# ANALIZA CONCEPTULUI DE DEPASIRE (OVERFLOW)

**CF** (*Carry Flag*) este flagul de transport. Are valoarea 1 în cazul în care în cadrul ultimei operatii efectuate (UOE) s-a efectuat transport în afara domeniului de reprezentare a rezultatului şi valoarea 0 în caz contrar. De exemplu, pt

| | | | | |
|---|---|---|---|---|
| 1001 0011 + | 147 + | | 93h + | -109 + |
| 0111 0011 | 115 | rezulta un transport de cifra semnificativa si | 73h | 115 |
| **1** 0000 0110 | 262 | valoarea 1 este depusa automat in CF | 106h | 06 |
| | **(fără semn)** | | **(hexa)** | **(cu semn)** |
| | **CF=1** | | | **OF=0** |

**flagul CF semnalează depăşirea în cazul interpretării FĂRĂ SEMN (CF=1).**

**OF** (*Overflow Flag*) este flag pentru depăşire **CU SEMN**. Dacă rezultatul ultimei instrucţiuni în interpretarea CU SEMN a operanzilor nu a încăput în spaţiul rezervat operanzilor (intervalul de reprezentare admisibil), atunci acest flag va avea valoarea 1, altfel va avea valoarea 0. Pentru exemplul de mai sus, OF=0.

**Definiţie (general, comprimată şi incompletă).** O *depăşire* este o condiţie/situaţie matematică ce exprimă faptul că rezultatul unei operaţii nu a încăput în spaţiul rezervat acestuia. (nici -147 nu "încape" în intervalul [-128..+127] şi nici pe un byte însă e mai dificil de intuit că definiţia cuprinde şi se referă şi la acest caz..)

**Definiţie mai exactă şi completă**. La nivelul procesorului şi a limbajului de asamblare o *depăşire* este o condiţie/situaţie matematică ce exprimă faptul că rezultatul UOE nu a încăput în spaţiul rezervat acestuia SAU că acest rezultat nu aparţine intervalului de reprezentare admisibil pe acea dimensiune de reprezentare SAU că operaţia efectuată este un nonsens matematic în respectiva interpretare (cu semn sau fără semn) şi nu poate fi astfel acceptată drept o operaţie matematică corectă.

**CF vs. OF. Conceptul de depăşire.**

| 1001 0011 + | 147 + | 93h + | -109 + | **(asta ne-am aştepta să** |
|---|---|---|---|---|
| 1011 0011 | 179 | B3h | - 77 | **obţinem ca rezultat** |
| **1** 0100 0110 | **326** | 1 46h | **- 186** !!!! | **şi în program !!!)** |
| **(reprezentare) binară** | **(interpretare fără semn)** | **(reprezentare) hexa** | **(interpretare cu semn)** | **- adică 326 şi -186 !** |

- 326 şi -186 sunt rezultatele corecte în baza 10 ale celor două interpretări ale OPERANZILOR binari de mai sus

Din discuţia asupra intervalelor de reprezentare admisibile şi respectiv din analiza tipului de probleme « Care este numărul minim de biţi pe care se poate reprezenta… 326 şi apoi respectiv -186 » va rezulta că

326 € [0..511] şi -186 € [-256..+255] şi că astfel numărul MINIM de biţi pe care se pot reprezenta 326 şi respectiv -186 este 9, iar reprezentarea lui -186 este: 512-186 = 326 = 1 46h = 1 0100 0110

Ca urmare, TOATE operaţiile de mai sus se desfăşoară CORECT MATEMATIC pe 9 biţi şi operanzii şi rezultatele finale INCAP în spaţiul rezervat, DACA operatiile se desfăşoară pe 9 biţi !!

Insă din păcate, adunarea de mai sus se desfăşoară la nivel de procesor pe 8 biţi (deoarece în limbaj de asamblare avem că ADD b+b → b) şi ca urmare dpdv MATEMATIC, aceasta NU se va desfăşura corect pe 8 biţi, nici 326 şi nici -186 neîncăpând pe 1 octet !! (CARE NU se va desfăşura corect pe 8 biţi mai exact ? – base 2, base 10, base 16 ?... NICI UNA nu se va desfăşura corect pe 8 biţi !!! şi de aceea CF şi OF = 1 ambele…)

Acest lucru este semnalat SIMULTAN de către flag-urile CF (pt interpretarea fără semn) şi respectiv OF (pt interpretarea CU semn), ambele flag-uri fiind setate la valoarea 1.

Ca urmare, ceea ce vom obţine ca şi rezultate şi mai ales ca EFECTE pe 1 byte în program va fi :

| | | | | |
|---|---|---|---|---|
| 1001 0011 + | 147 + | | 93h + | -109 + |
| 1011 0011 | 179 | rezulta un transport de cifra semnificativă şi | B3h | - 77 |
| 1 0100 0110 | +70 | valoarea 1 este depusă în CF | 1 46h | + 70 !!!! |
| **(fără semn)** | | | **(hexa)** | **(cu semn)** |
| **CF= 1** | | | | **OF= 1** |

(Deci nu obţinem din păcate 326 şi -186 aşa cum ar trebui corect matematic, ci +70 în ambele interpretări, adică operaţii incorecte matematic în ambele interpretări !!!!)

Prin setarea flag-urilor CF şi OF la valoarea 1, procesorul ne « transmite mesajul » că ambele interpretări în baza 10 ale operaţiei binare de adunare de mai sus sunt operaţii matematice incorecte !

---

| | | | |
|---|---|---|---|
| 0101 0011 + | 83 + | 53h + | 83 + |
| 0111 0011 | 115 | 73h | 115 |
| 1100 0110 | 198 | C6h | 198 !!!! |
| **(interpretarea fără semn a operanzilor)** | | **(hexa)** | **(interpretarea cu semn a operanzilor)** |

- 198 este rezultatul corect în baza 10 pentru ambele interpretări ale OPERANZILOR binari din adunarea de mai sus, INSA trebuie să vedem acum dacă rezultatul încape pe 8 biţi (DA – încape, de aceea vom avea CF=0) şi respectiv dacă rezultatul operaţiei binare în interpretarea **cu semn** este unul consistent cu corectitudinea operaţiei matematice efectuate (NU este, deoarece 11000110 NU este un număr pozitiv în interpretarea CU semn !), deci ceea ce vom obţine ca şi rezultate pe 1 octet va fi :

| | | | | |
|---|---|---|---|---|
| 0101 0011 + | 83 + | | 53h + | 83 + |
| 0111 0011 | 115 | NU rezulta un transport de cifră semnificativă | 73h | 115 |
| 1100 0110 | 198 | deci CF=0 | C6h | **-58** !!!! |
| **(fără semn)** | | | **(hexa)** | **(cu semn)** |
| **CF=0** | | | | **OF= 1** |

Setarea CF=0 exprimă faptul că interpretarea fără semn în baza 10 a adunării în baza 2 de mai sus este o operaţie corectă, atât la nivelul interpretării OPERANZILOR cât şi a REZULTATULUI. OF va fi însă setat de către procesor la valoarea 1, acest lucru însemnând că interpretarea cu semn în baza 10 a adunării în baza 2 de mai sus reflectă o operaţie incorectă (mai exact REZULTATUL obţinut este unul incorect în interpretarea cu semn!)

OF va fi setat la valoarea 1 (*signed overflow*) dacă pentru operaţia de adunare ne aflăm în una din următoarele două situaţii (*regulile de depăşire la adunare pentru interpretarea cu semn*). Sunt singurele două situaţii care provoacă depăşire la adunare în interpretarea cu semn:

0……+ sau 1…..+ (Semantic, cele doua situatii exprimă imposibilitatea acceptării matematice a celor
0…… 1….. 2 operatii : nu putem aduna două numere pozitive şi să obţinem unul negativ şi nici
-------- ------- nu putem aduna două numere negative şi să obţinem unul pozitiv).
1…… 0…..

In cazul scăderii, avem de asemenea două reguli de depăşire în interpretarea cu semn, consecinţe a celor două reguli de la depăşirea în cazul adunării :

1…….- sau 0…..- (Semantic, cele doua situatii exprimă imposibilitatea acceptarii matematice a celor 2
0…… 1….. operatii: nu putem scadea un numar pozitiv dintr-un nr.negativ si sa obtinem unul pozitiv
------ ------- şi nici nu putem scadea dintr-un nr pozitiv unul negativ si sa obtinem unul negativ).
0…… 1…..

| | | | | |
|---|---|---|---|---|
| **1** 0110 0010 - | 98 - | | 62h - | 98 - |
| 1100 1000 | 200 | | C8h | -56 |
| 1001 1010 | -102 | | 9Ah | 154 |
| **(în interpretarea fără semn A OPERANZILOR)** | | **(hexa)** | **(în interpretarea cu semn A OPERANZILOR)** | |

(REZULTATELE MATEMATICE CORECTE sunt precizate mai sus- ignorând **INTERPRETAREA** în vreun fel a configuraţiei binare 1001 1010)

Insă, dacă luăm în considerare şi interpretările REZULTATULUI obţinut în program, avem din păcate :

| | | | | | |
|---|---|---|---|---|---|
| **1** 0110 0010 - | 98 - | | | 62h - | 98 - |
| 1100 1000 | 200 | Avem nevoie de împrumut de cifră semnificativă | | C8h | -56 |
| 1001 1010 | 154 | pt efectuarea scăderii şi de aceea valoarea 1 | | 9Ah | -102 !!!! |
| | **(fără semn)** | este depusa în CF | | **(hexa)** | **(cu semn)** |
| | **CF=1** | | | | **OF=1** |

Ambele interpretări ale rezultatului obţinut în baza 2 SUNT INCORECTE MATEMATIC, deci CF şi OF vor fi ambele setate la valoarea 1.

Nici una dintre cele 2 interpretari nu este consistentă în baza 10 : 98-200 (interpretarea fără semn a scăderii) ar fi trebuit să furnizeze -102 ca rezultat matematic corect (valoarea aceasta fiind disponibilă doar în interpretarea cu semn !!), valoarea 154 nefiind în mod evident un rezultat corect ! Interpretarea CU SEMN furnizeaza 98-(-56) = -102 (rezultat evident incorect !), deoarece 98+56 = 154 (acesta ar fi trebuit sa fie rezultatul corect, insa interpretarea 154 pt rezultat este valabila doar in interpretarea fara semn). Ca urmare, se constată că pentru a fi corecte matematic rezultatele finale ar fi trebuit să fie exact invers repartizate celor 2 interpretari, însă nu este aşa, cele 2 operaţii matematice de mai sus (adică cele 2 interpretări asociate scaderii din baza 2) fiind ambele incorecte dpdv matematic. Ca urmare şi ca reacţie a mP 80x86 la aceasta situaţie vom avea CF=1 şi respectiv OF=1.

Tehnic vorbind, microprocesorul seteaza OF=1 doar în una din cele 4 situaţii prezentate mai sus (2 situaţii pt adunare şi respectiv 2 situaţii pt scădere) plus încă o situaţie pt înmulţire care va fi explicată în cele ce urmează.

Operatia de înmulţire NU furnizează depăşire la nivelul arhitecturii 80x86, spaţiul rezervat pt rezultat fiind suficient pentru ambele interpretări. Totuşi, pt a nu rămâne neutilizate flag-urile CF şi OF în cazul înmulţirii s-a luat decizia ca în cazul în care în cadrul operaţiei de înmulţire dimensiunea rezultatului se întâmplă să fie identică cu cea a operanzilor (b*b = b, w*w = w sau d*d = d) flag-urile CF şi OF să fie setate ambele la valoarea 0 (« no multiplication overflow », CF = OF = 0), iar dacă avem în mod real una dintre situaţiile b*b = w, w*w = d, d*d = qword, atunci CF = OF = 1 (« multiplication overflow »).

Cel mai grav efect al unei situaţii de depăşire se manifestă în cazul împărţirii : în cazul acestei operaţii, dacă câtul obţinut nu încape în spaţiul rezervat (spaţiul rezervat de către asamblor fiind byte pentru împărţire word/byte, word pentru împărţire doubleword/word şi respectiv doubleword pentru împărţire quadword/doubleword) atunci se va semnala situaţie de « depăşire la împărţire » cu efectul 'Run-time error' şi cu emiterea din partea sistemului de operare a unuia dintre cele 3 mesaje echivalente : 'Divide overflow', 'Division by zero' sau 'Zero divide'.

In cazul unei împărţiri care se efectuează corect, adică fără a se semnala depăşire, CF şi OF sunt nedefinite. Dacă avem însă depăşire, programul « crapă », execuţia lui se încheie, deci practic nu mai are nici un sens pentru nimeni să se întrebe ce valoare au la acel moment flag-urile CF şi OF…

**w/b → b 1002/3 = 334 = situaţie de depăşire (overflow) în cazul împărţirii – fatal – Run time error (**'Divide overflow', 'Division by zero' sau 'Zero divide' – oare DE CE se emite un astfel de mesaj care sugerează împărţirea la zero cu toate că aici am împărţit la 3 ??).

Pt.că dpdv al procesorului indiferent că împărţim la ZERO sau la ceva diferit de zero dar rezultatul NU ÎNCAPE în spaţiul rezervat, situaţia în care se ajunge este ACEEAŞI : run-time error pe motiv că NU ÎNCAPE!!

**334 nu încape pe un byte, iar INFINIT NU INCAPE IN NIMIC !!!!!!!!!**

**număr/0 – Zero divide = ESTE OVERFLOW PT CA INFINIT NU INCAPE IN NIMIC !!!!!!!!!**

**De ce am nevoie SIMULTAN de CF şi OF in EFLAGS ?? Nu ajunge un singur flag pt a îmi arăta PE RAND daca am sau nu depăşire fie în interpretarea cu semn fie în cea fără semn ? NU, pt că** în momentul efectuării unei operaţii de adunare sau scădere în baza2 se efectuează de fapt SIMULTAN 2 operaţii în baza10: una în interpretarea cu semn şi cealaltă în interpretarea fără semn.

În consecinţă e nevoie **SIMULTAN** de două flaguri diferite care sa se ocupe fiecare **SEPARAT** de **<u>câte una</u>** din cele 2 interpretări posibile în baza 10:

- CF – pt interpretarea fara semn ; OF – pt interpretarea cu semn

  Aceasta se întâmplă deoarece operaţia de adunare sau scădere exprimată IN BAZA 2 se efectuează IDENTIC, la fel deci, INDIFERENT DE INTERPRETAREA cu semn sau fără semn a operanzilor şi a rezultatului !!!! Acesta este şi motivul pt care în limbaj de asamblare NU EXISTA IADD sau ISUB ! Pt ca ele şi dacă ar exista NU ar funcţiona diferit de ADD si respectiv SUB !

  **ADD = IADD, SUB = ISUB – Pt ca în baza 2 operaţia exprimată se efectuează la fel INDIFERENT DE INTERPRETARE !!!!**

- **DE CE AM NEVOIE DE IMUL si IDIV ??**

Pentru că spre deosebire de adunare şi scădere, care funcţionează la fel în baza2 , indiferent de interpretare (cu semn sau fără semn) înmulţirea şi împărţirea CU SEMN şi FARA SEMN funcţionează diferit în cazul cu semn comparativ cu cazul fără semn !!

Ca urmare la adunare şi la scădere nu există necesitatea de a preciza ANTERIOR desfăşurării lor cum dorim să fie interpretaţi operanzii şi rezultatul, deoarece cele 2 operaţii oricum funcţionează la fel BINAR indiferent de cum dorim să le interpretam. Este suficient să ne decidem ULTERIOR efectuării operaţiei cum dorim sa fie interpretaţi operanzii şi rezultatul.

In schimb înmulţirea şi împărţirea NU funcţionează BINAR la fel în cele 2 interpretări, aici existând necesitatea de a preciza ANTERIOR unei înmulţiri sau împărţiri cum dorim să fie interpretaţi operanzii, iar acest lucru se face tocmai prin precizarea MUL şi DIV (dacă dorim operanzi fără semn) sau respectiv IMUL şi IDIV (dacă dorim operanzi cu semn).