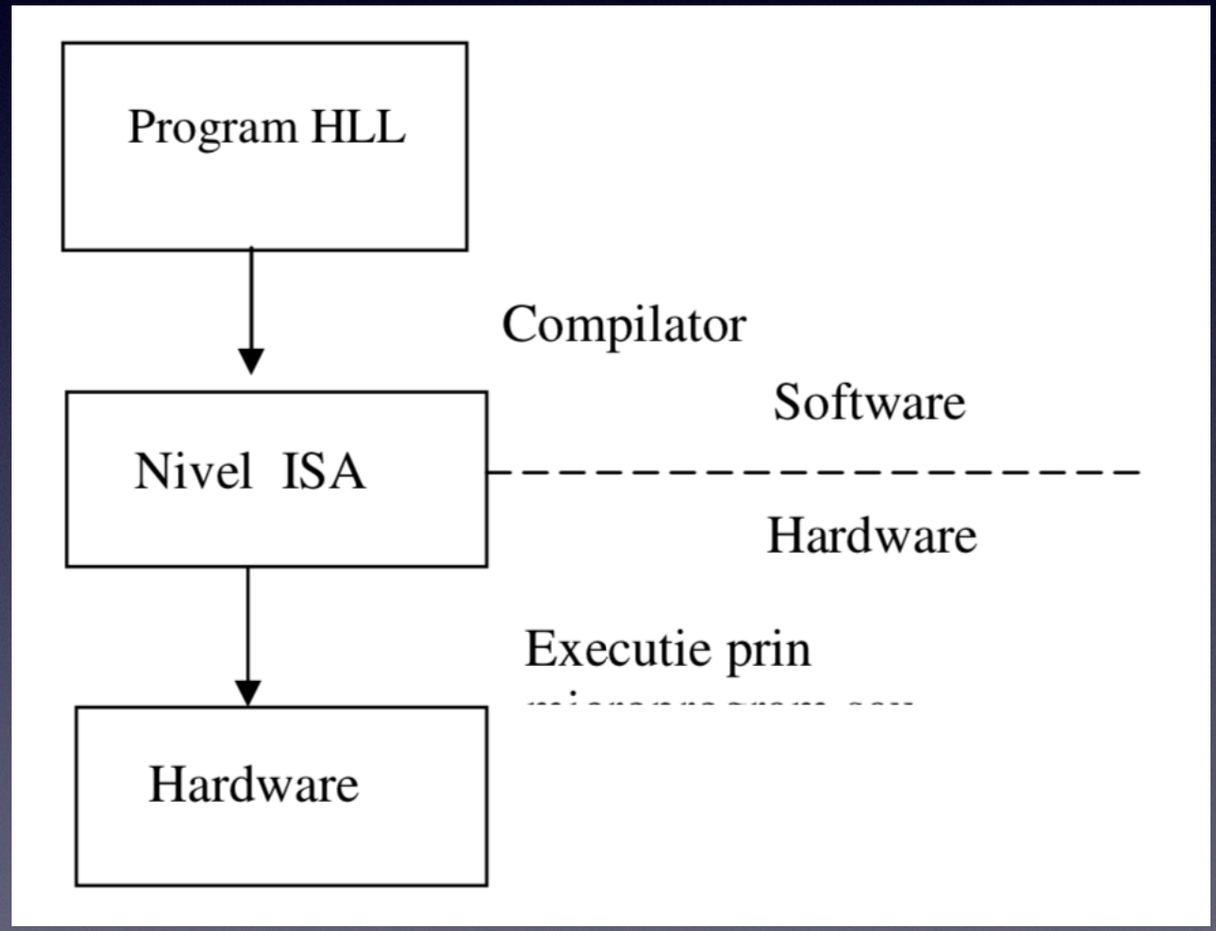
Nivelul arhitecturii setului de instrucțiuni ISA

Curs #6

Nivelul arhitecturii setului de instructiuni = ISA (Instruction Set Architecture)

- Acest nivel se află între nivelul microarhitecturii şi cel al sistemului de operare.
- El reprezintă interfaţa dintre hardware şi software.



Compilarea

- Un compilator „traduce” un program scris într-un limbaj de nivel înalt în cod maşină pentru a putea fi executat de CPU
- A nu se confunda un compilator cu un interpretor
- La procesoarele simple era preferabilă scrierea programelor direct în cod maşină deoarece se obţineau executabile mai mici şi mai rapide decât cele furnizate de compilatoare
- Programele şi procesoarele de astăzi sunt mult prea complexe pentru a putea scrie, întreţine şi obţine eficienţa maximă a codului maşină

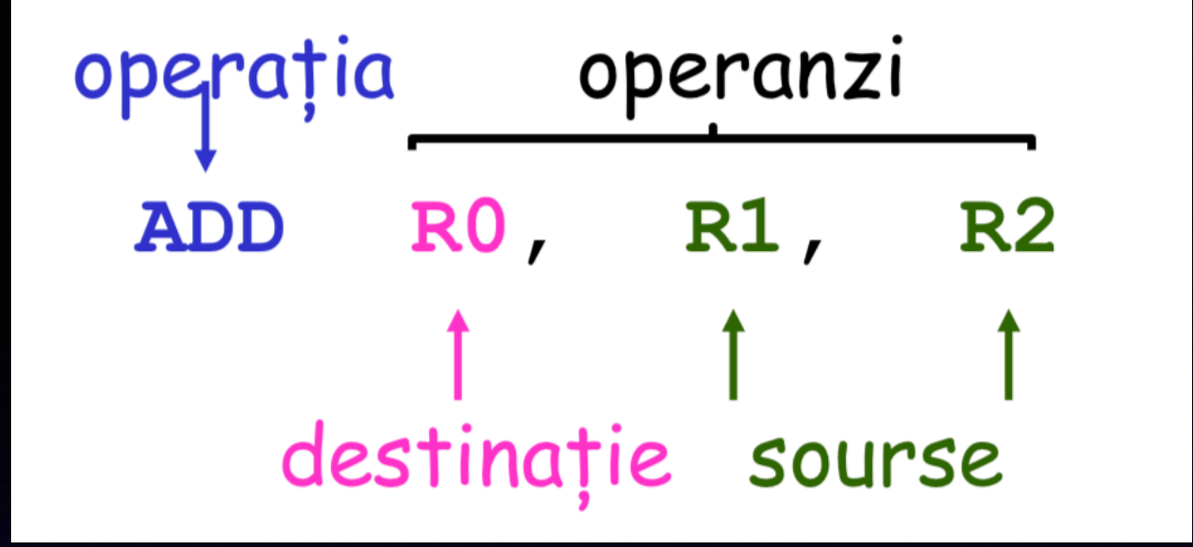
Limbaj de asamblare şi limbaj maşină

- Instrucţiunile în limbaj maşină sunt o succesiune de biţi (specifică fiecărei instrucţiuni)
- În programare se utilizează limbajul de asamblare:
  - se asociază „nume” operaţiilor şi operanzilor
  - există o corespondenţă biunivocă (aproape) între aceste nume şi instrucţiunile cod maşină (specifice procesorului) şi astfel programele scrise în limbaj de asamblare sunt portabile
- Traducerea între limbajul de asamblare şi limbajul maşină, şi invers, este realizată de asambloare

Setul de instructiuni al microprocesorului x86

1. instrucţiuni pentru transferuri de date
2. instrucţiuni aritmetice
3. instrucţiuni logice
4. instrucţiuni pentru manipularea şirurilor de caractere
5. instrucţiuni pentru controlul transferului programului
6. instrucţiuni pentru controlul procesorului

Sintaxă instrucțiuni



- Operanzii destinaţie pot fi:
  - regiştri ai microprocesorului (*reg*)
  - locaţii de memorie (*mem*)

- Operanzii sursă pot fi:
  - regiştri ai microprocesorului (*reg*)
  - locaţii de memorie (*mem*)
  - date constante (*data*)

1. Instrucţiuni pentru transferuri de date

- Se împart în patru clase:

a) cu scop general

b) specifice cu acumulatorul

c) cu obiect adresă

d) referitoare la indicatorii de condiţie

1.a. Instrucţiuni pentru transferuri de date cu scop general

MOV

Transferă o valoare din sursă în destinaţie (realizează o copie a sursei în destinaţie) - reprezintă instrucţiunea de **atribuire** în limbaj de asamblare

MOV dest, sursa	(Move data - Transfera date)
descriere formală:	dest ← sursa
descriere:	se copiază operandul sursă în operandul destinație; operanzii pot fi de tip byte sau word
operanzi:	reg, reg reg, mem mem, reg reg, data mem, data

PUSH

PUSH sursa;	(Push data - Salveaza date în stivă)
descriere formală:	(SP) ← (SP) - 2 SS: ((SP)+1) ← sursa (high) SS: ((SP)) ← sursa (low)
descriere:	se decrementează registrul SP cu 2 şi se copiază operandul sursă (word) în memoria stivă, cu octetul mai semnificativ la adresa mai mare, în vârful stivei;
operanzi:	registru general de 16 biți, registru segment; locație de memorie de 16 biți;

POP

POP dest;	(Pop data - Restaurează date din stivă)
descriere formală:	dest (high) ← SS: ((SP)+1) dest (low) ← SS: ((SP)) (SP) ← (SP) + 2
descriere:	se copiază octeții din stivă de la adresele (SP)+1 şi (SP) în destinație şi apoi se incrementează registrul SP cu 2.
operanzi:	registru general de 16 biți, registru segment; locație de memorie de 16 biți;

XCHG

XCHG dest, sursa;	(Exchange - Schimbă reciproc)
descriere formală:	dest ← sursa sursa ← dest.
descriere:	se transferă conținutul sursei în destinație şi reciproc; registrele segment nu pot apărea ca operanzi; cel puțin un operand trebuie să fie registru.
operanzi:	reg. - reg. reg. - mem. mem. - reg.

Exemple

```
MOV BX, CX
MOV CL, AL
MOV [1200], AX
MOV byte ptr [BX+200], 9A ;transfer imediat în memorie
```


```
MOV BX, CX
MOV CL, AL
MOV [1200], AX
MOV byte ptr [BX+200], 9A ;transfer imediat în memorie
```

1.b. Instrucţiuni specifice cu acumulatorul

- În această clasă intră instrucţiunile de transfer de la/la porturile de intrare/ieşire.
- Un operand se află întotdeauna în registrul acumulator, iar celălalt reprezintă portul.
- Dacă portul este pe 8 biţi se foloseşte registrul AL, iar dacă portul este pe 16 biţi se foloseşte registrul AX.

```
in ac, port ;citeşte un caracter în registrul acumulator de
la portul specificat.


out port, ac ;scrie un caracter din registrul acumulator în
portul specificat.
```

1.c. Instrucţiuni cu obiect adresă

- In acestă categorie intră instrucţiunile de încărcare a regiştrilor segment.

```
lds reg, mem32 ;incarcă registrul reg cu cuvântul de la
adresa de memorie mem32, iar registrul segment DS cu
valoarea de la locaţia mem32+2
```

```
les reg, mem32 ;incarcă registrul segment ES
```

- Aceste instrucţiuni sunt folosite pentru setarea pointerilor de tip far.

- O altă instrucţiune din acestă clasă este **`lea reg, mem`** care încarcă în registrul specificat adresa efectivă de memorie *mem*.

1.d. Transferuri de flaguri

În acestă categorie intră instrucţiunile:

- **`lahf`** – încarcă registrul AH cu conţinutul octetului mai puţin semnificativ al registrului de stare;
- **`sahf`** - încarcă octetul mai puţin semnificativ al registrului de stare cu conţinutul registrului AH;
- **`pushf`** - salvează în stivă registrul de stare
- **`popf`** - încarcă registrul de stare cu valoarea din vârful stivei.

2. Instrucțiuni aritmetice

Instrucţiunile aritmetice lucrează cu patru tipuri de date:

- binare fară semn (numere reprezentate, în memorie, binar în mărime);
- binare cu semn (numere reprezentate, în memorie, în mărime şi semn în complement faţă de doi);
- BCD neîmpachetate fară semn (un număr zecimal reprezentat pe un octet);
- BCD împachetate cu semn (două numere zecimal reprezentate pe un octet);

2. Instrucțiuni aritmetice

Instrucţiunile aritmetice se împart în:

- instrucţiuni pentru adunare
- instrucţiuni pentru scădere
- instrucţiuni pentru înmulţire
- instrucţiuni pentru împărţire

- Instrucţiunile aritmetice afectează indicatorii de condiţie din registrul de stare.

2.a. Instructiuni pentru adunare

- **`add dest, sursă`** - realizează operaţia dest+sursa cu depunere în dest
- **`adc dest, sursă`** - realizează operaţia dest+sursa+CF cu depunere în dest, unde CF=carry flag
- **`inc dest`** - realizează operaţia dest+1 cu depunere în dest
- **`aaa`** – (ASCII Adjust for Addition) realizează o corecţie ASCII la adunare a rezultatului stocat în registrul AL după o operaţie de adunare ASCII
- **`daa`** - (Decimal Adjust for Addition) realizează o corecţie BCD la adunare a rezultatului stocat în registrul AL după o operaţie de adunare BCD

2.b. Instructiuni pentru scadere

- **`sub dest, sursă`** - realizează operaţia dest-sursa cu depunere în dest
- **`subb dest, sursă`** - realizează operaţia dest-sursa-CF cu depunere în dest, unde CF=carry flag
- **`dec dest`** - realizează operaţia dest-1 cu depunere în dest
- **`neg dest`** - realizează operaţia 0-dest cu depunere în dest
- **`cmp dest, sursă`** - compară operanzii dest şi sursă prin scădere (dest-sursă), poziţionează corespunzător indicatorii de condiţie din registrul de stare fără să modifice destinaţia.
- **`aas`** – (ASCII Adjust for Substraction) realizează o corecţie ASCII la scădere a rezultatului stocat în registrul AL după o operaţie de scădere ASCII
- **`das`** - (Decimal Adjust for Substraction) realizează o corecţie BCD la scădere a rezultatului stocat în registrul AL după o operaţie de scădere BCD.

2.c. Instrucţiuni pentru înmulţire

- In general un operand se află în registrul acumulator (AX sau AL), iar rezultatul se încarcă tot în registrul acumulator (AX) şi în registrul DX partea mai semnificativă dacă operanzii sunt de doi octeţi.

- **`mul dest, sursă`** - realizează operaţia dest*sursa fără semn cu depunere în dest. Dest este AL sau AX

- **`imul dest, sursă`** - realizează operaţia dest*sursa cu semn cu depunere în dest. Dest este AL sau AX

- **`aam`** – (ASCII Adjust for Multiplication) realizează o corecţie ASCII la înmulţire a rezultatului stocat în registrul A după o operaţie de înmulţire ASCII

2.d. Instrucţiuni pentru împărţire

- Deîmpărţitul se află se află în registrul acumulator AX sau în AX şi DX iar rezultatul se încarcă în registrul acumulator (AL, AX). Restul împărţirii se întoarce în AH sau DX
- **`div dest, sursă`** - realizează operaţia dest/sursa fără semn cu depunere în dest.
- **`idiv dest, sursă`** - realizează operaţia dest/sursa cu semn cu depunere în dest.
- **`aad`** – (ASCII Adjust for Division) realizează o corecţie ASCII la împărţire a rezultatului stocat în registrul A după o operaţie de împărţire ASCII
- **`cbw`** – (Convert Byte to Word) – permite o extensie de semn a lui AL în AH
- **`cwd`** – (Convert Word to DoubleWord) – permite o extensie de semn a lui AX în DX

3. Instrucţiuni logice

- Se clasifică în:

a) instrucţiuni cu un operand (monadice)

b) instrucţiuni cu doi operanzi (diadice)

3.a. Instrucţiuni cu un operand

- `not dest` - realizează complementul faţă de 1 al operandului dest cu depunere în dest.
- deplasări:
  - `shl dest, contor` - realizează deplasarea logică la stânga a destinaţiei cu un număr de biţi specificaţi de contor
  - `shr dest, contor` - realizează deplasarea logică la dreapta a destinaţiei cu un număr de biţi specificaţi de contor
  - `sal dest, contor` – realizează deplasarea aritmetică la stânga a destinaţiei cu un număr de biţi specificaţi de contor
  - `sar dest, contor` - realizează deplasarea aritmetică la dreapta a destinaţiei cu un număr de biţi specificaţi de contor

3.a. Instrucţiuni cu un operand

- rotiri:
  - `rol dest, contor` - realizează rotirea logică la stânga a destinaţiei cu un număr de biţi specificaţi de contor
  - `ror dest, contor` - realizează rotirea logică la dreapta a destinaţiei cu un număr de biţi specificaţi de contor
  - `rcl dest, contor` – realizează rotirea cu carry la stânga a destinaţiei cu un număr de biţi specificaţi de contor
  - `rcr dest, contor` - realizează rotirea cu carry la dreapta a destinaţiei cu un număr de biţi specificaţi de contor

3.a. Instrucţiuni cu doi operanzi

- `and dest, sursa` - realizează operaţia *şi logic* între destinaţie şi sursa cu depunere în dest
- `or dest, sursa` - realizează operaţia *sau logic* între destinaţie şi sursa cu depunere în dest
- `test dest, sursa` – realizează operaţia *şi logic* între destinaţie şi sursa fără depunere în dest
- `xor dest, sursa` - realizează operaţia *sau exclusiv* între destinaţie şi sursa cu depunere în dest

4. Instrucţiuni pentru manipularea şirurilor de caractere

- Pentru toate instrucţiunile cu şiruri de caractere se consideră că şirul sursă este conţinut în segmentul curent de date, pointat de regiştri DS:ESI, iar şirul destinaţie în extrasegment, pontat de ES:EDI.

- Pentru sursă se poate considera şi un alt registru segment dacă se foloseşte un prefix de registru adecvat.

4. Instrucţiuni pentru manipularea şirurilor de caractere

- Sensul de parcurgere în memorie al şirurilor este indicat de indicatorul de condiţie DF (direction flag) din registrul de stare
  - 0 = sens crescător, de la adrese mici spre adrese mari
  - 1 = sens descrescător, de la adrese mari spre adrese mici.

4. Instrucţiuni pentru manipularea şirurilor de caractere

- Un prefix de un octet poate precede instrucţiunile cu şiruri pentru a indica că operaţia trebuie repetată pânã sunt îndeplinite condiţiile indicate. Exemple:
  - REP – repetare pânã CX=0
  - REPZ – repetare cât ZF=1, pânã ZF=0
  - REPNC – repetare cât CF=0, pânã CF=1

4. Instrucţiuni pentru manipularea şirurilor de caractere

- Instrucţiunile primitive cu şiruri de caractere sunt:

- **movb**, **movw** - transfera un operand de un octet (cuvânt) de la şirul sursă în destinaţie
- **cmpb**, **cmpw** - compară operanzii corespunzători din şirul sursă şi destinaţie, poziţionând corespunzător indicatorii de condiţie din registrul de stare.
- **scab**, **scaw** - compară elementul curent al şirului destinaţie cu valoarea din registrul AL. Dacă se execută în mod repetat scanează şirul destinaţie în căutarea valorii din AL.
- **lodb**, **lodw** - transfera un operand de un octet (cuvânt) de la şirul sursă în registrul AL (AX). Această operaţie nu poate fi repetată.
- **stob**, **stow** - transfera un operand de un octet (cuvânt) din AL (AX) în şirul destinaţie. Repetitiv se poate folosi pentru umplerea unui buffer cu o valoare dată.
- **inb**, **inw** - citeşte data de la un port de intrare
- **outb**, **outw** - scrie data la un port de ieşire

5. Instrucţiuni pentru controlul fluxului de executie a programului

- Se pot împărți în patru clase:

  a) transferuri necondiționale

  b) transferuri condiționale

  c) controlul iterațiilor

  d) întreruperi software

5.a. Transferuri necondiționale

`CALL` `eticheta`

- Instrucțiune pentru apeluri de subprograme. Se execută subprogramul de la eticheta specificată.
- Pașii parcurși în execuția subprogramului sunt:
  - salvare în stivă a adresei de revenire în programul apelant
  - transfer control (încărcare registru IP cu adresa de început a subprogramului).

5.a. Transferuri necondiționale

- Exista doua tipuri de apel:
  - NEAR – subprogram in acelasi segment de cod cu programul apelant (se indica doar adresa offset)
  - FAR - subprogram in alt segment de cod decat programul apelant (se indica adresa segment si adresa offset)
- Alte forme pentru CALL:
  `CALL reg` -> adresa de salt se afla in registru
  `CALL [reg]` -> adresa de salt se afla la adresa indicata in registru.

5.a. Transferuri necondiționale

- **`JMP`** `eticheta` - transfer necondiționat la adresa indicată de `eticheta`.
- **`JMP`** `SHORT nr.` – forma scurtă a instrucțiunii `JMP` care realizează un transfer necondiționat la instrucțiunea aflată cu +/- nr. octeți după/înainte de instrucțiunea curentă.
- **`RET`** – instrucțiune de întoarcere din subprogram care transferă controlul programului la adresa din varful stivei. De obicei aceasta reprezintă adresa de întoarcere în programul apelant.
- **`RET`** `nr.` – instrucțiune de întoarcere cu extragerea a nr. parametri din stivă.

5.b. Transferuri condiționate

Instructiune	Conditie de salt	Interpretare
JE, JZ	ZF = 1	Zero, Equal
JL, JNGE	SF ≠ OF	Less, Not Greater or Equal
JLE, JNG	SF ≠ OF sau ZF = 1	Less or Equal, Not Greater
JB, JNAE, JC	CF = 1	Below, Not Above or Equal, Carry
JBE, JNA	CF = 1 sau ZF = 1	Below or Equal, Not Above
JP, JPE	PF = 1	Parity, Parity Even
JO	OF = 1	Overflow
JS	SF = 1	Sign
JNE, JNZ	ZF = 0	Not Zero, Not Equal
JNL, JGE	SF = OF	Not Less, Greater or Equal
JNLE, JG	SF = OF şi ZF = 0	Not Less or Equal, Greater
JNB, JAE, JNC	CF = 0	Not Below, Above or Equal, Not Carry
JNBE, JA	CF = 0 şi ZF = 0	Not Below or Equal, Above
JNP, JPO	PF = 0	Not Parity, Parity Odd
JNO	OF = 0	Not Overflow
JNS	SF = 0	Not Sign

5.b. Transferuri condiționate

- Instructiuni care implementeaza structurile alternative ale programarii structurate.
- Ramificatiile se fac in functie de starea indicatorilor de conditie, iar salturile nu pot depasi +/- 128 octeti de la adresa instructiunii de salt.

5.b. Transferuri condiționate

- **`JZ eticheta`** - inseamna Jump on zero si transfera controlul programului la eticheta specificata daca flagul de ZERO =1
- **`JNZ eticheta`** - inseamna Jump on not zero si transfera controlul programului la eticheta specificata daca flagul de ZERO =0
- **`JC eticheta`** - inseamna Jump on carry si transfera controlul programului la eticheta specificata daca flagul de CARRY =1
- **`JNC eticheta`** - inseamna Jump not carry si transfera controlul programului la eticheta specificata daca flagul de CARRY =0
- **`JP eticheta`** - inseamna Jump on parity si transfera controlul programului la eticheta specificata daca flagul de PARITY =1
- **`JP eticheta`** - inseamna Jump not parity si transfera controlul programului la eticheta specificata daca flagul de PARITY =0

5.b. Transferuri condiționate

- Alte instructiuni:
  - **`JE eticheta`** – inseamna Jump on Equal si este idem JZ.
  - **`JNE eticheta`** – inseamna Jump not Equal si este idem JNZ.
  - **`JL eticheta`** – inseamna Jump on Less si inseamna salt daca dest<sursa
  - **`JG eticheta`** – inseamna Jump on Greater si inseamna salt daca dest>sursa
  - **`JLE eticheta`** – inseamna Jump on Less or Equal si inseamna salt daca dest<=sursa
  - **`JCXZ eticheta`** – inseamna salt la eticheta daca continutul registrului CX este 0.

5.c. Controlul iterațiilor

- Controlul iteratiilor este realizat cu instructiuni de tip LOOP care implementeaza structuri repetitive cu testul la sfarsit.
- Adresa de repetitie trebuie sa se incadreze in domeniul +/- 128 octeti de la instructiunea curenta.

- **`LOOP eticheta`** – decrementeaza pe CX cu 1 si face salt la instructiunea specificata de eticheta daca CX<>0.
- **`LOOPZ eticheta`** – decrementeaza pe CX cu 1 si face salt la instructiunea specificata de eticheta daca CX<>0 si ZF=1
- **`LOOPNZ eticheta`** – decrementeaza pe CX cu 1 si face salt la instructiunea specificata de eticheta daca CX<>0 si ZF=0

6. Întreruperi software

- Instrucțiunile generează apelul rutinei (subprogramului) de întrerupere cu numărul specificat.

```
INT nr
```

- Fiecare rutina de intrerupere are
  - un numar asociat care refera nivelul intreruperii
  - o adresa care la care se afla stocata.

- La I8086 există 256 de nivele de întrerupere.

6. Întreruperi software

- Pentru executia instructiunii INT nr (sau a rutinei asociate unei intreruperi hardware):
    - se salveaza in stiva registrul de stare si adresa de intoarcere in program
    - se calculeaza adresa din tabel corespunzatoare nivelului: **nr*4+adresa de baza** in tabel* si se preda controlul rutinei aflate la adresa respectiva.
- Rutinele de intrerupere se incheie cu instructiunea IRET.
    - `IRET` – instructiune care incarca registrul de stare si registrii CS si IP cu valorile din varful stivei.
    - `INTO` – instructiune care genereaza o intrerupere de nivel 4 daca OF=1.

*In memorie exista un tabel asociat intreruperilor numit “tabel al vectorilor de intrerupere”. El contine pentru fiecare nivel de intrerupere patru octeti ce reprezinta adresa de memorie unde se afla stocata rutina respectiva de intrerupere (2 octeti pentru adresa segment si 2 octeti pentru adresa offset).