orice doua numere intregi a si b, b ≠0, exista si sunt unice numerele intregi q si r, astfel incat:

1. a = bq + r

2. 0 ≤ r ≤ |b|.

Numarul a se va numi deimpartit, b impartitor, q catul si r restul

Teorema 2

Fie un numar natural b ≥ 2. Pentru orice numar natural x, exista si sunt unice numerele naturale n, a_0, a_1, …, a_n *unde* a_0, a_1, …, $a_n \in \{0, 1, 2, \dots, b-1\}$, $a_n \neq 0$, *astfel incat*

$$x = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b + a_0$$

Mai spunem ca reprezentarea lui x in baza b este $a_n a_{n-1} \dots a_1 a_0$

Este evident faptul ca a_0 este restul impartirii lui x la b, iar $a_n b^{n-1} + a_{n-1} b^{n-2} + \dots + a_1$ este catul aceleiasi operatii. In acest fel obtinem urmatorul algoritm de calcul al cifrelor reprezentarii unui numar natural intr-o baza data:

i=0
Cat timp x ≠ 0
a_i este restul impartirii lui x la b
x ia valoarea catului impartirii lui x la b
i = i +1

De exemplu daca dorim sa reprezentam pe 2013 in baza 8, obtinem:

2013 = 8*251 + 5
251 = 8*31 + 3
31 = 8*3 + 7
3 = 8*0 + 3

Deci reprezentarea numarului 2013 in baza 8 este 3735. Remarcam ca acest algoritm furnizeaza cifrele reprezentarii in ordinea inversa a scrierii.

In calculator numerele se reprezinta in baza 2, deci cu cifrele 0 si 1. Aceasta deoarece cifrele sunt memorate cu ajutorul unor circuite elementare care, daca prin ele circula tensiune mica valoarea este 0, si 1 altfel. O cifra in baza 2 o numim *bit.* Unitatea de baza a memoriei este *octetul*, adica 8 biti.

2013 reprezentat in baza 2 este
11111011101.

Se observa ca reprezentarile in baza 2 sunt greu de utilizat de catre om, din cauza numarului extrem de mare de cifre binare. De aceea se utilizeaza baze puteri ale lui 2, de regula 8 sau 16. Daca numarul este scris in baza 2^n, reprezentarea in baza 2 se obtine inlocuind fiecare cifra cu reprezentarea sa in baza 2 pe n biti (adica adaugand eventual zerouri nesemnificative reprezentarii binare). Invers, daca numarul este reprezentat in baza 2, reprezentarea in baza 2^n se obtine impartind numarul de la dreapta catre stanga in grupe de n biti si inlocuind fiecare grupa cu cifra corespunzatoare in baza 2^n.

Exemple folosind reprezentarea numarului 2013

Baza 8

In baza 8, numerele de la 0 la 7 (8-1) pe 3 biti se exprima astfel:

000 – 0
001 - 1
010 – 2
011 – 3
100 – 4
101 – 5
110 – 6
111 - 7

Folosind reprezentarea obtinuta mai sus 3735, reprezentarea in baza 2 este

11 111 011 101

Baza 16

Pentru a reprezenta numerele in baza 16, cu cifre de la 0 la 15, avem nevoie de sase simboluri suplimentare pentru cifrele cuprinse intre 10 si 15. Astfel reprezenterea cifrelor de la 0 la 15 pe 4 biti este urmartoarea

0000 – 0
0001 - 1
0010 – 2
0011 – 3
0100 – 4
0101 – 5

0110 – 6
0111 – 7
1000 – 8
1001 – 9
1010 – 10 = A
1011 – 11 = B
1100 – 12 = C
1101 – 13 = D
1110 – 14 = E
1111 – 15 = F

Astfel, pentru reprezentarea numerelor in baza 16 (hexazecimal) trebuie folosite simbolurile suplimentare (A, B, C, D, E, F)

Pornind de la reprezentarea in baza 2 a numarului 2013 obtinem prin formarea de grupe de 4 biti

111 1101 1101

deci 7DD in baza 16. Deci pornind de la reprezentarea hexazecimala 7DD obtinem

0111 1101 1101

Din acest moment vom reprezenta numerele in baza 16. Subliniem ca fiecare cifra in baza 16 se exprima pe 4 biti (tetrad). Deci reprezentarea externa a continutului unui octet este pe 2 cifre hexazecimale, adica 2 tetrade.

1.2 Reprezentarea numerelor intregi cu semn

O prima idee de reprezentare a numerelor cu semn ar fi formatul semn + valoare absoluta, lucru care inseamna ca sacrificam bitul cel mai semnificativ ca sa memoram semnul. Acest lucru determina un inconvenient major, de exemplu, la adunarea numerelor cu semn, adica, daca numerele au acelasi semn, se aduna valorile absolute, iar semnul rezultatului este

semnul comun al celor doua numere. Insa, daca numerele sunt de semn contrar, in functie de ordinea valorilor absolute, trebuie efectuata o operatie de scadere, urmata de stabilirea semnului rezultatului, etc

In calculator, numerele cu semn se reprezinta in complement fata de 2. Pe n biti se pot represent numerele de la 0 la 2^n-1. Cu semn, pe n biti, vom reprezenta numerele de la -2^{n-1} la $2^{n-1}-1$. Complementul fata de 2 (opusul numarului) se poate obtine in doua moduri:

1. Negand toti bitii (obtinand astfel complementul fata de 1) si adaugand la rezultat 1. Aceasta metoda este data in marea parte a referintelor
2. Scazand numarul din 0 (zero) (Metoda recomandata de autor)

Exemplu Numarul -2013

Cum s-a vazut mai sus, numarul 2013 se exprima pe 16 biti (2*4=8 tetrade) sub forma 07DD. Aplicand oricare dintre cele doua metode sugerate mai sus obtinem

F823

Atentie! Daca expandarea unui numar pozitiv pe n biti se face in mod natural prin adaugarea la stanga de cifre 0 nesemnificative, expandarea numarului negativ se face prin adaugarea de cifre 1 nesemnificative, Astfel reprezentarea lui 2013 pe 32 de biti va fi

00 00 07 DD

Pe cand reprezentarea lui 2013 pe 32 de biti va fi

FF FF F8 23

Avantajul reprezentarii numerelor cu semn in format complement fata de 2 este acela ca la operatii de adunare si

scadere, nu mai trebuie tinut cont de semn. Pur si simplu se face operatia in baza 2, rezultand, eventual, un transport la sfarsitul operatiei.

1.3 Reprezentarea numerelor fractionare

Se stie ca orice numar real se descompune in mod unic ca suma dintre un numar intreg (numit si *partea sa intreaga*) si in numar apartinand intervalului [0, 1) (numit si *partea sa fractionara*). Cum din sectiunea 1.2 stim sa reprezentam orice intreg intr-o baza data, ramane sa vedem cum reprezentam numerele positive subunitare.

Teorema 3.

Fie un numar natural b ≥ 2. Pentru orice numar $x \in (0,1)$, *exista numerele naturale* $a_1, a_2, \ldots, a_n, \ldots$ *unde* $a_1, a_2, \ldots,$ $a_n \in \{0,1,2,\ldots,b-1\}$, *astfel incat*

$$x = \frac{a_1}{b} + \frac{a_2}{b^2} + \cdots + \frac{a_n}{b^n} + \cdots$$

Mai spunem ca reprezentarea lui x in baza b este $0, a_1 a_2 \ldots a_n \ldots$ *(Asadar reprezentarea numerelor fractionare se face sub forma de serie si nu de suma finita ca la numere intregi)*

Reprezentarea este unica mai putin situatiile $0, a_1 a_2 \ldots a_n (b-1)(b-1)(b-1) \ldots = 0, a_1 a_2 \ldots (a_n+1)000 \ldots$ *unde* $a_n \leq b-2$. *(Deci doar numerele fractionare cu reprezentare finita pot avea doua reprezentari)*

Demonstratie

Folosind formula binecunoscuta

$$1 + q + q^2 + \cdots q^{n-1} = \frac{q^n - 1}{q - 1}$$

obtinem ca seria $1 + q + q^2 + \cdots q^{n-1} + \cdots$ converge catre $\frac{1}{1-q}$ pentru $q \in (0,1)$. (1)

Seria $\frac{a_1}{b} + \frac{a_2}{b^2} + \cdots + \frac{a_n}{b^n} + \cdots$ din enunt se majoreaza la seria $\frac{b-1}{b} + \frac{b-1}{b^2} + \cdots + \frac{b-1}{b^n} + \cdots$ care converge catre 1. Deci $a_1$este partea intreaga a numarului bx, iar $\frac{a_2}{b} + \cdots + \frac{a_n}{b^{n-1}} + \cdots$ este partea sa fractionara, daca nu totate cifrele a_i sunt egale cu b-1.

Din cele de mai sus deducem urmatorul algoritm de calcul al cifrelor reprezentarii, si in plus de ce putem avea doua reprezentari distincte in cazurile particulare date in enunt.

$$\boldsymbol{x_0 = x}$$

$$\boldsymbol{i = 1}$$

Cat timp “adevarat”

a_i este partea intreaga a lui bx_{i-1}

x_i este partea fractionara a lui bx_{i-1}

Ramane sa demonstram ca seria astfel construita converge intr-adevar catre .

Avand in vedere ca $a_i = bx_{i-1} - x_i$, obtinem ca

$$\frac{a_i}{b^i} = \frac{x_{i-1}}{b^{i-1}} - \frac{x_i}{b^i}$$

Sumand aceste egalitati pentru $i \in \{1, \dots, n\}$ obtinem ca

$$\frac{a_1}{b} + \frac{a_2}{b^2} + \cdots + \frac{a_n}{b^n} = x - \frac{x_n}{b^n}$$

deci

$\left| x - \left(\frac{a_1}{b} + \frac{a_2}{b^2} + \cdots + \frac{a_n}{b^n}\right)\right| = \frac{x_n}{b^n} \leq \frac{b-1}{b^n}$, care converge catre zero si, conform criteriului majorarii, demonstratia este incheiata.

Asadar, indiferent cati biti folosim pentru reprezentarea unui numar fractionar, nu avem nici o sansa de a memora toate cifrele sale. Totusi exista situatii in care putem sti toate cifrele.

Teorema 4

Fie un numar natural b ≥ 2. Un numar $x \in (0,1)$*, are reprezentarea in baza b periodica daca si numai daca este rational*

Demonstratie

"daca" Presupunem $x = \frac{p}{q}$ cu $p, q \in N, q \geq 2$. Folosind algoritmul din demonstratia Teoremei 2, obtinem ca toate partile fractionare x_i sunt fractii subunitare cu numitorul q, deci numitorii pot lua un numar finit de valori. In consecinta exista i, j diferite intre ele, astfel incat $x_i = x_j$. Deci procedeul este periodic o data cu repetarea a doua parti fractionare.

"numai daca" Grupand termenii din reprezentare dupa perioada obtinem o serie geometrica cu numere rationale si deci suma sa este un numar rational conform (1).

Demonstratia este incheiata.

Asadar pentru un numar rational putem spune cat este orice zecimala pentru un index dat, dar, in continuare, nu putem scrie toate zecimalele acestuia. In continuare caracterizam numerele care se pot exprima cu un numar finit de zecimale

.

Teorema 5

Fie un numar natural b ≥ 2. Un numar rational $\frac{p}{q} \in (0,1)$*, scris in forma ireductiblia (de data aceasta!) are reprezentarea in baza b finita daca si numai daca orice numar prim care divide pe*

q il divide si pe b (cu alte cuvinte q nu poate sa contina in descompunerea sa in factori primi decat factori primi care apar in descompunerea bazei)

Demonstratie

"daca" Prin amplificare cu un numar adecvat putem transforma fractia intr-una cu numitor putere a lui b. Scriind numitorul in baza b ca numar intreg si descompunand in fractii simple, obtinem o reprezentare in baza b finita.

"numai daca" Efectuam calculele si obtinem o fractie cu numitorul putere a lui b. Aceasta fractie se va aduce la forma ireductibila prin eventuale simplificari.

Demonstratia este incheiata.

Atragem in mod expres atentia ca un numar poate avea reprezentare finita intr-o baza dar nu si in alta. Exemplu $\frac{1}{10}$ in baza 10 se scrie ca 0,1 dar in baza 2 are reprezentare infinta din cauza factorului prim care divide pe 10 dar nu si pe 2.

Reprezentarea numerelor fractionare in calculator (bineinteles in baza 2) se poate realize astfel

1. Virgula fixa, folosind n octeti pemtru partea intreaga si m octeti pentru partea fractionara (desigur trunchiata eventual)

Aceasta reprezentare nu este recomandata si nici folosita de sistemele de calcul. Reprezentarea folosita uzual este urmatoarea;

2. Virgula mobila. Se scrie numarul in baza 2 si se omite virgula care separa partea intreaga de partea fractionara. Ne oprim la primul bit egal cu 1 din stanga si omitem bitii de 0 din stanga. Acest prim bit semnificativ il consideram

primul bit dupa virgula. Numarul obtinut il notam cu *m* si il vom numi *mantisa.* Numarul va fi $2^e m$ unde *e* este *exponentul*. Astfel daca exponentul *e* este pozitiv, virgula se va muta cu *e* pozitii la dreapta mantisei, rezultand astfel un numar spraunitar, iar daca exponentul *e* este negativ, virgula se va muta cu *e* pozitii la stanga mantisei, rezultand astfel un numar subunitar.

Forma de reprezentare in virgula mobile este dependent de platforma. Vom da ca exemplu standardul IEEE 754.

In simpla precizie, numerele se reprezinta pe 32 de biti si anume:

1 bit pentru semn

8 biti pentru exponent

23 de biti pentru mantisa in care se omite prima cifra de 1 semnificativa, care este implicita.

In dubla precizie, numerele se reprezinta pe 64 de biti si anume:

1 bit pentru semn

11 biti pentru exponent

52 de biti pentru mantisa in care se omite prima cifra de 1 semnificativa, care este implicita.

Exista anumite reprezentari particulare cum ar fi aceea ca numarul 0.0 se reprezinta cu toti bitii egali cu zero. Mai sunt si alte cazuri particulare cum ar fi plus/minus infinit, Nan (not a number), etc, dar nu insistam asupra acestora.

1.4 Reprezentarea caracterelor

La ora actuala cea mai utilizata forma de reprezentare a caracterelor este codul ASCII. In aceasta codificare fiecare caracter este reprezentat pe un octet folosind 7 biti (deci bitul cel mai semnificativ este totdeauna egal cu 0). Valorile in

hexazecimal sunt cuprinse intre 0 si 7F. Enumeram “pietrele de hotar” in reprezentarea ASCII:

- 0 – 1F caractere neimprimabile (tastate prin CTRL + “ceva” sau prin taste speciale gen F1, F2, etc)
- 20 blanc (primul caracter imprimabil)
- 21 – 2F semne de punctuatie
- 30 ‘0’ (caracterul corespunzator cifrei zero)
- 31 ‘1’ (caracterul corespunzator cifrei unu)
- ----------
- 39 ‘9’ (caracterul corespunzator cifrei noua)
- 3A - 40 alte semne de punctuatie
- 41 ‘A’
- 42 ‘B’
- ----------
- 5A ‘Z’
- 5B – 60 alte semne de punctuatie
- 61 ‘a’
- 62 ‘b’
- ----------
- 7A ‘z’
- 7B – 7E alte semne de punctuatie
- 7F backspace (neimorimabil)

  Ulterior, reprezentarea ASCII a fost extinsa:

  1. Folosind si bitul cel mai semnificativ din octet. In acest mod la cele 128 caractere utilizabile s-au mai adaugat inca 128
  2. Expanand reprezentarile pe doi octeti. In acest fel s-au putut reprezenta, de exemplu, si caractere din alte alfabete decat cel latin.

1.5 Alte reprezentari ale numerelor intregi fara semn

Numerele intregi fara semn se pot reprezenta si altfel :

a. In format zecimal impachetat (BCD). Fiecare cifra zecimala se va exprima pe 4 biti in format binar (0..9). In acest fel nu se pot folosi cifrele hexazecimale A…F, De exemplu numarul exprimat in baza 10 2013 va fi reprezentat pe doi octeti astfel:
   0010 0000 0001 0011
   Acelasi numar scris in baza 2 va fi reprezentat astfel
   0000 0111 1101 1101
b. In format ASCII sau zecimal despachetat. Fiecare cifra zecimala va fi reprezentata pe un octet ce reprezinta codul zecimal al cifrei. Astfel numarul 2013 va fi reprezentat pe 4 octeti astfel
   00000010 00000000 00000001 00000011

2. Principiile arhitecturii calculatoarelor

Conform conceptiei lui Andrew Tannenbaum, arhitectura calculatoarelor este organizata pe cinci nivele ierarhice, si anume:

1. Nivelul fizic
2. Nivelul microprogramat

---

3. Nivelul limbaj de asamblare
4. Nivelul sistem de operare
5. Nivelul aplicatie

Primele doua nivele sunt de natura hardware, iar celelalte de natura software. Trebuie sa tinem cont ca acest material este destinat unor student care au drept scop sa devina profesionisti ca programatori. Utilizatorii normali, fie cat sunt programatori sau nu, se afla pe nivelul 5, adica aplicatie. Dansii comunica direct cu

nivelul 4, sistem de operare. Din acest motiv, sistemul de operare reprezinta subiectul unui curs separate. Prezentul material prezinta primele 3 nivele, mai precis ce trebuie sa stie un programator despre acestea, deoarece sistemul de operare “are grija” ca aceste lucruri sa fi transparente utilizatorilor.

3. Nivelul fizic

3.1 Componentele principale ale sistemelor de calcul

Conform conceptiei lui von Neumann (cca 1940), sistemele de calcul au trei componente principale

1. unitate centrala de procesare (CPU)
2. memoria
3. perifericele

Facand o paralela cu creierul uman CPU ar fi materia cenusie (cea care gandeste), memoria ar fi materia alba (cea care stocheaza date), iar perifericele organele de simt (care servesc la comunicarea cu exteriorul). In gandirea lui von Neumann, perifericele comunica direct cu CPU. Totusi, schema gandita teoretic de von Neumann in anii 40 a evoluat mult de atunci.

CPU are drept componenta principala *procesorul* care are o (micro) memorie alcatuita din *registrii.* Fiecare tip de procesor (INTEL, MOTOROLA, RISC etc) poseda propriul sau set de registrii. Totusi, remarcam faptul ca orice processor trebuie sa contina cel putin doi registrii cu functionalitati speciale pe care ii vom numi generic *sp* si *pc*:

- *sp* numit si pointer de stiva (stack pointer) memoreaza adresa unui octet din memorie care reprezinta varful stivei. De cate ori se pune un element in stiva *sp* se

decrementeaze, iar cand se scoate un obiect din stiva *sp* se incrementeaza

- *pc* numit si contor de program (program counter) memoreaza adresa octetului din memorie care contine codul urmatoarei instructiuni ce trebuie executate. (**Acest punct va fi clarificat ulterior)**

Memoria este alcatuita din octeti, fiecare avand adresa proprie. Memoriile sunt de doua tipuri:

- RAM (Random Access Memory) ale carei octeti pot fi cititi si scrisi
- ROM (Read Only Memory) ale carei octeti pot fi doar cititi, dar nu si modificati. In principiu, continutul acestor memorii poate fi scris doar in fabrica cu ajutorul unor dispozitive speciale. Totusi, tehnologiile actuale permit inscrierea memoriilor ROM de catre CPU prin niste instructiuni special (care dureaza mai mult decat modificarea unei locatii normale de RAM)

La punerea sub tensiune, memoria RAM si registrii CPU au valori aleatoare. Exceptie face continutul registrului *pc* din motive lesne de inteles. Adresa memorata de *pc* la punerea sub tensiune este numita si adresa de *boot strap.* La aceasta adresa se afla o memorie ROM, astfel incat la fiecare pornire a sistemului se executa aceleasi instructiuni. Ultimele instructiuni memorate in ROM determina incarcarea codului obiect al sistemului de operare in RAM, pentru ca, ulterior, sa se execute instructiuni din acest program.

La randul lor, memoriile RAM poti fi:

- RAM dinamic (care necesita operatia de *refresh)*

- RAM CMOS. Continutul acestor memorii poate fi salvat pe baterie, astfel incat la repornirea sistemului sa regasim aceleasi date ca la oprire. Bineinteles ca daca bateria este descarcata, se pierde continutul memoriei.

2.2 Periferice

De regula, perifericele comunica direct cu procesorul, desi prin anumte tehnici speciale (DMA de exemplu) pot comunica direct cu memoria, adica sa scrie date direct in memorie si sa citeasca date direct din memorie.

Dintr-un punct de vedere, perifericele se impart in doua categorii:

- Comunicabile cu operatorul uman (human readable). De exemplu tastatura, ecranul, imprimanta
- Comunicabile cu alte sisteme de calcul (machine readeable). De exemplu hard disk, modem.

2.3 Sinteza circuitelor

Presupunem cunoscute definitia algebrei booleene si a regulilor de calcul in acestea. In acest context, noi vom lucra numai cu algebra booleeana bivalenta $B_2 = \{0, 1\}$, desi multe rezultate sunt adevarate in algebre booleene oarecari, nu neaparat bivalente. Vom nota xy conjunctia celor doua variabile. Mai mult vom nota cu x^0 negatul lui x.

Un circuit cu n intrari si m iesiri este o functie $f: B_2^n \to B_2^m$, adica $f = (f_1, f_2, \dots, f_m)$, unde $f_i: B_2^n \to B_2$.

Definitie

O functie $f: B_2^n \to B_2$ se numeste *booleana* daca $f(x_1, x_2, \dots, x_n) = V_{c_1,c_2,\dots c_n \in \{0,1\}} f(c_1, c_2, \dots, c_n) x_1^{c_1} x_2^{c_2} \dots x_n^{c_n}$, deci o functie booleeana este o disjunctie de maxim 2^n determinata in mod unic de valorile pe care la ia pentru {0, 1}. Se vor omite

termenii pentru care $f(c_1, c_2, \dots, c_n) = 0$. Aceasta este forma normala *disjunctiva.* Aplcand legile lui De Morgan putem obtine si forma normala *conjunctiva.*

Bineinteles, formele normale pot fi reduse la expresii mai simple utilizand regulile de calcul in algebra booleene.

Scopul nostru este sa scriem forma normala conjunctiva a unei functii booleene pornind de la tabelul sau de valori.

Sunt 2^n valori posiblie ale variabilelor. Pentru orice combinatie care are valoarea 1, se va scrie termenul corespunzator, mai precis conjunctiva tuturor variabilelor, dar negand variabilele cu valoarea 0.

Exemplu

x, y, z	$f(x, y, z)$
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	1

Deci functia este $f(x, y, z) = xy^{'}z\ V\ xyz^{'}\ V\ xyz$, care se mai poate scrie $xy^{'}z\ V\ xyz\ V\ xyz^{'}\ V\ xyz = xz(y\ V\ y^{'})V\ xy\ (z\ V\ z^{'}) = xy\ V\ xz$

Dupa operatia de obtinere a functiei booleene in forma normala, urmeaza sinteza circuitului folosind portile elementare NOT, AND si OR.

Ne vom limita la a obtine functia boleeana in forma normala, fara a desena circuitele folosind simbolurile consecrate pentru porti. Aceasta este treaba “hardistilor”. La Politehnica (Facultatile

de Electrotehnica, Electronica, Automatica) acest pas este esential.

Exemplificam sinteza unui sumator pe un bit. La o prima vedere acest circuit ar avea doua intrari. Dar de fapt are trei, caci ar putea interveni un transport de la o adunare anterioara. Numarul de iesiri este doi, mai precis rezultatul adunarii si al cifrei binare de transport

A	B	T_i	R	T_e
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Deci $R(A,B,T_i) = A^{'}B^{'}T_i \; V \; A^{'}BT^{'}{}_i \; V \; AB'T'{}_i \; V \; ABT_i$ si $T_e(A,B,T_i) = A^{'}BT_i \; V \; AB'T_i \; V \; ABT'{}_i \; V \; ABT_i$

Un sumator pe n biti se realizeaza legand in serie sumatoare pe un bit , transportul de iesire a fiecarui circuit fiind legat la transortul de intrare al circuituiul urmator

3. Nivelul microprogramat

Este vorba de instructiunile cablate, mai precis de ce “stie” sa faca procesorul. De exemplu microprocesorul INTEL 8080 pe 8 biti “stia” sa faca drept operatii aritmetice doar adunari si scaderi pe 8 biti. In acest fel pentru realizarea unei inmultiri trebuiau scrise instructiuni care sa realizeze aceasta operatie, folosind doar adunari si scaderi.

Microprocesorul 8086, pe 16 biti are cablate, in plus, si instructiuni de inmultire si impartire.

Abandonam aici acest nivel, caci este strans legat de sinteza circuitelor. Vom prezenta in continuare un limbaj de asamblare, mai precis cel al microprocesorului INTEL 8088, care, desi este destul de vechi, este suficient de reprezentativ pentru a intelege filozofia limbajelor de asamblare.

4. Nivelul limbaj de asamblare

Realizarea oricarei aplicatii utilizand un limbaj de programare (C, PASCAL, FORTRAN, COBOL) incepe cu scrierea unui fisier text, numit fisierul sursa. Acest fisier este citibil de catre om (human readable). Pornind de la textul sursa, compilatorul va genera un text in limbaj de asamblare, urmat de generarea *codului obiect,* care de data aceasta nu mai este citibil de catre utilizatorul uman. Codul obiect este o succesiune de instructiuni in cod masina care vor fi executate pe rand de catre processor. Reamintim ca fiecare procesor are codurile obiect proprii (instructiunile cablate)

Este evident ca orice aplicatie poate fi scrisa direct in limbaj de asamblare. Insa aceasta are doua inconveniente majore:

- Productivitate scazuta. O instructiune scrisa in limbaj de nivel inalt va fi tradusa de catre compilator intr-un numar destul de mare de instructiuni asamblor
- Programele scrise in asamblor nu sunt portabile. Sa presupunem ca am scris un program intr-un limbaj de nivel inalt, de exemplu C, utilizand un sistem de calcul prevazut cu microprocesor INTEL x86. Daca vrem ca palicatia sa fie executata de alt sistem de calcul, prevazut cu microprocesor MOTOROLA, vom compila

textul sursa C pe a doua platforma, obtinand astfel codul obiect pentru MOTOROLA. Daca aplicatia este scrisa direct in limbaj de asamblare, atunci trebuie rescrisa **complet** pentru a doua platforma, deoarece cele doua procesoare poseda seturi de registrii si instructiuni cablate total diferite

Procesorul nu “stie” sa execute dacat coduri obiect (instructiuni cablate). Fiecare instructiune cablata are codul sau binar propriu. In memorie fiecare instructiune ocupa un numar variabil de octeti, in functie de cate argumente are.

Este absurd sa cerem unui programator in limbaj de asamblare, sa cunoasca pe dinafara codurile binare ale tuturor instructiunilor. De aceea realizarea unui program in limbaj de asamblare incepe cu scrierea unui fisier text, folosind reprezentari literale ale instructiunilor, numite *mnemonice.* Programul asamblor va traduce acest fisier text in coduri obiect.

Continuam cu prezentarea limbajului de asamblare al microprocesorului INTEL 8088.

4.1 Registrii microprocesorului INTEL 8088

Toti registrii acestui microprocesor au cate doi octeti (16 biti). Dam in continuare reprezentarea lor sub forma de simboluri, precum si functionalitatile lor uzuale.

- AX care se imparte in doi subregistrii de cate un octet (8 biti) AL (partea cea mai putin semnificativa) si AH (partea cea mai semnificativa). Acest registru mai este numit si *acumulatorul principal.*
- BX care se imparte in doi subregistrii de cate un octet (8 biti) BL (partea cea mai putin semnificativa) si BH (partea cea mai semnificativa). Acest registru este

folosit drept pointer, adica contine, de regula, adrese din memorie.

- CX care se imparte in doi subregistrii de cate un octet (8 biti) CL (partea cea mai putin semnificativa) si CH (partea cea mai semnificativa). Acest registru este folosit drept contor de numar de iteratii intr-o bucla.
- DX care se imparte in doi subregistrii de cate un octet (8 biti) DL (partea cea mai putin semnificativa) si DH (partea cea mai semnificativa). Acest registru este folosit pentru expandarea acumulatorului pe patru octeti, necesara la operatiile de inmultire si impartire.
- SI (source index) pointer pentru zona de memorie ce reprezinta sursa unei operatii pe siruri de octeti.
- DI (destination index) pointer pentru zona de memorie ce reprezinta destinatia unei operatii pe siruri de octeti.
- BP (base pointer) contine adresa de baza a stivei
- SP (stack pointer) contine adresa varfului stivei
  Urmeaza registrii de segment, care vor fi explicate ulterior
- CS - code segment
- DS - data segment
- SS - stack segment
- ES - extra segment
  Urmeaza doi registrii speciali
- IP – contorul de program. Contine adresa unde este memorat codul instructiunii ce urmeaza a fi executata. Dupa executarea acestei instructiuni, continutul acestui registru se modifica automat cu adresa codului instructiunii urmatoare

- FLAGS – indicatorii de conditie. Bitii acestui registru vor descrie felul in care s-a executat urmatoarea operatie aritmetica. Enumeram mai jos indicatorii de conditie, pozitia lor in registru, precum si semnificatia lor.

- Bitul 0 – CARRY (notat si C). Valoarea 1 a acestui bit indica faptul ca operatia aritmetica s-a incheiat cu un transport. Exemplu: 55 + 55 = AA (fara transport), AA + AA = 54 (transport 1)
- Bitul 2 – PARITY (notat si P). Valoarea 1 a acestui bit indica faptul ca rezultatul operatiei aritmetice are un numar par de biti cu valoarea 1
- Bitul 4 – HALF CARRY (notat si A). Valoarea 1 a acestui bit indica faptul ca a avut loc un transport intre parte ace mai putin semnificativa si partea cea mai semnificativa
- Bitul 6 – ZERO (notat si Z). Valoarea 1 a acestui bit indica faptul ca rezultatul operatiei este zero.
- Bitul 7 – SIGN (notat si S). Valoarea 1 a acestui bit indica faptul ca rezultatul operatiei este negativ.
- Bitul 8 – TRAP (notat si T). Il vom explica ulterior.
- Bitul 9 – INTERRUPT (notat si I). Valoarea 1 acestui bit indica faptul ca intreruperile sunt activate. Intreruperile vor fi explicate ulterior.
- Bitul 10 – DIRECTION (notat si D). Acest bit este folosit de operatiile pe siruri de octeti si indica daca zonele de memorie sursa si destinatie se parcurg de la adresa mica la adresa mare (valoarea 1) sau invers.
- Bitul 11 – OVERFLOW (notat si O). Valoarea acestui bit 1 indica faptul ca a avut loc o depasire daca privim operanzii ca fiind cu semn. Exemplu: 55 + 55 = AA, AA + AA = 54. Ambele operatii se termina cu overflow. In primul

caz am adunat doua numere pozitive si a dat rezultat negativ, iar in al doilea caz am adunat doua numere negative si rezultatul este pozitiv.

4.2 Adrese de memorie relative si absolute

Adresele absolute pe care le poate adresa microprocesorul INTEL 8088 sunt pe cinci cifre hexazecimale, dar valorile registrilor au doar patru cifre hexazecimale. In realitate, registrii ale caror continut este interpretat drept pointer (BX, SI, DI, SP, BP si IP), contin in realitate o adresa relative, adica un deplasament in cadrul unui segment de memorie. Adresa absoluta se calculeaza cu ajutorul registrilor de segment astfel:

<registru de segment> * 16 + <deplasament>.

Facem observatia ca inmultirea unui numar de patru cifre hexazecimale cu 16 se face adaugand cifra hexazecimala 0 la sfarsitul numarului, obtinand astfel un numar de cinci cifre hexazecimale.

Adresele absolute ale instructiunilor se calculeaza folosind registrul CS, adresele absolute ale datelor se caclueaza, de regula, folosind registrul DS, iar adresele absolute ale obiectelor din stiva se calculeaza, de regula folosind registrul SS.

4.3 Operanzii instructiunilor si moduri de reprezentare ale acestora

Instructiunile INTEL 8088 pot avea maxim doi operanzi. Pentru aceste instructiuni vom numi operandul din stanga *destinatie*, iar pe cel din dreapta *sursa*. Operanzii pot fi:

1. Registrii, pe 8 sau 16 biti, diferiti de IP si flaguri
2. Continutul unor locatii de memorie pe 8 sau 16 biti. Acestea se reprezinta prin registrii de index BX, SI, DI, BP intre paranteze patrate. In situatia in care din context nu

se poate deduce daca operatia se efectueaza pe 8 sau 16 biti, atunci se poate prefixa operandul cu BYTE PTR (pentru 8 biti) si WORD PTR (pentru 16 biti). Exemple

[BX], BYTE PTR [SI], WORD PTR [DI]

De asemenea pot fi folosite si deplasamente multiple cum ar fi [BX][SI] sau [BX][DI] + 2.

Daca dorim sa citim date din alt segment decat cel implicit putem prefixa operandul cu simbolul registrului de segment, urmat de caracterul ":". De exemplu de la adresa memorata de BX dorim sa citim din ES si nu din DS scriem

ES:[BX]

Locatiile de memorie mai pot fi adresate folosind direct offsetul lor fara a mai folosi registrii de index. In acest fel adresa trebuie precedata de BYTE PTR sau WORD PTR

3. Constante. Este evident ca nu pot fi folosite decat pe post de sursa. Constantele pot fi expromate in urmatoarele moduri:
    - zecimal
    - hexazecimal urmata de caracterul H. O constanta hexazecimala care incepe cu una din cifrele A,…F trebuie precedata de o cifra 0 nesemnificativa.
    - binar (in baza) urmata de caracterul B
      Atragem atentia ca daca scriem in text un numar hexazecimal care contine cifrele A-F dar neurmat de litera H, asamblorul va furniza eroare

Nu se poate ca atat sursa cat si destinatia sa fie in memorie. Sunt interzise operatiile "memory to memory"

4.4 Instructiuni de transfer de date

Nici una din instructiunile descrise in aceasta sectiune nu modifica indicatorii de conditie

Instructiunea MOV

MOV *destinatie, sursa*

Exemple

```
MOV GAMMA_BYTE, AL
MOV AL, GAMMA_BYTE
MOV DS, AX
MOV DX, DS
MOV ARRAY[DI], DX
MOV BX, 1999
MOV BYTE PTR MEM_BYTE, 66
```

Comentarii

- In exemplele de mai sus GAMMA_BYTE, ARRAY si MEM_BYTE sunt offseturi in zona de date care se defines prin etichete sau prin directiva EQU. Nu ne ocupam de acest detaliu in acest material
- Nu se pot muta direct constant in registrii de segment
-

Instructiunea PUSH

PUSH *sursa*

Se pune o data de 16 biti in stiva. Registrul SP scade cu 2 unitati. Exemple

```
PUSH SI
PUSH ES
PUSH BETA[BX]
```

Instructiunea POP

POP *destinatie*

Extrage o data de 16 biti din stiva. Registrul SP creste cu 2 unitati. Exemple

POP DX

POP DS

POP ALPHA [BX]

Instructiunea XCHG

XCHG *destinatie1, destinatie2*

Schimba continutul celor doi operanzi intre ei. Exemple:

XCHG SI, AX

XCHG BX, DELTA_WORD

Instructiunea LEA (Load effective address)

LEA *destinatie, sursa*

Primul operand trebuie sa fie registru, iar al doilea o adresa de memorie. Se va incarca in registru adresa celui de al doiea operand si nu continutul locatiei de memorie. Exemplu:

LEA DX, BETA [BX] [SI]

Instructiunea LDS (Load data segment register)

LDS *destinatie, sursa*

Primul operand trebuie sa fie registru, iar al doilea o adresa de memorie. Se va incarca in registru valoarea cuvantului aflat la adresa de memorie desemnata, iar cuvantul urmator din memorie se va incarca in registrul de segment DS. Este de fapt o incarcare pe 32 de biti. Exemplu:

LDS SI, NEWSEG [BX]

Instructiunea LES (Load extra-segment register)

LES *destinatie, sursa*

Primul operand trebuie sa fie registru, iar al doilea o adresa de memorie. Se va incarca in registru valoarea cuvantului aflat la adresa de memorie desemnata, iar cuvantul urmator din memorie se va incarca in registrul de segment ES. Este de fapt o incarcare pe 32 de biti. Exemplu:

LES DI, NEWSEG [BX]

4.5 Instructiuni aritmetice

Toate aceste instructiuni afecteaza toti indicatorii de conditie mai putin DIRECTION, TRAP si INTERRUPT.

Instructiunea ADD

ADD *destinatie, sursa*

Se aduna sursa la destinatie. Exemple:

ADD AX, BX
ADD AX, BETA [SI]
ADD BETA [DI], AX
ADD AL, 3
ADD GAMMA [DI], 48
ADD CX, 1776

Instructiunea SUB

SUB *destinatie, sursa*

Se scade sursa din destinatie. Exemple:

SUB CH, DL
SUB DI, ALPHA [SI]
SUB MEM_BYTE [DI], BL
SUB AL, 4
SUB AX, 660

SUB CL, VAL_SIXTY
SUB MEM_BYTE [DI], VAL_SIXTY

Instructiunea MUL (inmultire de numere fara semn)

MUL *sursa*

Daca sursa este pe 8 biti, atunci se inmulteste valoarea registrului AL cu sursa si rezultatul este in AX. Daca sursa este pe 16 biti, atunci se inmulteste valoarea registrului AX cu sursa si rezultatul este in registrii AX (partea cea mai putin semnificativa) si DX (partea cea mai semnificativa). Exemple:

MUL BYTE PTR RSRC_BYTE
MUL WORD PTR RSRC_WORD

Instructiunea IMUL (inmultire cu semn)

IMUL *sursa*

Aceaste instructiune este similara cu instructiunea MUL, numai ca operanzii sunt priviti ca numere cu semn.

Instructiunea DIV (impartire fara semn)

DIV *sursa*

Daca sursa este pe 8 biti, atunci se imparte valoarea lui AX la sursa, catul fiind memorat in AL, iar restul in AH. Daca sursa este pe 16 de biti, se imparte numarul de 32 de biti continut in registrii (DX; AX) (AX contine partea cea mai putin semnificativa), catul fiind memorat in AX, iar restul in DX.

In ambele cazuri, daca rezultatele nu pot fi memorate in registrii corespunzatori, se va genera o intrerupere. Asa cum am mai spus notiunea de intrerupere va fi explicata ulterior.

Exemple:

DIV BYTE PTR DIVISOR_BYTE

DIV WORD PTR DIVISOR WORD

Instructiunea IDIV (impartire cu semn)

IDIV *sursa*

Aceaste instructiune este similara cu instructiunea DIV, numai ca operanzii sunt priviti ca numere cu semn.

Instructiunea ADC (adunare cu transport)

ADC *destinatie, sursa*

Aduna sursa la destinatie impreuna cu valoarea indicatorului CARRY care poate fi 0 sau 1. Este o instructiune utila in cazul in care dorim sa adunam numere ce ocupa mai multi octeti sau cuvinte de 32 de biti. Partea cea mai putin semnificativa se va aduna cu ajutorul instructiunii ADD, iar partile mai semnificative se vor aduna, pe rand cu ajutorul instructiunii ADC, adica cu transportul rezultat la adunarea anterioara.

Exemple:

ADC AX, SI
ADC AX, BETA[SI]
ADC ALPHA [BX] [SI], DI
ADC AL, 3
ADC ALPHA [BX] [DI], 4
ADC DH, 65

Instructiunea SBB (scadere cu transport)

SBB *destinatie, sursa*

Se scade sursa din destinatie impreuna cu valoarea indicatorului CARRY care poate fi 0 sau 1. Este o instructiune utila in cazul in care dorim sa scadem numere ce ocupa mai multi octeti sau cuvinte de 32 de biti. Partea cea mai putin semnificativa

se va scadea cu ajutorul instructiunii SUB, iar partile mai semnificative se vor scadea, pe rand cu ajutorul instructiunii SBB, adica cu transportul rezultat la scaderea anterioara.

Exemple:

```
SBB AX, BX
SBB BL, MEM_BYTE [DI]
SBB GAMMA [BX] [DI], SI
SBB AL, 60
SBB AX, 660
SBB BX, 2001
SBB BYTE PTR MEM_WORD [BX], 79
```

Instructiunea INC (incrementare)

INC *destinatie*

Incrementeaza destinatia cu 1. Nu modifica indicatorul CARRY.

Exemple:

```
INC CX
INC BYTE PTR [BX]
```

Instructiunea DEC (decrementare)

DEC *destinatie*

Decrementeaza destinatia cu 1. Nu modifica indicatorul CARRY.

Exemple:

```
DEC SI
DEC BYTE PTR MEM_WORD
```

Instructiunea NEG (schimbarea semnului)

NEG *destinatie*

Schimba semnul destinatiei, mai precis inlocuieste valoarea acesteia cu complementul sau fata de 2.

4.6 Instructiuni pentru calcule in BCD si ASCII

Instructiunea DAA (ajustare zecimala dupa adunare)

DAA

Sa presupunem ca registrii pe 8 biti AL si BL contin numere reprezentate in BCD. De exemplu AL contine 58H, iar BL 86H. Dupa executarea instructiunii

ADD AL, BL

Registrul AL va contine valoarea 0DEH, iar CARRY=0. In urma unui calcul corect BCD, AL va trebuie sa contina 44H, iar CARRY sa aiba valoarea 1. Aplicand dupa instructiunea de adunare DAA, AL si bitul CARRY vor avea valorile corecte BCD,

Facem observatia ca DAA, ca si toate instructiunile prezentate in aceasta sectiune nu au operanzi, ci se aplica doar registrului AL sau impreuna cu AH daca este vorba de o ajustare ASCII. De asemenea aceste instructiuni folosesc indicatorul HALF CARRY.

De asemenea, subliniem ca DAA se poate aplica dupa orice operatie de adunare pe 8 biti, fie ea ADD sau ADC.

Instructiunea DAS (ajustare zecimala dupa scadere)

DAS

Aceleasi explicatii ca la DAA, numai ca se aplica registrului AL in urma unei operatii de scadere, fie ea SUB sau SBB.

Instructiunea AAA (ajustare ASCII dupa adunare)

AAA

Sa presupulnem ca registrul AL contine valoarea 08H iar registrul BL contine 06H. Dupa executarea instructiunii

ADD AL, BL

AL va contine 0EH. In urma unui calcul corect in format zecimal despachetat, AL ar trebui sa contina valoarea 04H, iar AH 01H. Dupa aplicarea instructiunii AAA, AL si AH vor contine valorile corecte.

Instructiunea AAS (ajustare ASCII dupa adunare)

AAS

Aceleasi explicate ca la AAA, numai ca instructiunea se aplica in urma unei operatii de scadere.

Instructiunea AAM (ajustare ASCII dupa inmultire)

AAM

Sa presupulnem ca registrul AL contine valoarea 08H iar registrul BL contine 06H. Dupa executarea instructiunii

MUL BL

AL va contine 30H. In urma unui calcul corect in format zecimal despachetat, AL ar trebui sa contina valoarea 08H, iar AH 04H. Dupa aplicarea instructiunii AAA, AL si AH vor contine valorile corecte.

Instructiunea AAD (ajustare ASCII inainte de impartire)

Deimpartitul este continut in AX. Se aplica instructiunea AAD, dupa care se va face impartirea. In registrii se vor regasi valorile corecte in format zecimal despachetat.

4.7 Instructiuni booleene (operatii pe bit)

Toate instructiunile prezentate in aceasta sectiune afecteaza toti indicatorii de conditie, cu exceptia lui TRAP,

INTERRUPT si OVERFLOW, ca si la operatiile aritmetice, numai ca indicatorii CARRY si OVERFLOW vor lua in mod cert valoarea zero.

Instructiunea AND (SI bit cu bit)

AND *destinatie, sursa*

Realizeaza conjuncita boleeana bit cu bit a celor doi operanzi. Exemplu

01011000 AND
10001010

00001000

Exemple:

AND CX, DI
AND BX, GAMMA [BX] [SI]
AND GAMMA [BX] [DI], SI
AND AX, 3
AND DX, 7A46H
AND GAMMA [DI], 14

Instructiunea OR (SAU bit cu bit)

OR *destinatie, sursa*

Realizeza disjunctia booleeana bit cu bit a celor doi operanzi.

Exemplu:

00111010 OR
01011100

01111110
Exemple:
OR CX, DI
OR SI, ALPHA [BX] [SI]
OR GAMMA [DI], BX
OR AL, 1110110B
OR AX, 23F6H
OR DL, 23F5H
OR ALPHA [DI], 3DH

Instructiunea XOR (suma modulo 2 bit cu bit)
XOR *destinatie, sursa*
Exemplu
00111010 XOR
01010110

01101100
Exemple:
XOR CX, DI
XOR SI, ALPHA [BX] [SI]
XOR GAMMA [DI], BX
XOR AL, 0F8H
XOR AX, 23F8H
XOR DI, 23F5H
XOR GAMMA [BX] [DI], 0FACEH
Instructiunea NOT (negare bit cu bit)
NOT *destinatie*
Se complementeaza fiecare bit al destinatiei, adica se inlocuieste destinatia cu complementul sau fata de 1.
Aceasta instructiune nu modifica indicatorii de conditie.

4.8 Instructiuni de shiftare sau de rotire a bitilor

In principiu shiftarea de biti la stanga insemana inmultire cu 2, iar shiftarea la dreapta, impartire la 2. Instructiunile prezentate in aceasta sectiune modifica toti indicatorii de conditie cu exceptiile HALF CARRY, TRAP, INTERRUPT si DIRECTION.

Toate aceste instructiuni au sintaxa:

Mnemonica destinatie, n

unde *n* poate fi o constanta sau continutul unui registru pe 8 biti.

Instructiunea SAL (shift arithmetic left)

SAL *destinatie, n*

Destinatia va fi inmultita cu 2^n , iar CARRY va contine ultimul bit “pierdut” in urma acestei operatii.

Exemple:

```
SAL CX, 1
SAL BYTE PTR ALPHA [DI], 1
SAL AX, CL
SAL BYTE PTR MY_BYTE, CL
```

Instructiunea SAR (shift arithmetic right)

SAR *destinatie, n*

Destinatia va fi impartita cu 2^n , bitul de semn fiind duplicat cu cel mai semnificativ bit, adica bitul de semn. CARRY va contine ultimul bit “pierdut” in urma acestei operatii.

Exemple:

```
SAR CX, 1
SAR BYTE PTR MEM_BYTE, 1
SAR AX, CL
```

Instructiunea SHL (shift logical left)

SHL *destinatie, n*

Destinatia va fi inmultita cu 2^n , iar CARRY va contine ultimul bit “pierdut” in urma acestei operatii.

Exemple:

SHL CX, 1

SHL BYTE PTR ALPHA [DI], 1

SHL AX, CL

SHL BYTE PTR MY_BYTE, CL

Instructiunea SHR (shift arithmetic right)

SHR *destinatie, n*

Destinatia va fi impartita cu 2^n , bitul de semn fiind inlocuit cu 0. CARRY va contine ultimul bit “pierdut” in urma acestei operatii.

Exemple:

SHR CX, 1

SHR BYTE PTR MEM_BYTE, 1

SHR AX, CL

Instructiunea ROL (rotire la stanga)

Bitul cel mai putin semnificativ ia valoarea bitului cel mai semnificativ, iar CARRY ia valoarea bitului celui mai putin semnificativ al rezultatului.

Exemple:

ROL CX, 1

ROL AX, CL

Instructiunea ROR (rotire la dreapta)

Bitul cel mai semnificativ ia valoarea bitului celui mai putin semnificativ, iar CARRY ia valoarea bitului celui mai semnificativ al rezultatului.

Exemple:

ROR CX, 1

ROR AX, CL

Instructiunea RCL (rotire la stanga prin CARRY)

Bitul cel mai putin semnificativ ia valoarea lui CARRY, iar CARRY ia valoarea bitului care se “pierde”.

Exemple:

RCL CX, 1

RCL AX, CL

Instructiunea RCR (rotire la dreapta)

Bitul cel mai semnificativ ia valoarea lui CARRY, iar CARRY ia valoarea bitului care se “pierde”.

Exemple:

RCR CX, 1

RCR AX, CL

4.9 Instructiuni de test si comparare

Aceste instructiuni modifica doar indicatorii de conditie, nu si operanzi.

Instructiunea TEST

TEST *sursa1, sursa2*

Realizeaza operatie de conjunctie bit cu bit a celor doi operanzi (AND), fara a-i modifica, dar pozitioneaza indicatorii de conditie.

Exemple:

TEST BH, CL

TEST AX, GAMMA [BX] [SI]

TEST AX, 909

TEST SI, 798

TEST [BP] [DI], 6ACEH

Instructiunea CMP

CMP *sursa1, sursa2*

Realizeaza operatie de scadere a celor doi operanzi (SUB), fara a-i modifica, dar pozitioneaza indicatorii de conditie.

Exemple:

CMP BH, CL

CMP AX, GAMMA [BX] [SI]

CMP AX, 909

CMP SI, 798

CMP [BP] [DI], 6ACEH

4.10 Instructiuni de salt

Aceste instructiuni modifica registrul IP, care este contorul de program, cu valoarea unei etichete sau cu valoarea unei expresii pe 16 biti. In textul sursa etichetele se declara cu identificatori astfel:

```
        ADD AX, BX
ETICHETA_MEA:
        PUSH AX
```

Asamblorul va calcula offset-ul (adrsa) etichetei.

Instructiunea JMP (salt neconditionat)

JMP *adresa*

Realizeaza un salt neconditionat la adresa mentionata.

Exemple:

JMP ETICHETA_MEA

JMP ALPHA [BP] [DI]

JMP AX

Instructiunile J*xx* (salturi conditionate)

Aceste instructiuni au sintaxa identical cu cea a instructiunii JMP, singura difernta fiind mnemonica. Saltul se realizeaza daca indicatorii de conditie corespunzatori au valorile cerute. Aceste salturi pot fi:

Mnemonici + semnificatie	Mnemonica echivalenta	Indicatori de conditie pentru realizarea saltului
JA, (Jump if above)	JNBE	CARRY=0 si ZERO=0
JAE (Jump if above or equal)		CARRY=0
JB (Jump if bellow)	JC	CARRY=1
JBE (Jump if below or equal)	JNA	CARRY=1 sau ZERO=1
JE (Jump if equal)	JZ	ZERO=1
JG (Jump if greater)		SIGN = OVERFLOW si ZERO=0
JGE (Jump if greater or equal)	JNL	SIGN=OVERFLOW
JL (Jump if less)	JNGE	SIGN <> OVERFLOW
JLE (Jump if less or equal)	JNG	SIGN = OVERFLOW sau ZERO=1
JNA (jump if not above)	JBE	CARRY=1 si ZERO=1
JNAE (jump if not above or equal)	JB	CARRY=1
JNB (jump if not bellow)	JNC, JAE	CARRY=0

JNBE (jump if not below or equal)		CARRY=0 si ZERO=0
JNE (jump if not equal)	JNZ	ZERO=0
JNG (jump if not greater)	JLE	ZERO=1 si SIGN <> OVERFLOW
JNGE (jump if not greater or equal)	JL	SIGN <> OVERFLOW
JNL (jump if not less)	JGE	SIGN = OVERFLOW
JNLE (jump if not less or equal)	JG	ZERO=0 si SIGN=OVERFLOW
JNO (jump if not overflow)		OVERFLOW=1
JNP (jump if not parity)	JPO	PARITY=0
JNS (jump if not sign)		SIGN=0
JNZ (jump if not zero)	JNE	ZERO=0
JO (jump if overflow)		OVERFLOW=1
JP (jump if parity)	JPE	PARITY=1
JPE (jump if parity even)	JP	PARITY=1
JPO (jump if parity odd)	JNP	PARITY=0
JS (jump if sign)		SIGN=1
JZ (jump if zero)	JE	ZERO=1

Instructiunea JCXZ (salt daca CX este zero_

JCXZ *adresa*

Saltul se realizeaza daca CX = 0

Exemplu:

JCXZ ETICHETA_MEA

Instructiunea CALL (apel de subprogram)

CALL *adresa*

Instructiunea CALL produce ca si JMP schimbarea registrului IP cu *adresa.* In plus se introduce in stiva adresa instructiunii urmatoare.

Atentie: se pot apela si proceduri al caror cod se afla in alt segment decat CS-ul actual (asa zise proceduri FAR). In acest caz se va introduce in prealabil valoarea lui CS in stiva.

Deci instructiunea call este echivalenta cu o instructiune PUSH, urmata de JMP.

Exemple:

CALL MY_PROC

CALL WORD PTR [BX] [SI]

Instructiunea RET (intoarcere in rutina apelanta a procedurii)

RET

IP va lua valoarea citita din varful stivei. Deci RET este echivalenta cu un POP urmat de JMP.

Daca apelul a fost de tip FAR se va extrage din varful si continutul registrului CS.

Atentie: In mod normal, intr-o subrutina numarul de PUSH trebuie sa fie egal cu numarul de POP, astfel ca instructiunea RET sa gaseasca in varful stivei adresa instructiuni care urmeaza celei de apel a subrutinei.

Instructiunea IRET (intoarcere dintr-o rutina de serviciu a unei intreruperi)

La primirea unei intreruperi se va apela rutina de serviciu (handler) a acesteia (notiune ce va fi explicate ulterior). La aces apel se vor introduce in stiva indicatorii de conditie, valoarea lui

CS, urmata de adresa urmatoare insructiunii dupa care s-a primit intreruperea.

IRET extrage in IP valoarea gasita in varful stivei, in CS valoarea urmatoare si in indicatorii de conditie valoarea urmatoare. Dupa care realizeaza saltul. Deci IRET este echivalenta cu 3 POP urmat de un JMP.

Instructiunile de tip LOOP (iteratii)

Forma cea mai simpla a acestei instructiuni este

LOOP *adresa*

Se va decrement registrul CX. Daca in urma decrementarii, valoarea acestuia este diferita de zero se va face salt la *adresa.*

Aeasta instructiune nu afecteaza indicatorii de conditie. Exista si forme de LOOP “conditionate” in care bucla se opreste daca valoarea lui CX este zero **sau** daca indicatorii de conditie au valori corespunzatoare.

Mnemonica	Mnemonica echivalenta	Conditie suplimentara de oprire a buclei
LOOPE	LOOPZ	ZERO=0
LOOPNE	LOOPNZ	ZERO=1
LOOPNZ	LOOPNE	ZERO=1
LOOPZ	LOOPE	ZERO=0

Instructiunea INT (intrerupere software)

INT *numar*

Adresa absoluta de bootstrap este CS=0FFFFH si IP=0, adica 0FFFF0H. La adresa absoluta 0 (zero) se poate memora o tabela de 256 de intrari de cate patru octeti, primii doi reprezentand valoarea pe care o va lua IP dupa executia instructiunii, iar urmatorii doi valoarea pe care o va lua registrul CS dupa executia instructiunii.

Executia instructiunii se desfasoara dupa urmatorul scenariu: se pune in stiva valoarea registrului de flaguri, flagurile INTERRUPT si TRAP iau valoarea 0 (zero), se pune in stiva valoarea registrului CS, registrul CS va lua valoarea continuta la adresa *numar**4 + 2, se pune in stiva valoarea registrului IP, registrul IP va lua valoarea continuta la adresa *numar*4.* Se realizeaza salt la adresa furnizata de CS*16 + IP.

Deci aceasta instructiune este echivalenta cu 3 instructiuni de PUSH, resetarea a doua flaguri (INTERRUPT si OVERFLOW), urmata de un salt.

Exemple

INT 3

INT 67

Instructiunea INTO (interrupt if overflow)

INTO

Efect similar cu cel al instructiunii INT 4, cu deosebirea ca se executa doar cand OVERFLOW=1.

Observatie: In realitate se realizeaza un CALL de tip FAR, punand, suplimentar indicatorii de conditie in stiva. Iesirea corecta din subrutinele apelate se face cu IRET, si nu cu RET, pentru restaurarea corecta a indicatorilor de conditie.

4.11 Instructiuni pentru tratarea zonelor compacte de memorie

Toate mnemonicele acestor instructiuni pot fi postfixate de litera B, pentru operatii pe octet, sau de litera W pentu operatii pe cuvant, in situatia in care nu se deduce din context despre ce tip de operatie este vorba. Toate aceste instructiuni au sursa si/sau destinatie. Offset-ul sursei est in SI, iar cel al destinatiei in DI, cu observatia ca, pentru offset-ul memorat de DI, adresa absoluta a

destinatiei se calculeaza folosind registrul de segment ES. Toate aceste instructiuni folosesc indicatorul DIRECTION (pentru pozitionarea acestuia vezi instructiunile STD si CLD). Daca DIRECTION=0 atunci sursa si/sau destinatia se incremanteaza cu 1 sau 2 dupa caz, iar daca DIRECTION=1 atunci sursa si/sau destinatia se deceremenTeaza cu 1 sau 2 dupa caz. Aceste instructiuni pot fi prefixate de REP: sau REPZ:, REPE:, REPNZ:, REPNE:. In acest caz instructiunea se va repeta, decrementand la fiecare iteratie valoarea lui CX, pana cand valoarea lui CX ajunge zero **sau** indicatorul de conditie ajunge 0 (pentru REPZ si REPE) sau 1 (pentru REPNZ sau REPNE).

Instructiunea LODS (LODSB/LODSW) (incarcare in accumulator a valorii din memorie)

LODS

Pentru operatiile pe byte valoarea octetului de la adresa desemnata de SI se incarca in AL, iar pentru operatiile pe word valoarea cuvantului de la adresa desemnata de SI este incarcata in AX. SI se incrementeaza sau decrementeaza cu 1 sau 2, dupa caz. Indicatorii de conditie nu sunt afectati. In consecinta, singurul prefix care (poate) are sens este REP.

Instructiunea STOS (STOSB/STOSW) (descarcare a valorii acumulatorului in memorie)

STOS

Pentru operatiile pe byte valoarea lui AL se descarca la adresa desemnata de DI, iar pentru operatiile pe word, valoarea lui AX se descarca la adresa desemnata de DI. DI se incrementeaza sau decrementeaza cu 1 sau 2, dupa caz.

Indicatorii de conditie nu sunt afectati. In consecinta, singurul prefix care (poate) are sens este REP.

Instructiunea MOVS (MOVSB/MOVSW) (incarcarea a destinatiei cu valoarea sursei)

MOVS

Valoarea octetului sau a cuvantului de la adresa desemnata de SI se transfera la adresa desmneta de DI. SI si DI vor fi incrementate sau decrementate cu 1 sau 2, dupa caz. Indicatorii de conditie nu sunt afectati. In consecinta, singurul prefix care are sens este REP.

Instructiunea CMPS (CMPSB/CMPSW) (compararea zonelor de memorie)

CMPS

Se compara (prin scadere) valoarea octetului sau a cuvantului de la adresa desemnata de DI, cu cea a octetului sau a cuvantului de la adresa desemnata de SI. SI si DI vor fi incrementate sau decrementate cu 1 sau 2, dupa caz. Indicatorii de conditie sunt afectati. In consecinta, toate prefixele au sens sa fie folosite.

Instructiunea SCAS (SCASB/SCASW) (cautarea unei valori in memorie)

SCAS

Se compara (prin scadere) valoarea octetului sau a cuvantului de la adresa desemnata de DI, cu cea a registrului AL sau AX, dupa caz. DI va fi incrementat sau decrementat cu 1 sau 2, dupa caz. Indicatorii de conditie sunt afectati. In consecinta, toate prefixele au sens sa fie folosite.

4.12 Instructiuni pentru gestionarea indicatorilor de conditie

Aceste instructiuni permit accesul direct la indicatorii de condititie, in afara instructiunilor care ii afecteaza in mod implicit (aritmetice, logice etc) sau de a caror executie este influentata de valoarea acestora.

Instructiunea STC (Set carry flag)
STC

Atribuie indicatorului CARRY valoarea 1.

Instructiunea CLC (Clear carry flag)
CLC
Atribiue indicatorulul CARRY valoarea 0.

Instructiunea CMC (Complement carry flag)
CMC
Atribuie indicatorului CARRY negatul valorii curente.

Instructiunea STD (Set direction flag)
STD

Atribuie indicatorului DIRECTION valoarea 1.

Instructiunea CLD (Clear direction flag)
CLD

Atribuie indicatorului DIRECTION valoarea 0.

Instructiunea STI (Set INTERUPT flag)
STI

Atribuie indicatorului INTERUPT valoarea 1.

Instructiunea CLI (Clear INTERUPT flag)
CLI

Atribuie indicatorului INTERUPT valoarea 0.

Este acum momentul sa explicam notiunea de intrerupere hard. Un eveniment extern poate produce o intrerupere. Efectul intreruperii este un CALL de tip FAR catre o rutina (numita si HANDLER) de serviciu a acestei intreruperi. Se pun in stiva indicatorii de conditie, urmati de registrii CS si IP. O iesire corecta din rutina de tratare a intreruperii se face cu ajutorul instructiunii IRET (nu cu RET). Dupa iesirea din rutina de tratare a intreruperii cu IRET, programul intrerupt isi va relua activitatea obisnuita. **Atentie:** o programare uzual corecta a rutinelor de tratare a intreruperilor trebuie sa salveze la inceput toti registrii in stiva sis a ii restaureze inainte de IRET.

La primirea unei intreruperi, indicatorul de conditie INTERUPT este pus pe 0. Pana cand acest indicator nu ia valoarea 1, intreruperile sunt dezactivate, mai precis, pentru fiecare tip de intrerupere se memoreaza doar ultima intrerupere de tipul reaspectiv, iar la activarea intreruperilor, aceasta intrerupere va fi tratata.

Instructiunea PUSH (Salvare indicatori de conditie in stiva)
PUSHF
Salveaza in stiva registrul de indicatori de conditie

Instructiunea POPF (restaurare indicatori de conditie din stiva)

POPF

Reface registrul de indicatori de conditie cu ce gaseste in varful stivei. In mod evident, toti indicatorii de conditie vor fi afectati.

Instructiunea LAHF (incarca registrul AH cu valoarea unor indicatori de conditie)

LAHF

Valoarea registrului AH va contine, pe biti, urmatoarele valori:

SIGN; ZERO; Nedefinit; HALF CARRY; Nedefinit; PARITY; Nedefinit; CARRY

Instructiunea SAHF (incarca valoarea unor indicatori de conditie cu continutul registrului AH)

SAHF

Valoarea registrului AH contine, pe biti, urmatoarele valori:

SIGN; ZERO; Nedefinit; HALF CARRY; Nedefinit; PARITY; Nedefinit; CARRY

In mod evident, sunt afectati indicatorii de conditie implicati.

4.13 Instructiuni diverse

Instructiunea CWD (convert word to double word)

CWD

Expandeaza continutul registrului AX in registrii DX;AX. Mai precis daca cel mai semnificativ bit al lui AX este zero, DX ia valoarea zero, altfel (continutul lui AX este negativ) DX ia valoarea 0FFFFH.

Instructiunea CBW (convert byte to word)

CBW

Expandeaza continutul registrului AL in registrul AX. Mai precis daca cel mai semnificativ bit al lui AL este zero, AH ia valoarea zero, altfel (continutul lui AL este negativ) AH ia valoarea 0FFH.

Instructiunea XLAT (Translatare)

XLAT

Continutul lui AL este inlocuit cu cel al octetutului continut de adresa de memorie aflata la valoarea lui BX plus valoarea originala a lui AL.

Instructiunea NOP (No operation)

NOP

Nu executa nimic. Aceasta instructiune poate fi folosita pentru a realize o temporizare.

Instructiunea HLT (Halt)

HLT

Opreste activitatea procesorului.

Instructiunea ESC (escape)

ESC

Aceasta instructiune este folosita pentru a trece fluxul de instructiuni catre ale procesoare, cum ar fi, de exemplu, coprocesorul arithmetic 8087 care are drept instructiuni cablate calculele in virgule mobila, fapt care accelereaza in mod substantial aplicatiile care folosesc numere reprezentate in virgula mobile.

Instructiunea IN (citire de la un port)

IN AL, *port*

IN AX, *port*

Porturile pe 8 sau 16 biti trebuie legate din punct de vedere hard la niste periferice. *port* poate fi dat cu adresa directa (pe 16 biti) sau continut in registrul DX.

Exemple

IN AL, BYTE_PORT

IN AL, DX

Instructiunea OUT (scriere la un port)

OUT *port,*AL

OUT *port,* AX

Porturile pe 8 sau 16 biti trebuie legate din punct de vedere hard la niste periferice. *port* poate fi dat cu adresa directa (pe 16 biti) sau continut in registrul DX.

Exemple

OUT BYTE_PORT. AL

OUT DX, AX

Instructiunea WAIT

WAIT

Activitatea procesorului se opreste pana la primirea unei intreruperi. Daca rutina de tratare a intreruperii se termina cu IRET, activitatea procesorului se reia in mod normal, adica de la instructiunea urmatoare a lui WAIT.

4.14 Procesorul pe 32 de biti 80386

Spre deosebire de predecesorii sai (8086, 8088, 80286), procesorul 80386 prezinta doua deosebiri majore:

- Posibilitate de lucru in mod protejat

- Anumiti registrii sunt extinsi la 4 octeti (32 de biti). De exemplu AX este extins la EAX, cu posibilitatea invocarii cuvantului celui mai putin semnificativ prin AX, iar a octetilor din cadrul acestuia prin AH si AL. Dam in continuare lista registrilor extinsi la 32 de biti:

Registru 80386	Registru corespondent 8086/8088/80286	Se pot adresa subregistrii vechi pe 8 biti
EAX	AX	AH, AL
EBX	BX	BH, BL
ECX	CX	CH, CL
EDX	DX	DH, DL
ESI	SI	-
EDI	DI	-
EBP	BP	-
ESP	SP	-
EIP	IP	-
EFLAGS	FLAGS	-

5. In loc de concluzii. Mai precis puntea catre sistemul de operare

Asa cum am zis de la inceput, urmatoarea etapa este sistemul de operare. Este vorba de o aplicatie software care va “controla” toate aplicatiile lansate de utilizatori.

Mult success in tentativa de a deveni adevarate vedete in dezvoltarea aplicatiilor profesioniste.