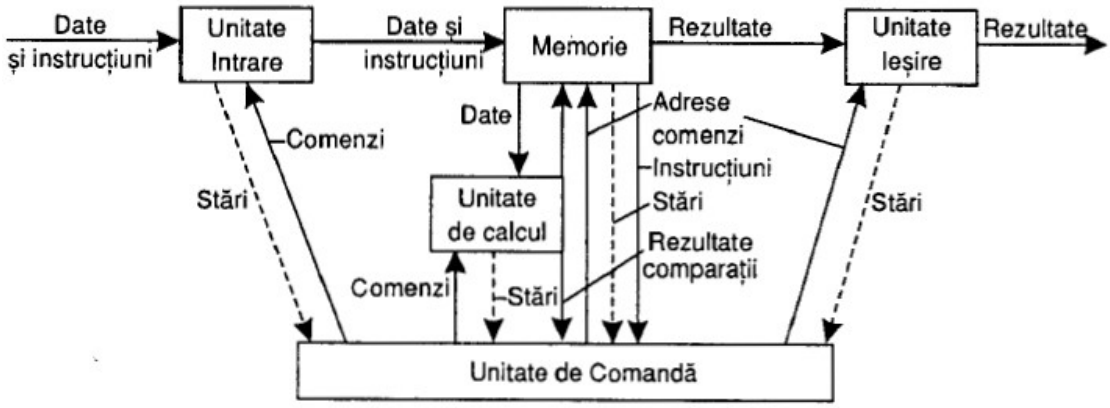
1. Structura unui calculator numeric de tip John von Neumann



2. Ce sunt asambloarele? Descrieti etapele procesului de asamblare.

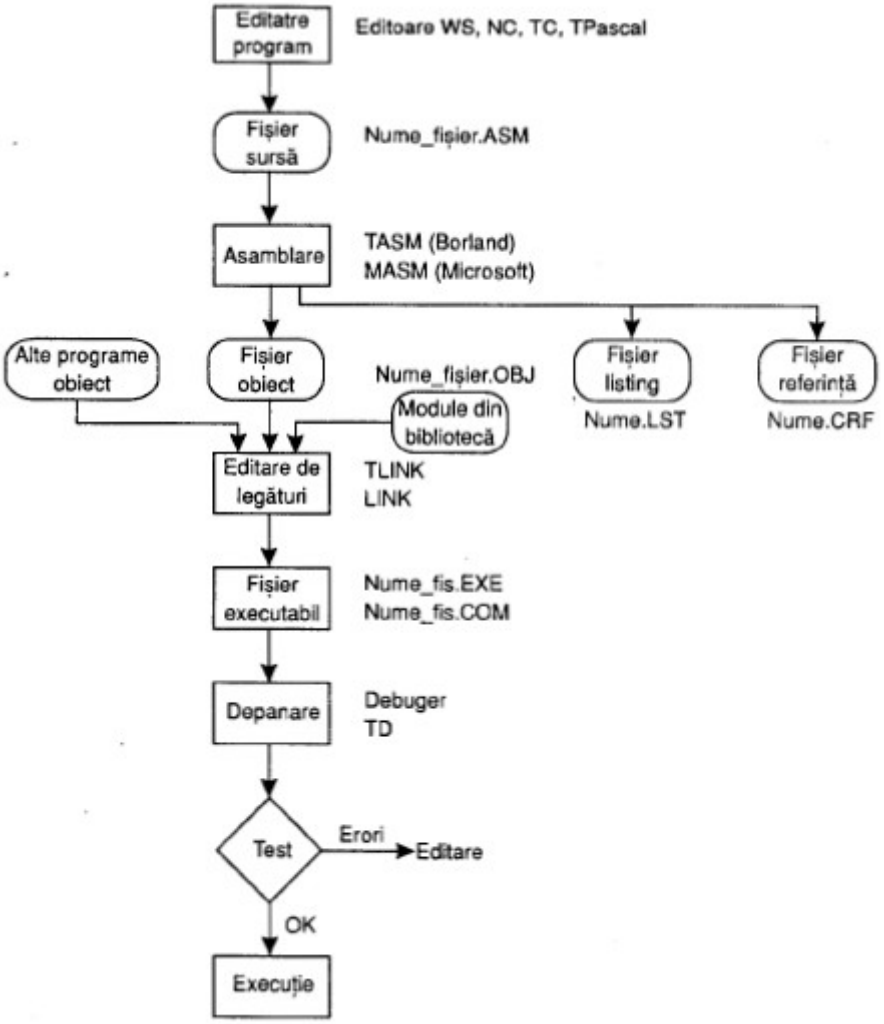
Asambloarele sunt programe care translateaza programele sursa, scrise in limbaj de asamblare, in cod masina.

Operatia de asamblare se realizeaza in doi pasi:

1. La primul pas se genereaza o tabela de simboluri, ce va contine toate numele simbolice din program, exceptand numele simbolice externe (definite in alte module), instructiuni si directive de asamblare. In aceasta etapa asamblorul contorizeaza instructiunile si datele si asociaza numelor simbolice o pozitie relativa (deplasament) fata de inceputul programului, ca si cum programul ar incepe de la adresa 0. In realitate, programul nu se incarca in memoria RAM de la adresa zero, care este zona folosita de sistyemul de operare, ci adresa de la care se incarca este furnizata de sistemul de operare, in functie de spatiul de memorie disponibil; din acest motiv programul furnizat de asamblor este relocabil.

2. La pasul al doilea se obtine programul obiect, translatand instructiune cu instructiune programul si inlocuind in instructiunile respective numele simbolice cu valorile numerice asociate in tabela de simboluri. Programul executabil se obtine in urma etapei editarii de legaturi, care permite legarea mai multor module relocabile intr-un singur fisier; acest fisier va contine fisierul executabil, rezolvandu-se toate referintele incrucisate care au fost folosite in alte module.

3. Etapele parcurse pentru realizarea unui program în limbaj de asamblare.



4. Operatiile executate în mod repetat de procesor, la care se rezuma functionarea sa. (ciclurile realizate de procesor pentru executia unei instructiuni)

1. Citeste instructiunea – pe durata acestui ciclu se transmite adresa instructiunii de executat si se aduce, din memorie, instructiunea in CPU (ciclul *Fetch*).
2. Decodifica instructiunea.
3. Transmite adresa si citeste un operand din memoriem daca se specifica in instructiune (ciclul *Read*)
4. Executa instructiunea (ciclul *Execution*).
5. Transmite adresa si scrie rezultatul in memorie, daca intructiunea o cere (ciclul *Write*).

5. Descrieti unitatile functionale ale unui procesor (286/386/486/Pentium).

Cele patru unitati functionale ale procesorului 286 sunt: unitatea de adresare (AU), unitatea de interfata cu magistrala (BU), unitatea de decodificare instructiuni (IU) si unitatea de executie (EU).

Unitatea de adresare(AU – Address Unit) determina adresa fizica si contine un sumator de deplasament (offset), care determina deplasamentul in functie de modul de adresare si un sumator de adrese fizice (de 20 sau 24 biti, in functie de modul de lucru); aceasta determina adresa fizica, ca suma intre adresa de segment si offset. De asemenea, unitatea mai realizeaza diferite verificari (in modul protejat), verifica limita si dimensiunea unui segment si furnizeaza adresa de baza a segmentului; aceasta unitate este complet izolata de exterior.

Unitatea de interfata cu magistrala (BU – Bus Unit) contine circuite driver si latch pentru adrese, o unitate de citire anticipata a instructiunilor (PreFetcher), interfata cu extensia de procesor (extensia de procesor este un alt procesor specializat, cum ar fi procesul(?) de intrare/iesire sau coprocesorul matematic, devenit unitate in virgula mobila - FPU), logica pentru controlul magistralei, o coada de instructiuni (de 6 octeti) si realizeaza transmisia/receptia datelor. Luungimea cozii a fost aleasa astfel incat BU sa tina ocupata unitatea de executie cat mai mult timp posibil. De fapt, aceasta unitate realizeza comunicatia cu exteriorul.

Unitatea de citire anticipata (prefetcher) a instructiunilor realizeaza functia de anticipare a programului. Atunci cand BU nu efectueza cicluri de magistrala (citire/scriere operand) pentru executia unei instructiuni, aceasta unitate utilizeaza BU pentru citirea secventiala, in avans a fluxului de instructiuni. BU lanseaza o cerere de citire (din memorie) de instructiune, imediat ce exista 2 octeti liuberi in coada de instructiuni; citirea

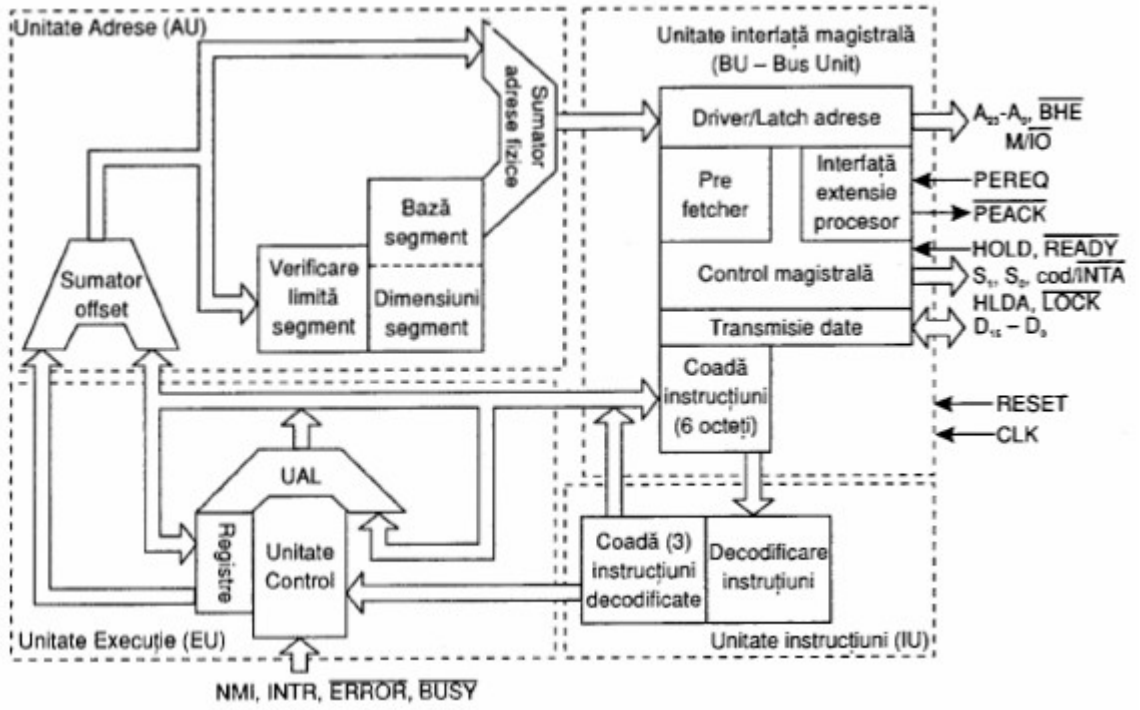
instructoiunilor se face numai pe 16 biti, intr-un ciclu de acces la memorie. Daca, prin program, se citeste de la o adresa impara, BU citeste octetul de la adresa impara si apoi reia citirea a cate doi octeti de la adresele pare urmatoare.

In general, BU contine cel putin un octet in coada de instructiuni, deci EU nu trebuie sa astepte executia instructiunilor. In coada sunt introduse instructiunile de la adresele imediat urmatoare instructiunii curente, reprezentand instructiunile ce urmeaza sa se execute, daca nu se intalneste o instructiune de transfer al controlului de program (salt). Daca, la un moment dat, se transfera controlul la alta locatie de memorie, coada este initializata, BU citeste instructiunea de la noua adresa, o transfera catre EU si incepe imediat citirea instructiunilor urmatoare.

Unitatea de decodificare a instructiunilor (IU – Instruction Unit) preia octetii din coada de instructiuni si ii translateaza in miocrocod. Instructiunile decodificate (in numar de trei) sunt puse intr-o coada, unde asteapta sa fie prelucrate de catre unitatea de executie (EU). Unitatea lucreaza in paralel cu celelalte unitati si incepe o noua decodificare in momentul in care o locatie din coada devine libera.

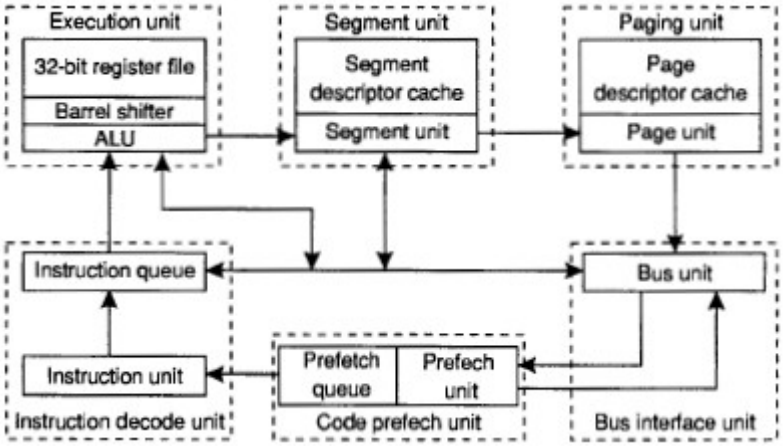
Unitatea de executie (EU - Execution Unit) executa instructiunile din coada de instructiuni decodificate si comunica cu celelate unitati. Aceasta unitate implementeaza functiile de executie ale tuturor instructiunilor, furnizeaza adrese si date spre AU, respectiv BU, manipuleaza registrele generale si indicatorii de stare. Cu exceptia unor pini de control, EU este complet izolata de exterior. Cand EU este gata pentru executia unei instructiuni, citeste codul acesteia din coada de instructiuni decodificate gestionata de IU si apoi executa instructiunea. Cele doua unitati functioneaza asincron si se sincronizeaza in caz de coada goala sau coada plina. Daca in timpul executiei unei instructiuni EU necesita un acces la memorie sau la circuitele de I/O, se lanseaza o cerere de acces catre BU, care controleaza si efectueaza accesul la adresa furnizata catre AU. Daca o astfel de cerere apare in timp ce BU este intr-un ciclu de magistrala de citire a unei instructiuni, BU termina ciclul curent dupa care raspunde cererii de acces lansata de EU.

6. Structura interna a procesorului 286.

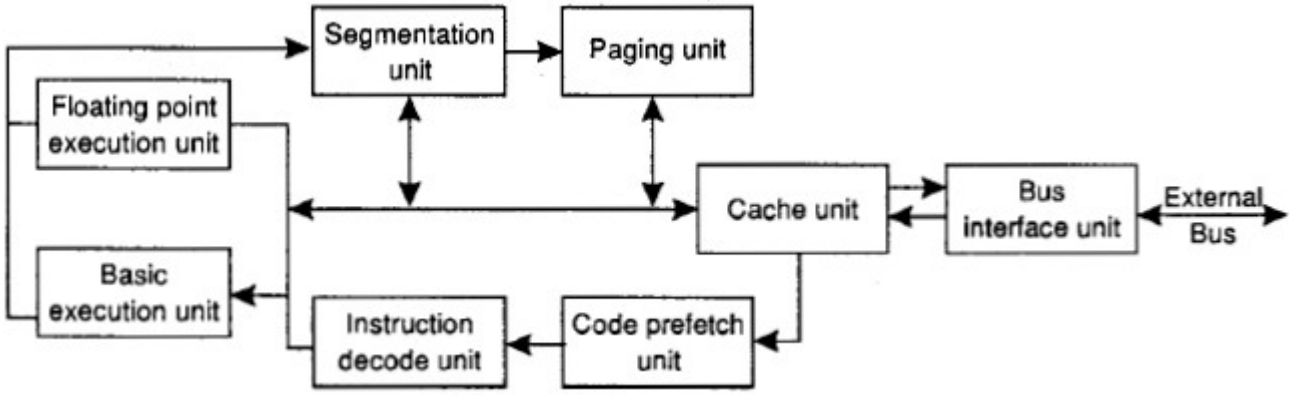


M/IO = separa ciclurile de memorie de cele I/O (1 = mem, halt; 0 = I/O, recunoastere intrerupere)
cod/INTA, S_1 – S_0 = semnale de stare a magistralei
READY = termina ciclul de magistrala
HOLD (bus HOLD request) = permite unui alt modul master (procesor) sa ceara controlul magistralei
HLDA (bus HOLD Acnkowledge)
PEREQ = Processor Extension operand REQuest
PEACK = Processor Extension operand ACKnowledge
INTR = INTerrupt Request
NMI = Non Maskable Interrupt

7. Structura bloc a procesorului 386.



8. Structura bloc a procesorului 486.



9. Unitatile componente ale procesorului 386/486.

Interfata cu magistrala (BIU - Bus Interface Unit) realizeaza legatura procesorului cu lumea externa. Orice alta unitate ce necesita date din exterior va solicita acestei unitati sa realizeze operatia respectiva. Pentru a realiza transferul datelor (la memorie sau la porturile de I/E), unitatea va furniza numai adrese fizice, deci adresele operanzilor trebuie, mai intai, sa treaca prin unitatea de segmentare si cea de paginare, daca este necesar.

Unitatea de citire anticipata a instructiunilor (IPU - Instruction Prefetch Unit sau Code Prefetch Unit) are rolul de a rnentine plina coada de instructiuni; unitatea de decodificare extrage instructiunile din aceasti coada, de 16 octeti, ori de cate ori se elibereaza octeti din aceasta coada. Unitatea de anticipare lanseaza o cerere catre BIU pentru citirea altor octeti din memorie (cite 4 octeti la un ciclu de citire). Deci BIU este ocupata cu o cerere de la o alta unitate, atunci, dupa servirea unitatii respective, va fi luata in considerare si cererea unitatii de anticipare. Unitatea IPU este anuntata ori de cite ori unitatea de executie prelucreaza o instructiune de salt (CALL, JMP sau intrerupere) astfel incat ea poate incepe citirea instructiunilor de la noua adresa. In aceste situatii coada este reinitializata, pentru a evita ca unitatea de executie sa primeasca instructiuni ce nu sunt in secventa executata in mod curent.

Unitatea de decodificare (IDU - Instruction Decode Unit). Aceasta unitate preia octeti individuali din coada de instructiuni si determina numarul de octeti necesari pentru a completa instructiunea urmatoare. O instructiune 80386 poate avea de la 1 la 16 octeti. Dupa extragerea intregii instructiuni din coada, unitatea reformuleaza codul operatiei

intr-un format de instructiune intern si plaseaza instructiunea decodificata intr-o coada de instructiuni (decodificate), care este de 3 operatii. Unitatea de decodificare va semnala unitatea BIU daca instructiunea decodificata va necesita o referire la memorie. Aceasta permite operanzilor instructiunilor sa fie obtinuti anterior executiei instructiunilor.

Unitatea de executie (EU - Execution Unit). Aceasta este acea parte din CPU (Central Processor Unit) care realizeaza calculele. Ea realizeaza diferite operatii (deplasari, adunari, inmultiri, impirtiri etc.) necesare pentru executia instructiunilor. Setul de registre se afla in aceasta unitate. Unitatea contine, de asemenea, o componenta logica denumita *barrel shifter*, care poate realiza deplasiri/rotiri multi-bit, intr-un singur ciclu de ceas. Aceasta facilitate este utilizata de EU nu numai in instructiunile de deplasare, dar si pentru accelerarea inmultirilor si

in generarea adreselor indexate. De asemenea, EU se adreseaza unitatii BIU, cand ea are datele necesare a fi transmise catre memorie sau magistrala de I/E.

Unitatea de segmentare (SU - Segmentation Unit) translează adresele segmentate în adrese liniare. Timpul de translare a segmentului este aproape în întregime ascuns de paralelismul arhitecturii procesorului 386. Timpul tipic este deci de 0 impulsuri de ceas, dar dacă este necesar timp pentru translarea adresei, acesta este de cel mult 1 impuls de ceas. Unitatea de segmentare conține o memorie (cache) ce păstrează informații din tabela de descriptori pentru fiecare dintre cele 6 registre segment.

Unitatea de paginare (PU - Paging Unit) primeşte adresele liniare generate de SU şi le converteşte în adrese fizice. Dacă paginarea nu este activată (deci PU este dezactivată), adresa liniară de la SU devine adresă fizică. Când paginarea este activată, spațiul de adrese liniar al procesorului 386 este împărțit în blocuri de 4096 octeți, denumite pagini. Fiecare pagină poate fi mapată (suprapusă) la o adresă fizică total diferită. Procesorul 386 utilizează o tabelă de pagini pentru a translata fiecare adresă liniară la o adresă fizică. Unitatea conține un cache asociativ denumit *tampon de translare a adresei liniare în adresă fizică-TLB* (Translation Lookaside Buffer), care conține intrările (adresele) pentru 32 de pagini, cel mai recent utilizate. Dacă o intrare în tabela de pagini nu este găsită în TLB, un ciclu de citire din memorie pe 32 de biți aduce intrarea din memoria RAM. în condiții normale de operare, mai puțin de 2% din toate referirile la memorie cer procesorului să caute în afara tabelei TLB o intrare în tabela de pagini. Timpul necesar pentru a realiza translația este între 0 si 5 impulsuri de ceas. Datorită tabelei TLB, întârzierea tipică este de 0,5 impulsuri de ceas.

10. Îmbunatatiri aduse procesoarelor Pentium fata de procesorul 486.

- memoria cache distinctă pentru instrucțiuni şi pentru date (câte 8 Ko pentru fiecare);
- două benzi de asamblare pentru numere întregi (denumite U şi V);
- prezicerea adreselor de salt (numită şi predicție dinamică) folosind un tampon care reține aceste adrese de salt (BTB - Branch Target Buffer);
- unitatea de calcul în virgulă mobilă organizată ca bandă de asamblare;
- magistrala externă pe 64 de biți, deci într-un singur acces (impuls de ceas) la memorie se pot transfera cuvinte de 64 de biți;
- arhitectura internă îmbunătățită (superscalară) şi algoritmii îmbunătățiți (rapizi) pentru execuția instrucțiunilor FPU (cele în virgulă mobilă) într-un singur impuls de ceas sau două instrucțiuni cu numere întregi într-un singur impuls de ceas;
- folosirea parității pentru magistrala de date şi memoriile cache interne;
- monitorizarea performanțelor; execuție pas cu pas;
- alte facilități: modul de administrare al sistemului, extensie pentru mod virtual, suport biprocesor, gestionarea alimentării; detecția erorilor şi a integrității datelor.

11. Descrieti câteva (2-3) concepte noi utilizate la procesoarele Pentium.

Bandă de asamblare (pipeline): mai multe instrucțiuni maşină sunt încărcate secvențial într-o linie complexă de prelucrare, asemănătoare cu memoria FIFO (în locul celulelor de memorie, aici sunt automate secvențiale, care efectuează diferite modificări asupra datelor ce trec prin pipeline) sau cu o bandă de asamblare. Prin pipeline se sparg (translează) instrucțiunile în micro-operații ce se execută în unități separate. Rezultatul este asamblat şi se trimite exteriorului. Dacă s-au încărcat date incorecte în structurile secvențiale din pipeline (de exemplu la execuția speculativă ultimul salt a fost greşit), atunci banda pipeline este golită şi reîncărcată cu date corecte; deci el conține toate operațiile/rezultatele ce se execută în acel moment în diferite stadii, în funcție de poziția lor în structura secvențială din pipeline.

Microinstrucțiune: instrucțiunile în cod maşină sunt sparte în 1-4 microinstrucțiuni de lungime fixă, care se execută cvasiparalel. Timpii lor de execuție sunt cunoscuți şi astfel elementele de prelucrare din pipeline pot face o planificare corectă, în timp, a execuției.

Superscalar: procesoare cu mai multe benzi de asamblare, paralele, care pot executa mai multe instrucțiuni simultan, într-un singur impuls de ceas (Pentium II are două benzi, U şi V); nivelul superscalar reprezintă numărul de instrucțiuni care se pot executa simultan. Se mai utilizează şi termenul de superbandă de asamblare, care înseamnă bandă de asamblare cu peste 10 niveluri (etape sau stagii).

12. Operatiile realizate de procesor pe durata unui ciclu de magistrala, pentru comanda altor dispozitive (memorie, echipamente de I/E).

- se activează pe magistrala de adrese adresa unei locații de memorie sau a unui port I/O şi se memorează într-un registru de adrese extern;
- se generează semnalul de comandă corespunzător pentru citire/scriere date;

- dispozitivul selectat (memoria sau portul I/O) realizează transferul de date şi transmite procesorului un semnal de răspuns, pentru a încheia ciclul.

13. Resursele procesorului (registrele sale, dimensiune, tipul lor, etc.)

Registre generale – funcții speciale

31 16	7 0	7 0	
EAX	AX	(AH) (AL)	Instrucțiuni
EDX	DX	(DH) (DL)	de I/O, *, /
ECX	CX	(CH) (CL)	Numărător
EBX	BX	(BH) (BL)	Registre de
EBP	BP		bază (base)
ESI	SI		Registre
EDI	DI		Index
ESP	SP		Indicator stivă
31 16	15 0		

Registre segment

Code Segment (CS)
Data Segment (DS)
Extra Segment (ES)
Stack Segment (SS)

Registre de stare și control

31 16	15 0
EIP	IP
EF	F
	MSW

14. Descrieti indicatorii din registrul indicatori (stare, control, speciali).

Indicatori de stare:

- OF (Overflow Flag) - este poziționat pe 1 dacă rezultatul unei operații aritmetice depăşeşte domeniul de reprezentare (limita superioară/inferioară a acestuia) pentru numerele cu care se lucrează, adică rezultatul nu poate fi memorat în destinația stabilită de instrucțiune (de exemplu, împărțirea prin zero sau adunarea/înmulțirea a două numere mari - la limita superioară a domeniului).
- SF (Sign Flag) - este indicatorul de semn al rezultatului unei operații şi, de fapt, coincide cu primul bit al rezultatului, fiind 1 dacă numărul este negativ şi 0 pentru pozitiv.
- ZF (Zero Flag) - este poziționat pe 1 dacă rezultatul unei operații aritmetice sau logice este zero, altfel 0
- AF (Auxiliary carry Flag) - este 1 dacă în urma execuției unei instrucțiuni a apărut un transport din rangul 3 în rangul 4 sau un împrumut dinspre rangul 4 spre rangul 3. Acest indicator este utilizat pentru implementarea aritmeticii pentru numere zecimale codificate binar (BCD – Binary Coded Decimal).
- PF (Parity Flag) - este poziționat pe 1 când numărul de unități din rezultat este par (paritate pară)
- CF (Carry Flag) - se poziționează pe 1 dacă a apărut un transport sau un împrumut în/din rangul cel mai semnificativ al rezultatului, în urma execuției unei instrucțiuni aritmetice.

Indicatorii de control:

- DF (Direction Flag) - este utilizat de instrucțiunile pe şiruri şi specifică direcția de parcurgere a acestora (0 parcurgere de la adrese mici spre adrese mari;1 invers)
- IF (Interrupt Flag) - acest indicator controlează acceptarea semnalelor de întrerupere externă. Dacă IF = 1, este activat sistemul de întreruperi, adică sunt acceptate semnale de întrerupere externă (mascabile, pe linia INTR), altfel acestea sunt ignorate. Indicatorul nu are influență asupra semnalului de întrerupere nemascabilă - NMI.
- TF (Trace Flag) - este utilizat pentru controlul execuției instrucțiunilor în regim pas cu pas (instrucțiune cu instrucțiune), în scopul depanării programelor. Dacă indicatorul este 1, după execuția fiecărei instrucțiuni se va genera un semnal de întrerupere intern (pe nivelul 1). Evident, execuția secvenței de tratare a acestor întreruperi se va face cu indicatorul TF = 0.

Indicatori speciali:

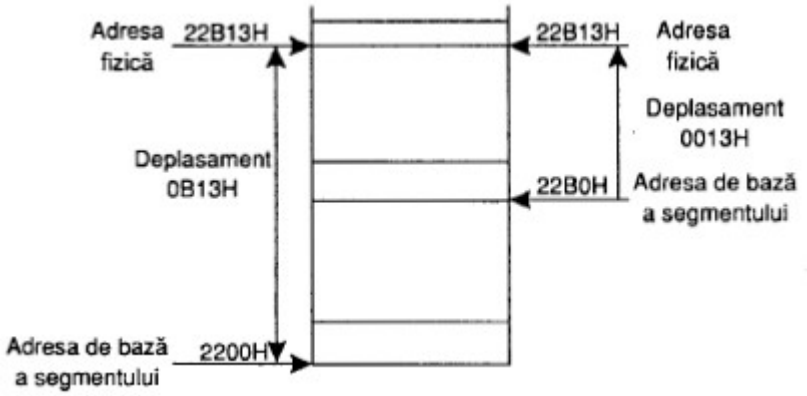
- IOPL (Input/Output Privilege Level) - acest indicator ocupă doi biți şi defineşte dreptul de a utiliza instrucțiuni de intrare/ieşire (I/O). Aceste instrucțiuni, precum şi instrucțiunile ce operează asupra lui IF, sunt denumite instrucțiuni „sensibile" la IOPL, deoarece ele nu pot fi executate decât dacă CPL <=IOPL - procedura care execută aceste instrucțiuni trebuie să se execute la un privilegiu cel puțin egal cu cel specificat de IOPL.
- NT (Nested Task) - este automat poziționat pe 1 sau O de operațiile de comutare de task; execuția unei instrucțiuni IRET, cu NT=1, realizează o comutare de task.

La procesorul 386/486 acest registru (EF) are 32 biți, dintre care ultimii 16 sunt identici cu aceştia, dar în plus mai sunt utilizați încă doi biți (16 şi 17) care au următoarea semnificație:

- RF (Resume Flag) - acest indicator dezactivează temporar excepțiile de depanare (debug), astfel încât să se poată reporni o instrucțiune după o excepție de depanare, fără a se genera imediat o altă excepție de depanare; adică nu se execută o instrucțiune de breakpoint, dacă este returnat controlul de excepție de depanare direct la o astfel de instrucțiune.

- VM (Virtual 8086 Mode) - acest bit arată că se execută un program 8086, dacă este poziționat pe 1.

15. Organizarea memoriei (modalitatea de memorare a datelor, instructiunilor, referirea la memorie-adresa logica).



Memoria este organizată ca seturi de segmente de lungime variabilă. Fiecare segment este o secvență continuă, liniară de până la 64 Kocteți (2^16). Adresa fizică de început a unui segment este multiplu de 16. Fiecare segment este alcătuit din locații succesive de memorie şi este o unitate independentă şi adresabilă separat. Fiecărui segment i se asociază o adresă de bază, care este adresa de start, în spațiul memoriei. Segmentele, în modul real, pot fi adiacente, disjuncte sau suprapuse total sau parțial. O locație de memorie poate fi conținută în mai multe segmente. Unei locații de memorie i se asociază o adresă fizică, de 20 sau 24 biți, care identifică în mod unic locația, şi o adresă logică (de fapt cu acestea lucrează programatorul) care nu este unică. Adresa logică se compune dintr-un selector (adică adresa de bază a segmentului) şi un deplasament față de începutul segmentului; practic aceeaşi adresă fizică poate fi referită de mai multe adrese logice: modificând în mod corespunzător deplasamentul, pentru o altă adresă de bază a segmentului se poate accesa aceeaşi locație de memorie, ca în figura.

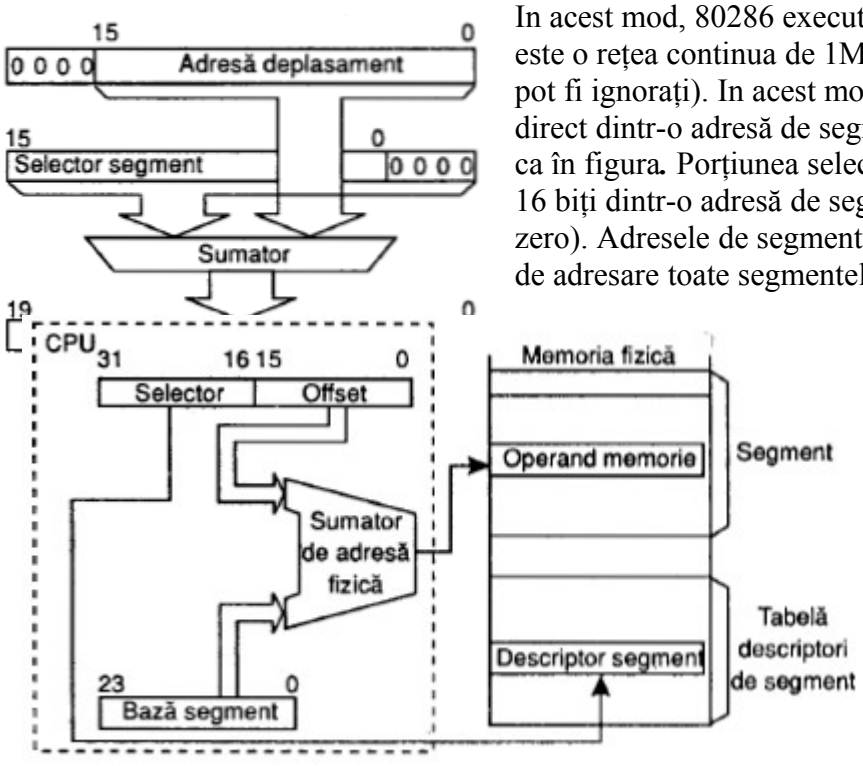
Din punct de vedere fizic, memoria este organizată sub forma a două blocuri: blocul superior (high) este conectat la magistrala de date pe liniile D_{15}-D_8, iar blocul inferior (low) conectat la liniile D_7-D_0. Ambele blocuri sunt selectate în paralel, pe liniile de adresă $A_{19}(A_{23})$-A_1; pentru accesul selectiv la octetul inferior, la cel superior sau la ambii octeți simultan, deci la un cuvânt, se activează în mod corespunzător liniile A_1 şi BHE\. Pentru procesoarele 386/486 poate fi considerat încă un bloc de memorie, conectat la liniile de date D_{31}-D_{16}, pentru a se putea citi şi cuvinte de 32 biți.

Pentru citirea/scrierea datelor de 16 sau 32 biți, memorate la adrese ce nu sunt multiplu de 2 sau 4, vor fi necesare două cicluri de magistrală, în loc de unul, dacă datele sunt aliniate în memorie.

Convenția de memorare a datelor multiplu de octeți, 16, 32, 64 sau 80 biți, este: se începe cu octetul mai puțin semnificativ la prima adresă (cea mai mică) şi se depun în continuare octeții următori, în ordinea crescândă a semnificației, terminând cu octetul cel mai semnificativ la adresele următoare.

O clasă specială de date este cea pe dublu cuvânt (32 biți), denumite şi pointeri sau referințe, utilizate pentru a adresa date şi cod. Şi acestea se memorează după aceeaşi regulă: la adresa mai mică se află deplasamentul (offset), iar la adresele următoare se află adresa de segment a pointerului.

16. Generarea adreselor fizice în modul real de adresare.



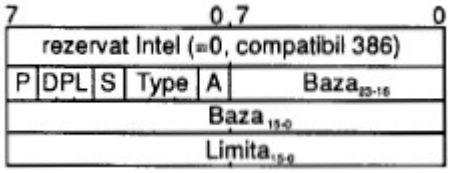
In acest mod, 80286 execută setul de instructiuni 8086. Memoria fizică este o rețea continua de 1Mo, adresabilă prin pinii A_0A_{19} şi BHE (A_{20}-A_{23} pot fi ignorați). In acest mod, procesorul generează o adresă de 20 biți, direct dintr-o adresă de segment de bază de 20 biți şi un offset de 16 biți, ca în figura. Porțiunea selectorului unui pointer este interpretată ca primii 16 biți dintr-o adresă de segment de 20 biți (ultimii patru sunt întotdeauna zero). Adresele de segment sunt întotdeauna multiplii de 16. în modul real de adresare toate segmentele au dimensiunea de 64 Ko. In modul real sunt rezervate două zone de memorie, şi anume: zona de inițializare a sistemului (FFFF0-FFFFFH) si zona tabelei de întreruperi (00000H-003FFH).

17. Generarea adreselor fizice în modul protejat de adresare virtuala.

Modul protejat utilizează pointeri de 32 biți, constând dintr-un selector de 16 biți şi un offset de 16 biți. Selectorul specifică un index

într-o tabelă rezidentă în memorie, în loc de primii 16 biți ai adresei reale de memorie. Adresa de bază de 24 biți a segmentului
dorit se obține de la tabelele din memorie. Deplasamentul se adună la adresa de bază a segmentului pentru a forma adresa fizică, ca în figura.

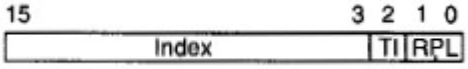
18. Structura unui descriptor de segment (câmpurile continute de acesta).



- P-(Present), dacă este 1, segmentul este mapat în memoria fizică altfel nu este mapat în memoria fizică, iar baza şi limita nu sunt utilizate;
- DPL - (Descriptor Privilege Level) este atributul de privilegiu, utilizat în teste de privilegiu;
- *S*- (Segment Descriptor) specifică tipul de segment definit de descriptor (1 este descriptor de segment de cod sau date; 0 descriptor de sistem)
- Type - specifică tipul segmentului: de cod sau de date; cei trei biți au următoarea
- semnificație:
  (3) Executable = 0 => un descriptor de segment de date şi, în acest caz
  - (2) Expansion Direction (ED) (0 => segmentul se extinde spre adrese mari (date; offset<=limita); 1 => segmentul se extinde spre adrese mici (stivă; offset > limita))
  - (1) Writeable (0 => nu se poate scrie în acel segment)

  (3) Executable = 1 => un descriptor de cod şi atunci semnificația biților următori este:
  - (2) Conforming (1 => acest segment poate fi executat numai când CPL >= DPL)
  - (1) Readable (0 => segmentul nu poate fi citit, adică este numai executabil, altfel el poate fi citit)
- A - (Accessed) (0 => segmentul nu a fost adresat; 1 => selectorul de segment a fost încărcat în registrul segment, sau utilizat de instrucțiuni de test selector)

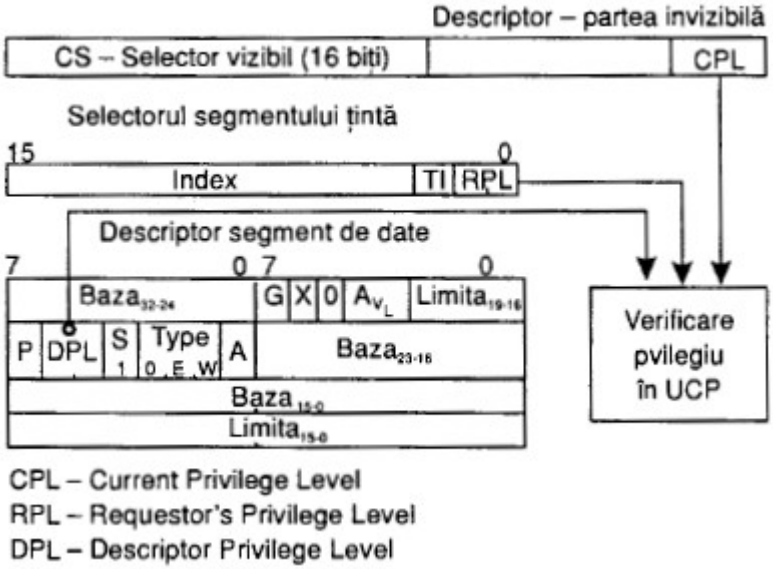
19. Formatul unui selector. Modul de calcul al spatiului virtual de memorie gestionat de procesor.



- RPL - (Requested Privilege Level) - nivelul de privilegiu dorit;
- TI - (Table Indicator) - acest bit face selecția între tabela de descriptor globală/locală.

Ceilalți biți (15-3) selectează intrarea respectivă din tabelă. Deci biții RPL, din câmpul 1+0, nu sunt implicați, de fapt, în selectarea şi specificarea segmentelor. Ceilalți 14 biți rămaşi din componența selectorului desemnează în mod unic un segment particular. Spațiul de adresă virtuală al unui program va fi format din cel mult 2^14 segmente distincte, întrucât dimensiunea maximă a unui segment este de 64 Kocteți (2^16), rezultă că spațiul virtual de adresă al unui program poate ajunge la 1 Goctet (2^14*2<6 = 230) de locații adresabile individual. La 386/486, întrucât dimensiunea unui segment este de 32 de biți, se ajunge la un spațiu total de adresă virtuală de 64 Tocteți (2^14*2^32 = 2^46).

20. Niveluri de privilegiu. Obiecte recunoscute de procesor ce contin niveluri de privilegiu.



Mecanismele de protecție ale procesorului 286 se bazează pe noțiunea de *ierarhie de încredere* (sau de privilegiu). Există patru niveluri de privilegiu, mergând de la 0 - cel mai privilegiat, până la 3 - cel mai puțin privilegiat. Următoarele obiecte, recunoscute de procesor, conțin niveluri de privilegiu:

- descriptorii de segment conțin un câmp DPL - Descriptor Privilege Level, adică descriptor de nivel de privilegiu;
- selectoarele de segment conțin un câmp RPL - Requestor's Privilege Level, adică nivelul de privilegiu al procedurii căreia îi aparține selectorul;

- un registru intern al procesorului memorează CPL - Current Privilege Level, care este egal cu DPL al segmentului pe care îl execută procesorul.

Instrucțiunile pot încărca un registru segment de date (şi deci utiliza segmentul țintă) numai dacă nivelul DPL al segmentului țintă este, numeric, mai mare sau egal cu maximul dintre nivelurile CPL şi RPL ale selectorului. Deci, procesorul preia nivelurile CPL şi RPL şi creează EPL = max(CPL,RPL), denumit nivel efectiv de privilegiu, adică nivelul cel mai puțin privilegiat din tre cele două; apoi acesta (EPL) se compară cu DPL.

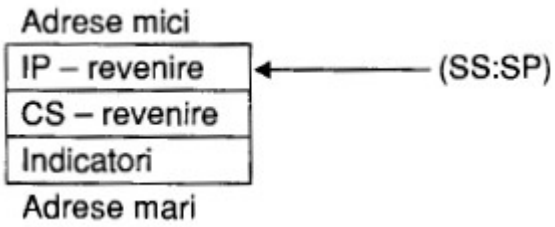
Domeniul adresabil al unui task depinde de valoarea lui CPL, şi anume pot fi adresate segmentele de date de la acelaşi privilegiu sau segmente mai puțin privilegiate (adică nivelul de privilegiu asociat segmentului adresat trebuie să fie, numeric, mai mare sau cel puțin egal cu cel al taskului curent), în acest fel, se poate preveni ca o procedură de aplicație să citească sau să modifice tabelele sistemului de operare.

21. Organizarea spatiului de intrare / iesire.

Procesorul are două spații de adrese fizice distincte: memoria şi I/O (intrări/ieşiri). Dispozitivele periferice (consola, ecranul, tastatura, imprimanta etc.) sunt controlate şi comandate de procesor, în general, prin operații de citire/scriere la anumite locații, diferite de spațiul de memorie. Acest spațiu este denumit spațiu de l/O şi accesul la locațiile respective se face cu instrucțiuni specifice (In, Out). Locațiile din acest spațiu sunt denumite registre dispozitiv sau porturi de l/O. In general, în acest spațiu sunt plasate perifericele, deşi procesorul poate permite maparea (suprapunerea) în memorie a perifericelor.

Spațiul de I/O constă din 64 Kocteți şi poate fi împărțit în 64 Kporturi de 8 biți, 32 Kporturi de 16 biți sau, la 386/486,16 Kporturi de 32 biți. Pentru a accesa acest spațiu nu se utilizează registre segment şi, deci, nici mecanismul de segmentare sau pagină. Pinul M/IO specifică spațiul de memorie adresat (fizică sau I/O), deci dacă se adresează o locație de memorie sau un port de I/O. Instrucțiunile I/O, IN şi respectiv OUT pot furniza adresa direct în instrucțiune, ca o constantă de 8 biți (pentru porturile din spațiul 0-255) sau indirect prin registrul DX (pentru tot spațiul de 64 K).

22. Sistemul de întreruperi. Tipuri de întreruperi.



O întrerupere transferă execuția la o nouă adresă de program. Vechea adresă (CS:IP) a programului şi starea maşinii (registrul de indicatori) sunt salvate în stivă pentru a permite refacerea programului întrerupt. Structura stivei după apariția unei întreruperi este următoarea (fig).

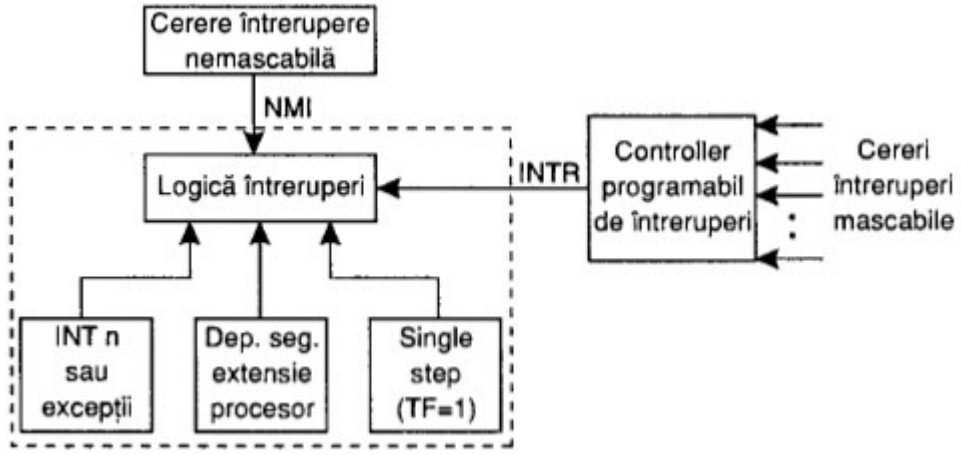
Intreruperile sunt împărțite în trei clase: inițiate de hardware, instrucțiunile de întrerupere INT şi excepții dejnstrucțiune. întreruperile inițiate hardware apar ca răspuns la un semnal extern şi sunt clasificate ca nemascabile şi mascabile. Programele pot determina o întrerupere prin execuția instrucțiunii INT. Excepțiile de instrucțiune apar când o condiție neobişnuită, care împiedică execuția ulterioară a instrucțiunii, este detectată în timp ce încearcă să execute o instrucțiune. Adresa returnată de o excepție va referi întotdeauna instrucțiunea ce a cauzat excepția şi include orice prefix de instrucțiune.

Prioritatile intreruperilor:

Ordin	Intrerupere
0	Instrucțiunea INT sau excepție
1	Intrerupere „Single Step"
2	Intrerupere nemascabilă (NMI)
3	Depăşire segment extensie de procesor
4	Intrerupere mascabilă (INTR)

Dacă apar mai multe întreruperi simultan, ele sunt servite în ordinea din tabelă. Dacă alte întreruperi rămân active, ele sunt prelucrate înainte de prima instrucțiune din rutina de tratare a întreruperii. Ultima întrerupere prelucrată va fi prima servită.

23. Reprezentarea schematica a sistemului de întreruperi.



24. Întreruperile externe.

Intreruperile externe, care pot fi transmise de dispozitive externe, sunt primite de procesor pe două linii: NMI şi INTR.

Linia INTR este controlată de PIC (Programmable Interrupt Controller); rolul său este de a accepta cereri de întrerupere de la dispozitivele externe, de a determina care cerere este cea mai prioritară şi de a activa linia INTR, dacă dispozitivul care a cerut întreruperea este mai prioritar decât cel curent în servire. Când linia INTR devine activă, procesorul execută diferite acțiuni, în funcție de starea indicatorului de validare a întreruperilor (IF); oricum, nu se execută nici o acțiune până nu se termină execuția instrucțiunii curente, pe durata căreia a apărut întreruperea. Dacă IF=0, deci întreruperile mascabile sunt dezactivate procesorul ignoră semnalul INTR. întreruperile se activează cu instrucțiunea ŞTI (SeT Interrupt enable flag), deci IF=1 şi se dezactivează cu instrucțiunea CLI (CLear Interrupt enable flag). Ele pot fi, de asemenea, mascate selectiv prin transmiterea de comenzi corespunzătoare către PIC.

Instrucțiunile de activare a întreruperilor se vor lansa după sfârşitul instrucțiunii următoare, pentru a nu încărca excesiv stiva, întreruperea este recunoscută prin executarea a două cicluri de magistrală de recunoaştere a întreruperii (INTA\); pe durata acestor două cicluri se activează semnalul LOCK\, pentru a indica altor procesoare că nu pot obține magistrala. Primul ciclu este utilizat de procesor pentru a transmite către PIC că se onorează cererea, iar în timpul celui de-al doilea ciclu, PIC răspunde plasând un octet pe magistrala de date, care conține tipul întreruperii (32+255), asociat cu servirea cererii de întrerupere. Tipul este asignat la inițializarea procesorului, prin program. Procesorul utilizează acest cod pentru a apela o procedură de tratare a întreruperii.

O cerere externă de întrerupere poate veni şi pe linia NMI, care e comandată pe frontul semnalului (INTR e comandată pe nivel) şi este utilizată pentru evenimente deosebite (cădere tensiune, repornire). întreruperile nemascabile, NMI, nu pot fi dezactivate şi au prioritate față de cele mascabile, INTR.

25. Întreruperile interne.

Intreruperile interne sunt evenimente sincrone şi sunt răspunsuri ale procesorului la anumite evenimente detectate în timpul execuției unei instrucțiuni, întreruperile externe (mascabile şi nemascabile) sunt evenimente asincrone. Deosebirea majoră între aceste două tipuri de întreruperi este originea lor: o întrerupere internă este întotdeauna reproductibilă prin reexecuția programului şi datelor ce au cauzat întreruperea, în timp ce o întrerupere externă este, în general, independentă de execuția curentă a taskului. Intreruperile interne sunt de două tipuri. Un tip de întrerupere este denumit excepție, deoarece întreruperea apare numai dacă există o anumită condiție de eroare, care nu permite execuția corespunzătoare a instrucțiunii; celălalt tip de întrerupere generează întrerupere ori de câte ori instrucțiunea INTn este executată. Aceste instrucțiuni sunt utilizate fie în scopul de a testa rutinele utilizatorului pentru tratarea întreruperilor externe, fie în scopul de a apela rutine DOS. Pentru apelul rutinelor DOS se utilizează acest mecanism şi nu mecanismul de apel de procedură din două motive: fie nu se cunosc aceste adrese, fie că ele se modifică de la o versiune la alta.

Exemple de excepții sunt: eroare de împărțire, depăşire detectată de instrucțiunea INTO, depăşire limite detectată de instrucțiunea BOUND, depăşire segment (overrun), cod de operație invalid, eroare de extensie de procesor etc. In modul protejat sunt detectate mult mai multe condiții de eroare, care rezultă într-o întrerupere internă, în acest mod de adresare virtuală, tabela de vectori de întrerupere nu are adrese fizice fixe şi deci nu poate fi adresată direct. De aceea, programele care, în modul real de adresare, manipulează direct tabela de vectori de întrerupere, nu vor lucra în modul protejat.

Celălalt tip de întrerupere generează întrerupere ori de câte ori instrucțiunea (INT?) este executată. Aceste întreruperi, instrucțiunile INT, sunt utilizate, de obicei, pentru testarea procedurilor de tratare a întreruperilor sau pentru a apela servicii DOS.

Intreruperile interne au următoarele caracteristici:

- codul întreruperii este conținut în instrucțiune sau este predefinit;
- nu se execută cicluri de recunoaştere întrerupere;
- nu pot fi dezactivate, cu excepția întreruperii „single step";
- sunt prioritare față de cele externe.

26. Întreruperile : " single step (pas cu pas)" si "breakpoint".

Intreruperea single step. Această întrerupere, de tipul 1, este generată în mod automat după execuția fiecărei instrucțiuni, dacă indicatorul TF este 1. Ea mai este denumită şi excepție de depanare deoarece permite execuția unui program instrucțiune cu instrucțiune (adică „pas cu pas"). Procesorul salvează în stivă registrul de indicatori şi adresa următoarei instrucțiuni de executat şi face indicatorii TF şi IF zero, astfel ca la intrarea în procedura de tratare a întreruperii single step, procesorul nu mai este în modul pas cu pas; se evită astfel recursivitatea infinită. Procedura va permite vizualizarea programului întrerupt (registre, indicatori, variabile memorie etc.); la sfârşitul procedurii se vor reface indicatorii din stivă, astfel că programul poate reveni în modul de execuție pas cu pas. Procesorul nu dispune de instrucțiuni directe pentru modificarea acestui indicator (TF); el poate fi modificat numai prin modificarea imaginii sale din stivă (utilizând instrucțiunile PUSHF, POPF).

Intreruperea de breakpoint (suspendare). Această instrucțiune, INT 3, este singura instrucțiune de întrerupere care are lungimea de un octet (toate celelalte ocupă câte doi octeți, primul fiind codul instrucțiunii, iar cel de-al doilea tipul întreruperii). Ea este folosită la implementarea de programe de depanare, deoarece necesită un singur octet de cod şi poate substitui foarte uşor orice octet de cod de instrucțiune. Ea permite în acest mod suspendarea programului şi executarea rutinei de tratare â întreruperii de pe acest nivel.

Instrucțiunea permite în plus şi inserarea de noi instrucțiuni în program, fără a fi recompilate sau reasamblate. Aceasta se poate face salvând primul octet de instrucțiune şi înlocuindu-l cu instrucțiunea INT 3. Procedura de tratare a întreruperii de breakpoint va conține noile instrucțiuni, plus codul pentru refacerea octetului de instrucțiune salvat (înlocuit anterior de INT 3) şi va decrementa adresa de revenire, salvată în stivă (mai exact IP) înainte de revenire, astfel ca instrucțiunea înlocuită să fie executată după instrucțiunile inserate.

Metoda aceasta de depanare a unui program este mai complicată decât utilizarea registrelor de depanare, deoarece necesită crearea unui alias pentru segmentul de cod, salvarea octetului instrucțiunii originale, înlocuirea lui cu INT 3 şi refacerea octetului la terminarea întreruperii.

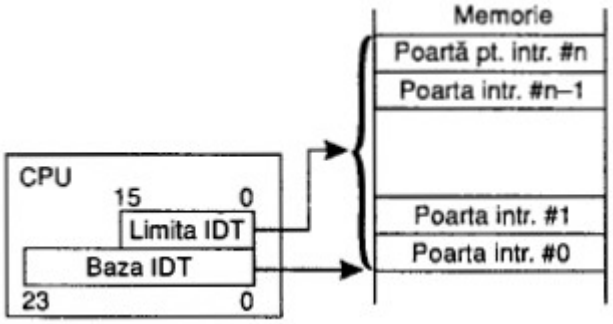
27. Tabela vectorilor de întrerupere.

Această tabelă reprezintă legătura dintre codul întreruperii şi procedura definită pentru a servi întreruperea asociată cu acel cod. Câțiva dintre vectorii cei mai prioritari sunt rezervați de INTEL pentru funcții dedicate. Utilizatorul nu poate folosi nici unul dintre primii 32 de vectori, în modul real, această tabelă ocupă 1 Koctet de memorie (pentru fiecare tip de întrerupere dintre cele 256 posibile este rezervat un pointer, adică un dublu cuvânt) care conține adresa procedurilor de tratare a întreruperilor. Adresa este memorată astfel: cuvântul cel mai semnificativ conține adresa de bază a segmentului, iar cuvântul mai puțin semnificativ conține deplasamentul; ele sunt memorate la adrese succesive în ordinea: cuvântul mai puțin semnificativ (deplasamentul) şi apoi cuvântul mai semnificativ (selectorul de segment).

Funcția	Număr întrerupere	Instrucțiuni legate de intr.	Returnează adr. instr. excepție
excepție eroare împărțire	0	DIV, IDIV	
întrerupere Single Step	1	toate	da
întrerupere NMI	2	toate	-
întrerupere de breakpoint	3	INT 3	-
excepție depăşire detectată de INTO	4	INTO	-
excepție depăşire domeniu instr. BOUND	5	BOUND	nu
excepție cod instr. invalid	6	orice cod nedefinit	da
excepție extensia procesor nu este	7	ESC sau WAIT	da
disponibilă	8	LIDT	da
limită tabelă într. prea mică	9	ESC	da
într. depăşire seg. extensie procesor	13	toate instr. cu	da
excepție depăşire segment	16	referire la mem	da
într. eroare extensie procesor	10-12, 14,	ESC sau WAIT	-

întreruperi rezervate	15, 17-31
întreruperi def. de utilizator	32-255

28. Întreruperile în mod protejat (IDT, diferente fata de modul real).



Pentru acest mod este definită o tabelă de descriptori de întrerupere IDT - (Interrupt Descriptor Table), care defineşte procedurile pentru toate cele 256 de întreruperi. Această tabelă este referită de un registru intern al procesorului IDTR - Interrupt Descriptor Table Register - care conține adresa de bază a tabelei (24 biți) şi limita acesteia (16 biți). Registrul IDTR este încărcat de instrucțiunea LIDT de codul executat pe nivelul O de privilegiu, decătre inițializarea sistemului. Tabela poate fi plasată oriunde în memorie (figura). Fiecare intrare în tabela IDT conține un descriptor de poartă (gate), format din 4 cuvinte (8 octeți), care conține referința la rutina respectivă. Sunt permise în tabela IDT doar trei tipuri de porți: întrerupere (interrupt gate), capcană (trap gate) şi task (task gate). Primele două prelucrează întreruperile în cadrul aceluiaşi task, în timp ce a treia realizează o comutare de task. Porțile task din IDT sunt identice cu cele din GDT (descriptorii sunt asemănători celor de sistem) şi operează în acelaşi mod. Diferența dintre primele două este că prima pune indicatorul IF pe zero (dezactivează întreruperile), în timp ce a doua lasă indicatorul nemodificat. IDT nu necesită toate cele 256 de intrări; registrul limită (16 biți) permite mai puțin decât numărul total de intrări.

Intrările neutilizate vor avea zero în octetul de drepturi de acces. Accesul în afara limitelor sau la un descriptor eronat va genera o eroare de protecție generală (GP), cu un cod de identificare a erorii (a vectorului IDT invalid), întreruperile 0-31 sunt rezervate de INTEL, astfel că limita IDT trebuie să fie cel puțin 255, pentru numărul minim de intrări.

O poartă task poate fi invocată de o întrerupere/excepție; starea maşinii se salvează în segmentul TSS curent şi se încarcă o nouă stare din segmentul TSS asociat porții task. Astfel, o întrerupere poate avea propriul său spațiu de adresă (LDT); In plus, procedura de tratare a intreruperii este impiedicata de utilizarea stivei aplicatiilor întrerupte şi de coruperea oricăror registre. O comutare de task durează mai mult decât un transfer de poartă, dar oferă avantajul separării față de taskul curent.

29. Structura procedurilor de tratare a întreruperilor.

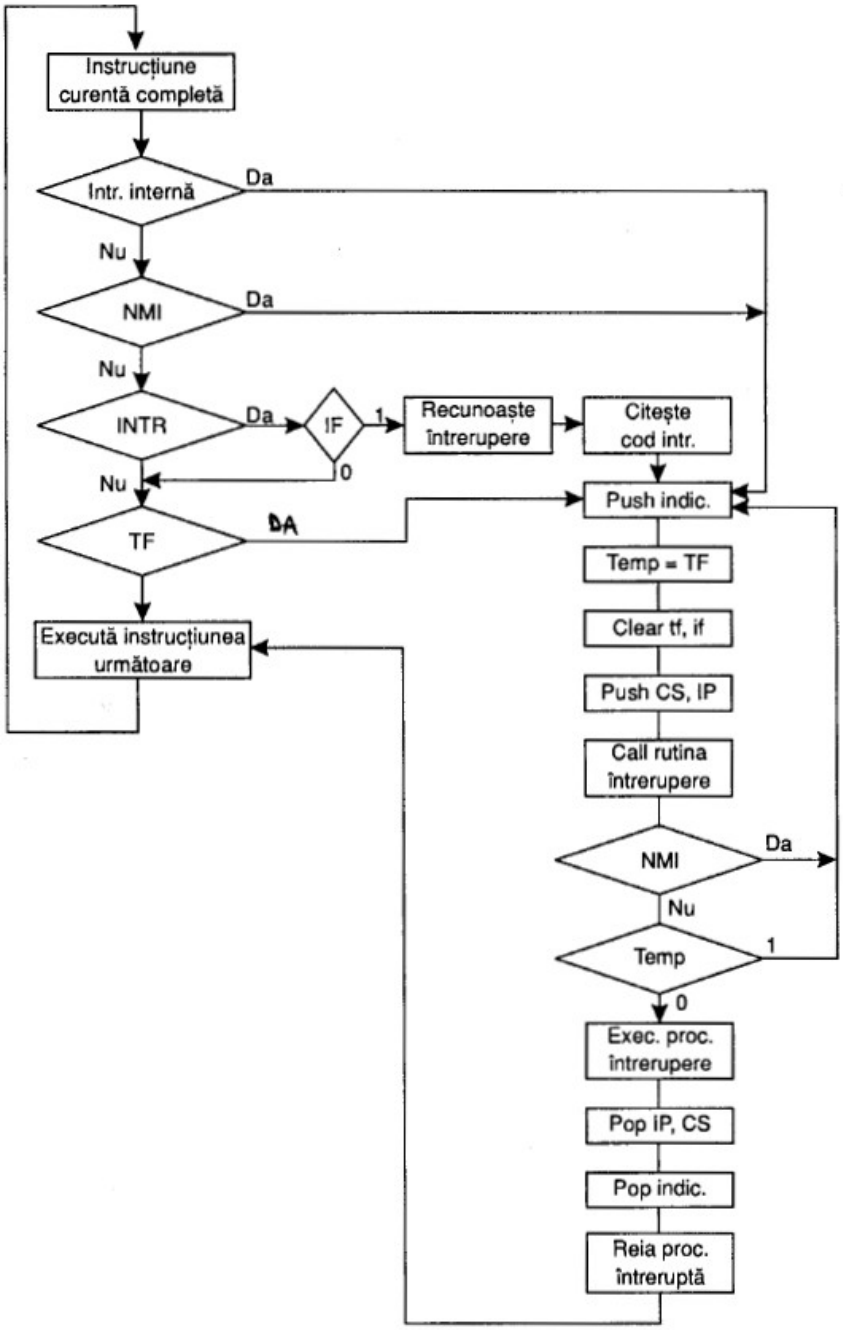
Procesorul serveşte întreruperile numai între sfârşitul unei instrucțiuni şi începutul următoarei. Dacă se utilizează un prefix de repetare a unei instrucțiuni pe şiruri, întreruperile şi excepțiile pot apărea între repetări. Când o procedură de tratare a întreruperii este executată, indicatorii şi registrele CS şi IP sunt puşi în stivă, iar TF şi IF sunt şterşi.

Procedura poate reactiva semnalele de întrerupere externe cu instrucțiunea ŞTI, permițând astfel să fie întreruptă de o cerere, pe linia INTR, mai prioritară decât cea curent servită. Procedura de întrerupere poate fi totuşi întreruptă de o cerere NMI, chiar dacă sistemul de întreruperi nu este activat.

Ca orice procedură, procedurile de tratare a întreruperilor trebuie să salveze registrele înainte de a le actualiza şi să le refacă înainte de terminare. De obicei, pe durata secțiunilor critice, adică cea de salvare şi refacere a stării programului întrerupt, întreruperile sunt dezactivate urmând a fi reactivate după aceste secțiuni. Dacă întreruperile sunt dezactivate un timp prea îndelungat, într-o procedură, cererile de întrerupere pe linia INTR pot fi pierdute.

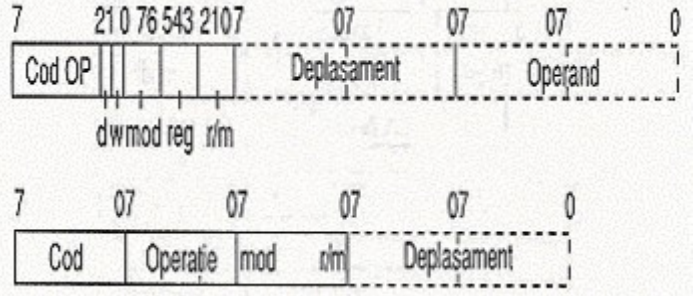
Procedura de tratare a întreruperii se termină cu instrucțiunea l RET (Interrupt RETurn); se presupune că stiva este adusă în aceleaşi condiții ca la apelul procedurii. Procedurile de întrerupere software pot fi avantajoase în sisteme care realocă dinamic programe în timpul execuției. De exemplu, în figura se arată cum sunt prelucrate întreruperile simultane (eroare la împărțire, întreruperea single step este activă şi apare pe durata execuției instrucțiunii de împărțire, cerere de întrerupere pe linia INTR).

30. Operatiile realizate de procesor la aparitia unei întreruperi.



31. Formate de instructiuni.

Instrucțiunile procesorului 80286 au o lungime variabila şi pot avea de la 1 până la 6 octeți.



Primul format – codul operației este reprezentat pe 1 octet
Al doilea format – codul operației este reprezentat pe 2 octeți
Semnificația biților:
d – direction – specifică direcția rezultatului operației
d = 0 r/m_r/m [Op] reg (registru)
1 reg_reg [Op] r/m (registru sau memorie)
w – word bit – indică tipul operanzilor:
w = 0, operand de tip octet;
w = 1, operand de tip cuvânt (2 octeți)

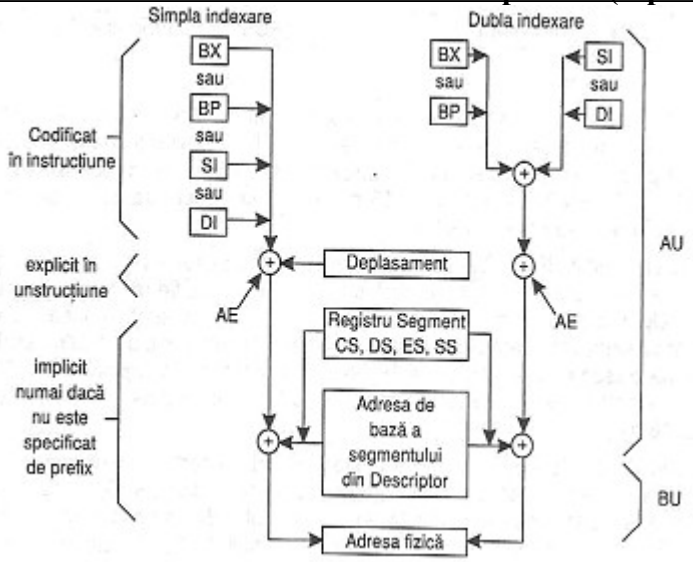
mod – modul de lucru
= 11 – r/m se referă la un registru
= 00 – câmpul de
= 01 – deplasament de tip octet
= 10 – deplasamentul are 16 biți

32. Moduri de adresare (enumerare, exemple de instructiuni).

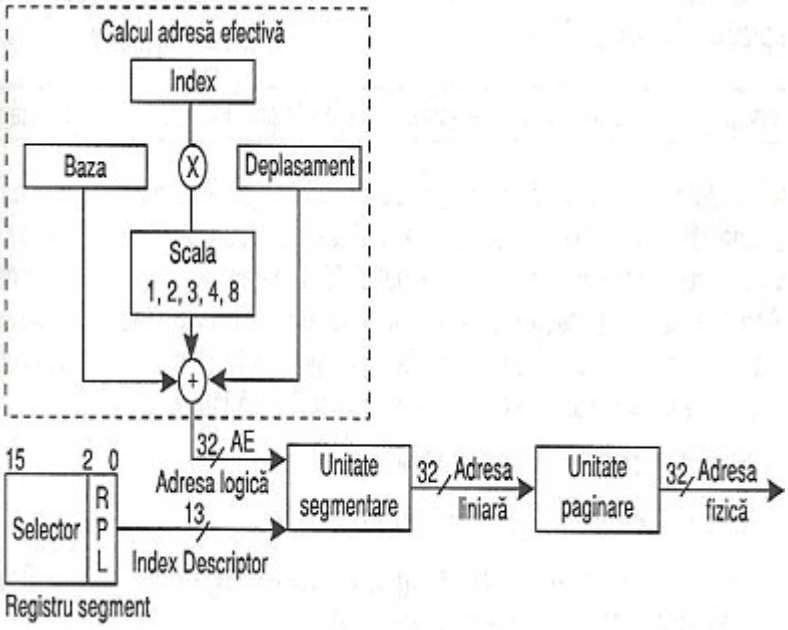
Există în principal şase moduri de adresare:
- directă, adresa efectivă (AE) a operandului este reprezentată de deplasamentul conținut în instrucțiune;
- indirectă, AE este într-unul dintre registrele de bază sau index;
- bazată, AE este suma dintre deplasament şi conținutul unui registru de bază (BX sau BP);
- indexată, AE este suma dintre deplasament şi conținutul unui registru index (SI sau DI);
- bazată şi indexată, AE este suma dintre conținutul a două registre: unul de bază şi unul index;
- bazată şi indexată cu deplasament, AE se obține ca sumă între un registru de bază, un registru index şi un deplasament;

Mai pot fi puse în evidență şi modurile de adresare:
- imediată, operandul este conținut în instrucțiune;
- la registre, operandul se află într-un registru;
- adresarea şirurilor;
- adresarea porturilor de I/O.

33. Modul de calcul al adresei fizice a unui operand (reprezentare schematica).



34. Modul de transformare a adresei logice în adresa fizica.



35. Sursele (registrele segment) pe baza carora se calculeaza adresa fizica, în functie de modul de adresare (de tipul referintei).

Adresa fizică determinată de AU din adresa logică se calculează din surse diferite în funcție de tipul referinței:

Tip referință la memorie	Segment de bază		Offset
	Implicit	Alternant	
Fetch instrucț.	CS	-	IP
Operații stivă	SS	-	SP
Variabile mem.	DS	CS, ES, DS	Adr. Efect.
Operatii şiruri - şir sursă - şir destinație	DS ES	CS, ES, SS -	SI DI
BP utilizat ca registru bază	SS	CS, ES, SS	Adr. Efect.

36. Tipuri de prefixe.

Pentru adresarea într-un alt segment decât cel implicit, se va preceda instructiunea respectiva cu un prefix de segment (override) care va specifica, în mod explicit, registrul segment utilizat. în ceea ce priveste formatul instructiunilor, la procesoarele 386/486 el este asemanator:

Cod operație	mod reg r/m	SI B	Deplasament	Operand
1-2octeți	0-1 octeți	0-1 octeți	0,1,2,4 octeți	0,1,2,4 octeți

în plus fata de formatul anterior, poate fi prezent în instructiune octetul SIB (Scale * Index + Base), care specifica registrul index utilizat (acesta poate fi oricare dintre registrele EAX, 120 Programare în limbaj de asamblare EBX, ECX, EDX, EBP, ESI sau EDI, deci nu poate fi utilizat ca registru index ESP), factorul de scalare cu care se înmulteste registrul index (acesta poate avea una dintre valorile 1,2, 4, 8) si registrul de baza (oricare dintre registrele EAX EDI) utilizat pentru calculul adresei efective. Daca în cazul procesorului 286 o instructiune nu poate fi precedata decât de prefixe de segment (care specifica registrul de segment utilizat pentru determinarea adresei fizice) sau prefixe de instructiune (de repetare sau de blocare a accesului altui procesor la magistrala sa), instructiunile procesorului 386/486 pot fi precedate de urmatoarele prefixe (de câte un octet fiecare): [prefix instructiune [prefix dimensiune adresa [prefix dimensiune operand | prefix segment]

37. Descrierea (reprezentarea schematica) unor moduri de adresare; exemple de instructiuni si situatii de utilizare a acestora.

Adresarea directă nu implică nici un registru, adresa efectivă este specificată chiar în codul instrucțiunii, prin deplasament.

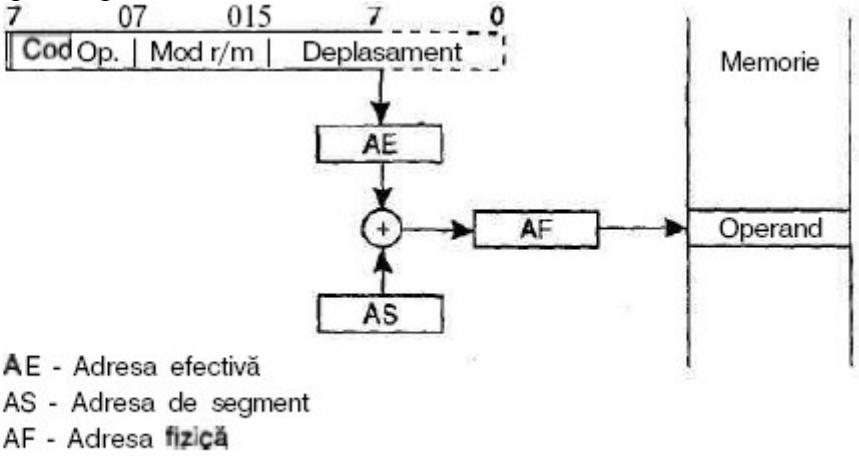


Figura 7.7 Adresarea directă

Exemple de instrucțiuni:

mov ax,adr_w ; adr_w - adresa de cuvânt

mov adr_w[2],si ; transfer la adresa adr_w + 2

Adresarea indirectă prin registre face referire la memorie prin intermediul registrelor de index sau de bază care vor conține, în acest caz, adresa efectivă a operandului.

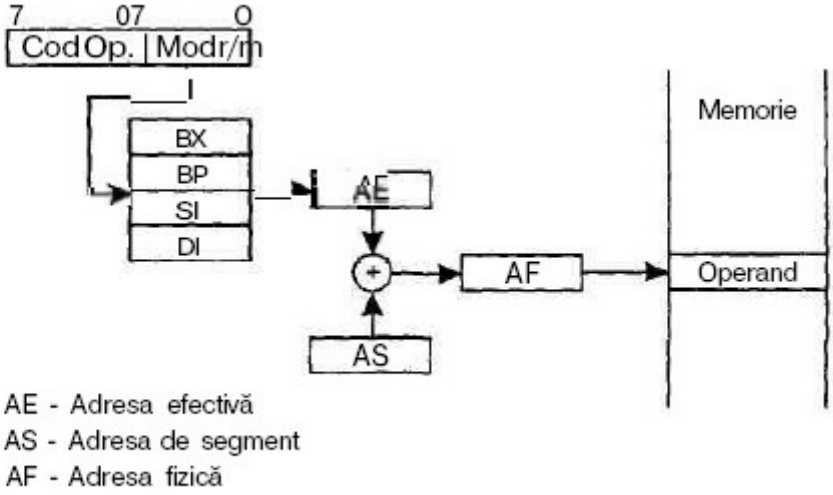


figura 7.8 Adresarea indirectă, prin registre

Sintaxa instrucțiunilor în limbajul de asamblare utilizează pentru adresarea indirectă operatorul index [].

Exemple de instrucțiuni:

mov ax,[bx]

mov bx, [si]

La procesorul 286 pot fi utilizate doar registrele index şi de bază pentru adresarea indirectă prin registre.

Adresarea bazată determină adresa efectivă adunând conținutul unui registru de bază cu deplasamentul din instrucțiune.

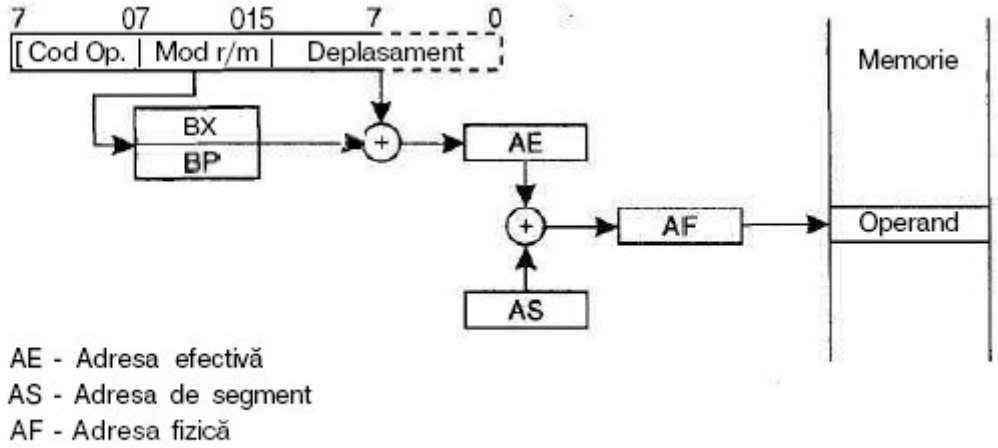


Figura 7.9. Adresarea bazată

Dacă se utilizează registrul BP ca registru de bază, atunci se pot obține operanzi din segmentul curent de stivă, referit de SS (dacă nu este prezent un alt prefix segment). Astfel pot fi referite datele din stivă, fără a descărca stiva.

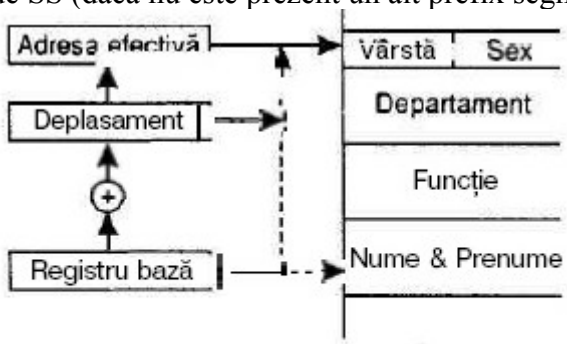


Figura 7.10 Adresarea elementelor unei structuri

Adresarea bazată furnizează, de asemenea, un mod convenabil de a adresa structuri de date similare, care pot fi memorate la adrese diferite de memorie, în acest caz, registrul de bază se va referi la baza structurii, iar elementele structurii vor fi adresate prin deplasamentul lor față de bază. Aceleaşi câmpuri ale structurii pot fi accesibile prin simpla modificare a registrului de bază, cu adresa de început a noii structuri. Prin modificarea doar a deplasamentului se pot referi alte câmpuri ale structurii (figura *7.10).*

Exemple de instrucțiuni:

mov ax,depl[bx]

mov ax,[depl + bx]

mov ax,[bx] + depl

Adresarea indexată este asemănătoare cu cea bazată, întrucât adresa efectivă se obține tot ca o sumă între un registru, de această dată index SI sau Dl, și deplasamentul din instrucțiune.

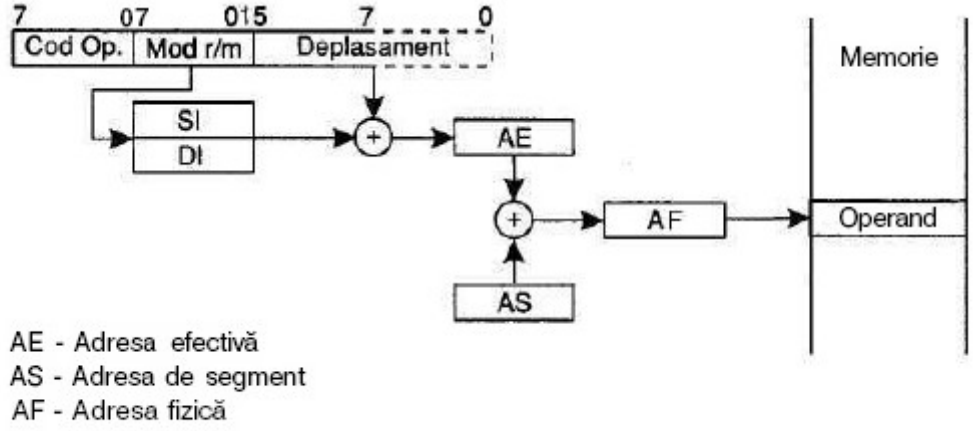


Figura 7.11 Adresarea indexată

Acest mod de adresare este utilizat pentru referirea elementelor unui vector. Deplasamentul va marca începutul vectorului, iar registrul index va selecta elementul, prin poziția sa relativă în cadrul vectorului. Deoarece toate elementele vectorului sunt de aceeaşi

lungime, prin operații aritmetice elementare asupra registrului index se va selecta orice element; din acest motiv, la procesoarele 386/486 se poate specifica un factor de scală (1,2,4,8) pentru index, pentru a referi vectori cu componente de lungime fixă de 1, 2, 4, 8 octeți.
Adresarea la registre, în acest caz, adresa efectivă a operandului este adresa unui registru general, adică operandul este într-un registru.

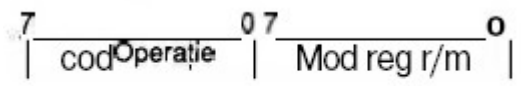


Figura 7.16 *Adresarea la registre*

Exemple de instrucțiuni:
mov ax, si;
mov ah, ci;
mov ds,bx

38. Identificatori în limbajul de asamblare (standard si definiti de utilizator).

Identificatorii sunt secvențe de lungime 1-^31 caractere alfanumerice şi speciale: ? _ @ $. într-un identificator nu este permis spațiul liber. Un identificator trebuie să înceapă cu o literă sau cu unul dintre caracterele speciale. Dacă identificatorul începe cu punct (.), atunci el are semnificația de comandă adresată asamblorului. Asamblorul nu deosebeşte literele mari de cele mici. Identificatorii sunt şi ei de două tipuri: standard (sau predefiniți) şi definiți de utilizator. Identificatorii standard, denumiți şi cuvinte cheie, sunt folosiți pentru denumirea unor funcții şi argumente stabilite la definirea limbajului şi fac parte din vocabularul limbajului. Astfel de identificatori sunt:
- nume de instrucțiuni: MOV, ADD, SUB, INT etc.;
- nume de resurse: AX, BX,.., CS, DS,.., AH, AL etc.;
- nume de operatori: MOD, OFFSET, SEG, PTR, TYPE etc.;
- pseudoinstrucțiuni pentru asamblor: ASSUME, MODEL etc.

Identificatorii definiți de utilizator vor fi folosiți pentru:
- nume de variabile, adrese de instrucțiuni, operanzi;
- nume de proceduri, segmente, macroinstrucțiuni.

39. Tipuri de constante si reprezentarea lor.

Constantele pot fi numerice sau şiruri de caractere, formate pe baza anumitor reguli şi cărora li se asociază o valoare. Constantele numerice pot fi: întregi, reale sau zecimale (codificate binar). Constantele întregi reprezintă numere întregi; ele pot fi utilizate, în funcție de context, pentru: date, adrese sau operanzi imediați. Constantele pot fi reprezentate în binar, octal, zecimal sau hexazecimal; reprezentarea în una dintre aceste baze se specifică printr-una din literele B, Q, D sau H, care urmează constanta: -binar: 0110101OB; -octal: 152Q; -zecimal: 106D sau 106; -hexazecimal: 6aH; Pentru a evita confuzia între constantele hexazecimale şi identificatori, o constantă hexazecimală trebuie să înceapă întotdeauna cu o cifră (de exemplu: OabcdH, pentru a nu fi confundată cu identificatorul abcdH). Constantele zecimale pot fi urmate sau nu de litera D, deoarece se consideră implicit baza 10. Există o directivă de asamblare care permite stabilirea reprezentării într-o bază implicită a constantelor: `.RADIX < expresie >` unde < expresie > trebuie să aibă o valoare întreagă (2, 8,10 sau 16) şi este evaluată în zecimal, lată de exemplu ce valori se încarcă în registrul AX după utilizarea acestei directive (am folosit constante pentru expresia bazei): `.RADIX 2`

```
(AX) = 0004H mov AX,100;
.RADIX 8
mov AX,100;
.RADIX 10
mov AX,100;
.RADIX 16
mov AX,100; (AX) = 0100H
(AX) = 0040H
(AX) = 0064H
```

Constantele întregi pot fi reprezentate în memorie pe 8,16, 32 sau 64 de biți. Constantele reale sunt reprezentate în virgulă mobilă, prin mantisă şi exponent, în general sub forma: [±] întreg, fracție [E ± exponent]

Asamblorul codifică aceste constante reale în reprezentarea cu virgulă mobilă pe 32, 64 sau 80 de biți, după tipul constantei:

dd (define double word), pe 32 de biți, real simplă precizie;

dq (define quadruple word), 64 de biți, real dublă precizie;

dt (define ten bytes), 80 de biți, real temporar;

Codificarea în virgulă mobilă diferă de la un asamblor la altul, de la un compilator la altul, dar formatul cel mai răspândit este cel IEEE:

exponent mantisă

• S_m- semnul mantisei (1 bit);
• exponent- este exponentul numărului deplasat cu o anumită valoare, în funcție de tipul reprezentării:
7fH, pentru formatul pe 32 biți;
3ffH, pentru formatul pe 64 biți;
3fffH, pentru formatul pe 80 biți;
Exponentul pentru cele trei formate se reprezintă pe 8,11 şi respectiv 15 biți;
• mantisa este normalizată, adică:
1.00...O <= mantisa <=1.11...1

şi se reprezintă, pentru cele trei formate, pe 23, 52 şi respectiv 64 de biți; de remarcat că pentru primele două formate nu se reprezintă prima cifră, deci prima unitate din fața virgulei, care este presupusă implicit, cu ajustarea corespunzătoare a exponentului. Pentru cel de-al treilea format se reprezintă şi prima unitate, cea din fața virgulei (punctului zecimal). Exponentul din reprezentarea internă este deplasat (biased), pentru a simplifica operațiile de comparare cu numere reale; în acest mod, procesorul lucrează numai cu valori pozitive pentru exponent. De exemplu, pentru valori reale în simplă precizie (32 de biți), exponentul este în intervalul [1,254], în loc de [-127, +126] cât arfi fost domeniul exponentului pentru o reprezentare cu semn. Exponentul minim (0) este utilizat doar pentru reprezentarea valorii 0.0 şi pentru valori nenormalizate, iar cel maxim (255) este utilizat pentru a reprezenta o condiție de eroare (NaN - Not a Number), adică o codificare cu exponent pentru 11 ...11 nu reprezintă o valoare numerică. Numerele negative diferă de numerele pozitive numai prin bitul de semn al mantisei. Să considerăm, de exemplu reprezentarea câtorva numere; pentru simplitatea reprezentării am ales valori care se reprezintă exact. Pentru o declarație de forma

a) dd 1.0 ; 2^0

deci

S_m = 0;

exponent = O + 7f (deplasarea);

mantisa = 1.00...0;

se obține următoarea reprezentare internă:

S_m, exp, mant = 0, 01111111, 00...0 = 3F 80 00 00 H

b) dd –1.0 ; -2^0

rezultă

S_m, exp, mant = 1, 01111111, 00...0 = BF 80 00 00 H

c) dd 4.0 ; 2^2

rezultă

exponent = 2 + 7f = 81 H;

S_m, exp, mant = O, 10000001, 00...0 = 40 80 00 00 H

d) dd –0.625 ; $2^{-1} + 2^{-2}$;

rezultă

exponent = -1 + 7f = 7e;

mantisa = 1.0100...0;

S_m, exp, mant = 1, 01111110, 010...0 = BF 20 00 00 H

Valorile sunt memorate în ordinea inversă scrierii lor, adică la adresa cea mai mică, octetul cel mai puțin semnificativ, iar la adresa cea mai mare (ultima), octetul cel mai semnificativ. Un alt tip de constante îl reprezintă constantele zecimale codificate binar (BCD); ele pot fi de două tipuri, şi anume: - zecimal împachetat; - zecimal neîmpachetat (denumit şi format ASCII). Primul format se caracterizează prin faptul că fiecare dintre cele două tetrade ale unui octet conține o cifră zecimală, deci valoarea unui astfel de octet este cuprinsă în intervalul 0-99; cel de-al doilea format conține numai o singură cifră, pe tetrada mai puțin semnificativă, cealaltă fiind O, ca în reprezentarea următoare:

Aceste constante pot fi reprezentate şi pe 80 de biți utilizând o declarație de tipul:
dt 12345678
pentru care se pot introduce maxim 18 cifre, deoarece octetul cel mai semnificativ conține pe primul bit semnul numărului, restul acestui octet (7 biți) fiind nefolosit. Memorarea se face după aceeaşi regulă, adică începând cu octetul cel mai puțin semnificativ şi terminând cu octetul cel mai semnificativ. Formatul zecimal

neîmpachetat mai este denumit şi format ASCII, deoarece există o corespondență directă între formatul BCD neîmpachetat şi cel ASCII (singura diferență la acest format față de cel BCD este că prima tetrada conține valoarea 3, bineînțeles, pentru cifre). Memorarea acestui număr se face, conform regulii cunoscute, începând cu octetul cel mai puțin semnificativ, la adresa adr si terminând cu octetul cel mai semnificativ, la adresa adr + 9.

De exemplu, pentru o declarație de forma:
dt 12345678
reprezentarea internă va fi:
00 00 00 00 00 00 12 34 56 78.

Constantele exprimate prin şiruri de caractere constau dintr-o succesiune de caractere incluse între apostrofuri ('Exemplu') sau între ghilimele ("şir caractere"). Dacă în interiorul unui şir de caractere, un subşir trebuie să apară între apostrofuri sau ghilimele, atunci aceste caractere se dublează: '1 Dec"1918'sau"1 Dec""1918"

40. Tipuri de propozitii în limbaj de asamblare; sintaxa unei instructiuni.

O propoziție este o succesiune de cuvinte şi caractere din alfabetul limbajului de asamblare, construită după anumite reguli sintactice, având lungimea maximă de 128 de caractere şi care poate fi:
- instrucțiune;
- pseudoinstrucțiune.

O instrucțiune reprezintă o propoziție care este tradusă în cod maşină. O pseudoinstrucțiune este o propoziție care nu se traduce în cod maşină, ea reprezentând o directivă pentru asamblor, pentru coordonarea modului de lucru al asamblorului:
- definiri de constante;
- rezervări de zone de memorie;
- controlul procesului de asamblare;

Instrucțiuni în limbaj de asamblare
Sintaxa generală a unei instrucțiuni este:
[eticheta:] [mnemonica [operanzi]] [; < comentarii >]
unde eticheta - este un câmp care conține un nume simbolic format din maximum 31 de caractere alfanumerice şi caracterele speciale _, ?, @, $. Primul caracter este o literă sau un caracter special. Fiecare etichetă are asociată o valoare numerică ce reprezintă adresa relativă în cadrul segmentului (sau grupului de segmente) din care face parte; mnemonica- reprezintă numele instrucțiunii sau al pseudoinstrucțiunii; operanzi - este un câmp a cărui existență şi format depinde de tipul instrucțiunii. Dacă instrucțiunea are doi operanzi, cei doi sunt denumiți destinație şi sursă; ei apar în instrucțiune în această ordine. Operanzii pot fi şi expresii care conțin registre, constante, identificatori şi operatori (aritmetici, de indexare, logici, de deplasare, relaționali, de tip sau de conversie).

41. Declararea datelor în limbaj de asamblare, exemple.

Declararea datelor se realizează cu ajutorul unor pseudoinstrucțiuni care asigură: alocarea de memorie pentru date, specificarea tipului datelor şi inițializarea datelor. Datele pot fi specificate prin nume, expresii sau şiruri de caractere, care se evaluează în procesul de asamblare la constante. Operația de alocare şi inițializare a datelor are loc în procesul de asamblare; datele sunt înscrise în fişierul obiect, de unde vor fi preluate la execuție.
Sintaxa este următoarea:
[nume_variabila]tip[lista_expresii] [;comentarii]
sau
[nume_variabila]tip[numar] DUP ([lista_expresii])
unde
- *nume_varibila* este un identificator, construit după regulile specificate anterior;
- *tip* este mnemonica pseudoinstrucțiunii, care implicit specifică tipul datelor:

db - define byte; defineşte date de tip octet întregi, cu sau fără semn;
dw - define word; cuvânt de 16 biți ce defineşte date întregi, cu sau fără semn;
dd - define double; dublu cuvânt, de 32 de biți ce poate defini un pointer sau o valoare întreagă sau reală, cu sau fără semn;
dq - define quadruple; cuvânt de 64 biți ce poate defini o valoare reală sau întreagă;
df - define float; real pe 6 octeți;
dp - define pointer; referință specifică pentru 386, reprezentată pe 6 octeți (segmentul pe 2 octeți, iar deplasamentul pe 4 octeți);
dt - define ten bytes; dată de 10 octeți de tip rea! sau zecimal împachetat;
nume_structură - numele unei structuri utilizate pentru definirea unor date cu această

structură;
• *listă_expresii* sunt expresii separate prin virgule, evaluate la constante, reprezentând valorile cu care vor fi inițializate datele definite.
Dacă apare simbolul ?, înseamnă că datele respective nu se inițializează. Construcția este recursivă. Evaluarea expresiilor se face la asamblare.
• număr reprezintă o expresie ce reprezintă factorul de multiplicare al listei de expresii care urmează după operatorul DUP.
Exemple:
```
adb -100 ; se pot defini valori in intervalul [-128, +127]
; sau [O, 255]
lit db 'abcde' ; se vor memora codurile ASCII ale caracterelor din
; lista: 61 h, 62h, 63h, 64h, 65h
aw dw -1000 ; domeniul de valori: [-32768, 32767] sau [O, 65535]
AB dw 'AB' ; va depune la adresa AB: 42h, 41 h
adrAB dw AB ; se va depune deplasamentul etichetei AB,
; stabilit la asamblare
off_AB dw offset AB ; deplasamentul etichetei AB, stabilit la localizare
seg_AB dw seg AB ; adresa segment de la localizare
rez dw ? ; conținut nedefinit
v1 dd 1.0 ; valoare reala definita pe 32 biți
v2 dd 1A2B3C4Dh ; valoare intreaga pe 32 biți
adr_v1 dd v1 ; pointer pentru referirea lui v1
```
Pentru a multiplica definirea unor date se utilizează operatorul DUP, alături de db, dw etc.
Sintaxa este următoarea:
< număr > DUP (expresie pt. valoarea inițiala)
în acest fel se definește o structură formată dintr-un <număr> de elemente, iar un element poate fi: expresie numerică, semnul ? dacă nu se specifică valoarea inițială, o listă de elemente sau alți operatori DUP.
Exemple:
```
db 100 dup (0) ; multiplicarea valorilor inițiale
db2 dup (O, 3 dup (1)), 10, 25, 100)
dw 100 dup (5 dup (4), 7)
db 10 dup ('mesaj ', Odh, Oah)
```

42. Operatorii SEG, OFFSET, TYPE.

SEG < expresie >. Acest operator furnizează adresa de segment a unei adrese exprimată printr-o expresie, care poate fi: variabilă, operand memorie, etichetă, numele unui segment sau numele unui grup de segmente.
Exemplu:
```
beta dw
mov bx, seg beta
mov ds,bx
```
OFFSET < expresie >. Operatorul furnizează deplasamentul în cadrul segmentului pentru o expresie de tip: variabilă (simplă sau structurată), operand memorie, etichetă. Aplicat unei variabile sau etichete, operatorul furnizează deplasamentul variabilei sau etichetei. Valoarea este cunoscută în momentul editării legăturilor, când se face alinierea finală a segmentului, deoarece deplasamentele de la asamblare se pot modifica la localizare, dacă segmentul este combinat cu părți ale aceluiași segment, definite în alte module sau dacă nu este aliniat la un paragraf. **Exemplu:** el poate fi utilizat în adresarea indirectă a variabilelor
```
şir dw
mov bx, offset şir ; adresa de început a şirului
mov si, O ; se inițializează indexul primului
; element din şir
add ax, [bx] [si] ; se insumeaza elementele
```
Dacă se utilizează directiva *group,* operatorul OFFSET nu va returna deplasamentul unei variabile din grup, ci va returna deplasamentul variabilei din segmentul său. Pentru a returna deplasamentul variabilei din grupul respectiv, trebuie utilizat prefixul grupului respectiv. **Exemplu:**
```
dgrup group datai, data2
datai segment
```

datai ends
data2 segment
val db
dw val; deplasamentul in cadrul segmentului ; furnizat de asamblor
dw dgrup:val ; deplasamentului in grup
Limbajul de asamblare 135
dd val; deplasamentul in segment + adresa, segment
dddr gup:val; deplasamentul grup + adresa segment
dataSends
mov bx, offset val; deplasamentul in segment
mov bx, offset grup:val; deplasamentul in grup
Observație:
- *offset [bx]* este echivalent cu *[bx]*
- *offset depl[bx]* este echivalent cu *[depl+bx]*

TYPE < expresie >. Furnizează un număr întreg ce reprezintă tipul expresiei asupra căreia se aplică; are un singur argument, care poate fi o variabilă sau etichetă. Dacă expresia este o variabilă (simplă sau structurată) operatorul returnează numărul de octeți pe care se reprezintă componentele variabilei simple sau numărul de octeți ai variabilei structurate. Valorile returnate de operator, în funcție de tipul variabilei sunt: byte- 1;
word- 2;
dword- 4;
qword- 8;
pword- 6;
fword- 6;
tbyte- 10; ~ ~~
structura - numărul de octeți ai structurii;
Se utilizează, de obicei, în calcule asupra vectorilor sau structurilor, pentru a determina adresa următorului element. Dacă argumentul este de tip etichetă, operatorul, în funcție de tipul etichetei, returnează valorile: near- ffffH;
far- fffeH;
Exemple:
var dw
mov bx, type var ; (BX) = 2
vectorddIO dup (?)
mov si, type vector ; (SI) = 4
num_BCD dt 13245768,
mov bx.type num_BCD; (BX) = 10

43. Operatorii LENGTH, SIZE si PTR.

LENGTH < expresie >. Acest operator furnizează o valoare întreagă ce reprezintă numărul de elemente ale unei variabile (declarată utilizând operatorul DUP). Dacă variabila a fost declarată prin utilizarea operatorului DUP imbricat (unul inclus într-altul), atunci operatorul LENGTH va returna o valoare asociată operatorului exterior. Dacă variabila nu a fost declarată utilizând operatorul DUP, atunci operatorul LENGTH va returna valoarea 1. **Exemple:**
n1 db 50 dup (?);
n2 dw 150 dup (O, 1, ?);
n3 dd 200 dup (10, 20, 15 dup (?));
mes db1Exemplu de mesaj:';
length n1 = 50;
length n2 = 150;
length n3 = 200;
length raes = 1;

SIZE < expresie >. Furnizează o valoare ce reprezintă numărul de octeți ocupați de o variabilă şi este în relație cu LENGTH şi TYPE prin identitatea:
SIZE = LENGTH * TYPE
Dacă o variabilă a fost declarată utilizând operatorul DUP imbricat, SIZE furnizează, conform relației anterioare, numai valoarea operatorului DUP exterior. Pentru declarațiile anterioare, acest operator va furniza următoarele valori:
size n1 = 50;
size n2 = 300;

size n3 = 800;
size mes = 1;
PTR < expresie >. Acest operator realizează conversia de tip pentru o variabilă, operand din memorie sau etichetă, fie pentru a specifica tipul adresării, fie pentru a permite adresarea unor date cu un alt tip decât cel cu care au fost declarate. Sintaxa acestui operator este: tip PTR < expresie > unde *tip* poate fi: BYTE, WORD, DWORD, QWORD, PWORD, FWORD, TBYTE sau nume de structură pentru operanzi din memorie şi variabile, sau NEAR, FAR, PROC, unknown (lasă la latitudinea asamblorului determinarea tipului FAR sau NEAR utilizat) pentru etichete de instrucţiuni, nume de proceduri. De fapt, acest operator asociază următoarele atribute expresiei: tip expresie
variabila sau eticheta
expresie numerică
atribute asociate
segment
SEG <expr>
nedefinit
offset
OFFSET <EXPR>
<expresie>
type
tip
tip
Exemple:
a) Explicitarea tipului referinţei la memorie, pentru referinţe anonime:
inc byte ptr [bx]
inc word ptr [si]
mov byte ptr [bx],99
mov word ptr [di],100
and word ptr [bp],100h
b) Specificarea tipului de salt (intersegment sau intrasegment):
jmp dword ptr [bx]
}mp near ptr etich ; pentru etich declarata in alt modul,
; daca aceasta eticheta este in acelaşi segment sau grup
; de segmente cu instrucţiunea de salt
c) Referirea unor variabile cu un alt tip decât cel cu care au fost declarate:
aw dw OabcH, 12abH
ab db OaH, 12H
mov al, byte ptr aw
mov ax, word ptr ab
Limbajul de asamblare 137
d) Crearea unei variabile anonime la un offset dat dintr-un segment:
mov al,ds: byte ptr 5 ; refera octetul de la ds:5
mov bx,datai: word ptr 3000H ; refera cuvântul de la deplasamentul 3000H din
; segmentul datai
Aceste două declaraţii sunt echivalente cu următoarele două:
mov al,ds:[5]
mov bx,datai:[3000]

44.Operatorii SHORT, WIDTH, MASK.

SHORT. Acest operator accepta un argument de tip eticheta (un offset adresabil prin registrul segment CS).Se utilizeaza in instructiuni de salt conditionat, neconditionat si in instructiuni de apel de procedura, cand codul tinta are un deplasament autorelativ de un octet cu semn (adica tinta saltului este in intervalul –128 ->127 fata de instructiunea de salt). In mod normal, instructiunea de salt va codifica deplasamentul la instructiunea tinta ca un numar intreg pe 16 biti. Utilizand acest operator, cu conditia ca tinta sa fie in intervalul [-128,+127], asamblorul va codifica deplasamentul tintei pe un singur octet, reducand codul generat cu un octet:
jmp SHORT etich.
De fapt asamblorul genereaza in mod automat un deplasament pe 8 biti, daca deplasamentul auto relativ al tintei este in intervalul [-128,+127], dar pt referinte externe va genera un deplasament pe 16 biti. Daca referinta este in

intervalul mentionat, utilizand operatorul Short se va genera un deplasament de un octet, scurtand codul generat pt instructiunea de salt cu un octet.
WIDTH si MASK. Acesti operatori unt folositi pt a returna numarul de biti sau o masca de biti pt inregistrare, care reprezinta un tip de date structurat. Operatorul WIDTH returneaza numatul de biti ai unei inregistrari sau al unui camp al unei inregistrari, dupa cum argumentul sau este numele inregistrarii sau numele unui camp al inregistrarii. Operatorul MASK are ca argument numele unui camp al unei inregistrari si returneaza o masca de biti, definiti ca fiind 1 pt pozitiile campului respectiv si 0 pt celelalte pozitii. Exemplu:

```
model record A:3,B:1,C:4,D:5,E:3   ;reprezinta numerele asociate campurilor respective, impreuna cu numarul
                                   ;  de biti pt fiecare camp
mov cx, mask C     ; (CX)=000 0 1111 00000 000=0f00H
mov cl, width D    ; (CL)=5, dimensiunea campului D
mov cl, C          ;(CX)=8, pozitia campului C
```

45. Directivele ORG, EQU si simbolul contor program $.

Directiva EQU. Aceasta directiva permite asignarea unei valori la un simbol, in momentul asamblarii. Sintaxa este:
<nume> EQU <expresie>
unde expresia poate fi: constanta, orice expresie corecta in limbaj de asamblare, simbol definit anterior, sir de caractere cu diferite semnificatii (referinta indexata, operator prefix de segment si operanzii sai, nume de instructiuni). Diferenta dintre EQU si = este ca numele simbolice ce folosesc EQU nu isi pot modifica valoarea pe durata asamblarii programului.
Directiva ORG. Contorul de locatii poate fi initializat prin aceasta directiva cu o anumita valoare pozitiva astfel:
ORG <expresie>
unde expresia este evaluata *modulo(64K)* si nu poate contine referinte ulterioare. Nu poate fi asignata la o eticheta, adica o declaratie de forma: *start: ORG 100 H*, sua pentru date.
Simbolul contor program $ (contor locatie). Reprezinta adresa curenta a contorului de program, adica adresa relativa in cadrul segmentului curent al instructiunii sau a datelor de asamblat. La inceputul fiecarui segment, contorul de locatii este initializat la 0 si e actualizat de asamblor pe masura asamblarii programului. Exemple:

```
 lista dw 11, 22, 33, 0aah
lung_lista equ ($ linie-lista)/2
mesaj db ‘ mesaj $ ‘
lung_mesaj equ $ - mesaj
```

46. Definirea completa a segmentelor.

Se utilizeaza o declaratie de forma:

```
nume_seg SEGMENT [tip_aliniere][tip_combinare][tip_utilizare][clasa_seg]
….
….<corpul segmentului>     ;instructiuni sau date
….
nume_seg ENDS
```

unde nume_seg este numele asociat segmentului, care trebuie sa fie unic si caruia i se asociaza o adresa de segment (de 16 biti), corespunzatoare pozitiei in memoria segmentului, in faza de executie a programului.
Initializarea unui registru segment(DS,ES,SS) cu un segment declarat revine utilizatorului, care o va face in mod explicit in cadrul programului, utilizand pt aceasta numele segmentului respectiv:

```
mov ax,<nume_segment>
mov ds, ax                    ; in mod asemanator pt ES sau SS
```

47.Definirea simplificata a segmentelor.

.MODEL tip_model
Prin aceasta directiva se specifica dimensiunea si modul de dispunere a segmentelor in memoria RAM. Modelele de implementare, tip_model, sunt urmatoarele:

- tiny – pt care LP + LD + LS < un segment (64K)
- small – pt LP < un segment (64K), LD + LS < un segment
- medium – pt LP > un segment (64K), LD + LS < un segment
- compact - pt LP < un segment (64K), LD + LS > un segment
- large – pt LP > un segment (64K), LD + LS >un segment
- huge – la fel cu modelul anterior, cu diferenta ca referintele sunt normalizate.

Abrevierile utilizate sunt: LP, LD si LS; ele reprezinta lungimea programului (codului), dimensiunea memoriei pt date si respectiv pt stiva.

48.Asocierea segm cu registrele segm. Pseudoinstr. ASSUME.

Deoarece intr-un program pot exista mai multe segmente de diferite tipuri, asamblorul trebuie informat in leg cu cele 4 segm logice curente, lucru realizat cu pseudoinstr ASSUME:

ASSUME<reg_segment>:

<reg_segment> - unul din registrii CS,DS,ES sau SS; - numele unui segm/grup de segm ce va fi adresat de registrul segm respectiv sau de operatorul SEG si numele unei variabile/etichete din segm respectiv:

assume cs: code, ds: dgrup, es: SEG var_a, ss: SEG varf_stiva

Pt anularea asocierii: - una din formulele:

assume <reg_seg>: NOTHING ; anuleaza
; asocierea reg_seg la orice segm

assume NOTHING ; anuleaza asocierile facute,
; pt toate registrele

49.Initializarea registrelor segment.

Registrele CS:IP – initializate pt executie de sist de operare la incarcarea programului executabil in memorie a i ele sa contina adresa primei instr ce trebuie executata din program:

END [adresa_de _start]

<adresa_de_start> - expresie ce specifica prima instr executabila si care va initializa continutul registrului IP. Adresa de start este optionala (poate lipsi pt unele module de program ce cont date sau proceduri).

CS – initializat de SO, cu prima adresa de segment disponibila; IP – initializat cu deplasamentul primei instr executabile in cadrul segm. Pt SS:SP – initializare facuta de: - utilizator, dc se folosesc directive complete de segmentare si nu se specifica tipul si clasa segm de stiva; - automat (SP – initializat cu dimens segm de stiva declarata in directiva *.stack* sau rezervata in declararea completa a segm de stiva). DS, ES etc – initializate explicit si intotdeauna de programator.

50. Prefix segment.Referinte anonime.

Dc o referinta la o variabila nu este acoperita de o directiva ASSUME, asamblorul poate fi informat explicit asupra registrului segment al variabilei prin codificarea unui prefix de segment de forma: *reg_seg: instructiune (referinta)* in fata variabilei respective. Initializarea e facuta de utilizator.

Dezavantaje: - e valabila doar pt o instructiune (prefixul de segment trebuie precizat pt fiecare referinta la o variabila) - e foarte usor sa se greseasca pt ca se refera la un registru de segment si nu la un nume de segment.

Variabilele referite astfel: [bx], [bp], word ptr [si], byte ptr [di], [bx].camp, sunt denumite *referinte anonime*, pt ca nu este precizat numele variabilei din care sa se poata det segmentul caruia ii apartin. In astfel de situatii registrele de segment utilizate pt adresare sunt det in mod implicit pe baza unor reguli.

51.Reguli pt det registrului segment implicit.

Se refera la cazurile in care in instr. nu apar referiri explicite la registrele de segm sau nu apar nume de variabile din care sa reiasa registrul de segm care trebuie utilizat. Registrele de segm utilizate implicit sunt:

-pt registrele de baza: SS pt BP, DS pt. BX;

-pt.registrele index: DS pt ambele registre index (DI, SI). Dc instr. foloseste si un registru de baza si unul index, atunci se foloseste regula coresp. registrului de baza. Instr. care lucreaza cu stiva (push,pop,call,ret,int,iret) utilizeaza SS:SP si nu pot fi prefixate. Instr. pe siruri utilizeaza pt sirul destinatie registrul segm ES. Referintele anonime ce nu fol. registrul segm DS pt sirul sursa necesita specificarea explicita a reg. de segm utilizat si tipul operanzilor:

movs es:byte ptr [di],ss:[si]
movs word ptr es:[di],ss:[si]

52.Definirea si utilizarea grupurilor de segmente.

<nume_grup> GROUP <lista_nume_segmente>

<nume_grup> - numele grupului de segm pt det adresei de sgm utilizata pt referirea in cadrul grupului de sgm. Utilizat pt.: a initializa un registru de segment, intr-o pseudoinstr. ASSUME:

```
assume  ds:dgrup
        mov ax,dgrup
        mov ds,ax
```

ca prefix de operand, pt a specifica utilizarea valorii de baza a grupului sau deplasamentul in cadrul acestuia:

mov ax,offset dgrup:var1 ; deplasamentul in grup
dd dgrup:var2 ; adresa in cadrul grupului

<lista_nume_segmente> - nume de segmente sau expresii de forma: SEG nume_variabila, SEG nume_eticheta

53. Setul de instructiuni; setul de instructiuni de baza.

Setul de instructiuni e impartit in 3 subseturi distincte: setul instructiunilor de baza, setul extins de instructiuni (instr pt I/O din proceduri, BOUND, I/O pe bloc), setul de instructiuni pentru controlul de sistem (controlul mecanismelor de administrare, protectia memoriei virtuale).

8086(88) instruct. de baza

80186 set extins de instruct.

80286 instruct. control sistem

80386 instruct. pe 32 biti

80486 instruct. in virgula mobila

PENTIUM instruct. MMX (date/64 biti)

Setul de instr.de baza: instr. de transfer de date, aritmetice, operare pe bit (logice,deplasare, rotire), operare pe siruri, transfer al controlului de program, pt controlul de procesor.

54.Formatul datelor aritmetice.

4 tipuri de numere:

-binare: fara semn si cu semn

-zecimale: neimpachetate si impachetate (ambele fara semn)

Nr binare pot fi de 8 sau 16 biti (32 biti la 386). Nr zecimale – memorate in octeti: 2 cifre/octet la formatul zecimal impachetat, o cifra/octet – cel neimpachetat. Intervale: -nr binare fara sgn: 0..255 octet, 0..65535 cuvant, 0..2^32-1 cuvant dublu; -cele cu sgn – in cod complementar (fata de 2). Pt nr fara sgn, depasirea e detectata de indicatorul CF, pt nr cu sgn depasirea e detectata de OF. Pt a det depasirea – instructiuni de salt conditionat pt CF/OF dupa operatia respectiva.