

## ANALIZA CONCEPTULUI DE DEPASIRE (OVERFLOW)

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

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

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

**OF** (*Overflow Flag*) este flag pentru depășire **CU SEMN**. Dacă rezultatul ultimei instrucțiuni în interpretarea CU SEMN a operanzilor nu a încăput în spațiul 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.

**Definiție (general, comprimată și incompletă).** O *depășire* este o condiție/situație matematică ce exprimă faptul că rezultatul unei operații nu a încăput în spațiul rezervat acestuia. (nici -147 nu “încapă” în intervalul [-128..+127] și nici pe un byte însă e mai dificil de intuit că definiția cuprinde și se referă și la acest caz..)

**Definiție mai exactă și completă.** La nivelul procesorului și a limbajului de asamblare o *depășire* este o condiție/situație matematică ce exprimă faptul că rezultatul UOE nu a încăput în spațiul rezervat acestuia SAU că acest rezultat nu aparține intervalului de reprezentare admisibil pe acea dimensiune de reprezentare SAU că operația efectuată este un nonsens matematic în respectiva interpretare (cu semn sau fără semn) și nu poate fi astfel acceptată drept o operație matematică corectă.

## CF vs. OF. Conceptul de depășire.

1001 0011 +	147 +	93h +	-109 +
<u>1011 0011</u>	<u>179</u>	<u>B3h</u>	<u>- 77</u>
<b>1</b> 0100 0110	326	1 46h	- 186 !!!!
(reprezentare)	(interpretare	(reprezentare)	(interpretare cu semn)
binară	fără semn)	hexa	

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

Din discuția asupra intervalelor de reprezentare admisibile și respectiv din analiza tipului de probleme « Care este numărul minim de biți pe care se poate reprezenta... 326 și apoi respectiv -186 » va rezulta că

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

Ca urmare, TOATE operațiile de mai sus se desfășoară CORECT MATEMATIC pe 9 biți și operanzii și rezultatele finale INCAP în spațiul rezervat, DACA operațiile se desfășoară pe 9 biți !!

Însă din păcate, adunarea de mai sus se desfășoară la nivel de procesor pe 8 biți (deoarece în limbaj de asamblare avem că  $ADD\ b+b \rightarrow b$ ) și ca urmare dpdv MATEMATIC, aceasta NU se va desfășura corect pe 8 biți, nici 326 și nici -186 neîncăpând pe 1 octet !! (CARE NU se va desfășura corect pe 8 biți mai exact ? – base 2, base 10, base 16 ?... NICI UNA nu se va desfășura corect pe 8 biți !!! și de aceea CF și OF = 1 ambele...)

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

Ca urmare, ceea ce vom obține ca și rezultate și mai ales ca EFECTE pe 1 byte va fi :

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

(adică operații incorecte matematic în ambele interpretări !!!!)

Prin setarea ambelor flag-uri CF și OF la valoarea 1, procesorul ne « transmite mesajul » că ambele interpretări în baza 10 ale operației binare de adunare de mai sus sunt operații matematice incorecte !

0101 0011 +	83 +	53h +	83 +
<u>0111 0011</u>	<u>115</u>	<u>73h</u>	<u>115</u>
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 încapă pe 8 biți (DA – încapă, de aceea vom avea CF=0) și respectiv dacă rezultatul operației binare în interpretarea cu semn este unul consistent cu corectitudinea operației matematice efectuate (NU este, deoarece 11000110 NU este un număr pozitiv în interpretarea CU semn !), deci ceea ce vom obține ca și rezultate pe 1 octet va fi :

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

Setarea CF=0 exprimă faptul că interpretarea fără semn în baza 10 a adunării în baza 2 de mai sus este o operație corectă, atât la nivelul interpretării OPERANZILOR cât și 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 operație incorectă (mai exact REZULTATUL obținut este unul incorect în interpretarea cu semn!)

OF va fi setat la valoarea 1 (*signed overflow*) dacă pentru operația de adunare ne aflăm în una din următoarele două situații (*regulile de depășire la adunare pentru interpretarea cu semn*). Sunt singurele două situații care provoacă depășire 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 și să obținem unul negativ și nici
-----		-----	nu putem aduna două numere negative și să obținem unul pozitiv).
1.....		0.....	

În cazul *scăderii*, avem de asemenea două reguli de depășire în interpretarea cu semn, consecințe a celor două reguli de la depășirea în cazul adunării :

1.....-	sau	0.....-	(Semantic, cele doua situatii exprimă imposibilitatea acceptării matematice a celor 2
0.....		1.....	operatii: nu putem scădea un număr pozitiv dintr-un nr.negativ și să obținem unul pozitiv
-----		-----	și nici nu putem scădea dintr-un nr pozitiv unul negativ și să obținem unul negativ).
0.....		1.....	

<b>1</b> 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)

Însă, dacă luăm în considerare și interpretările REZULTATULUI obținut, avem :

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

Ambele interpretări ale rezultatului obținut în baza 2 SUNT INCORECTE MATEMATIC, deci CF și 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 furnizează 98-(-56) = -102 (rezultat evident incorect !), deoarece 98+56 = 154 (acesta ar fi trebuit să fie rezultatul corect, însă interpretarea 154 pt rezultat este valabilă doar în interpretarea fără semn). Ca urmare, se constată că pentru a fi corecte matematic rezultatele finale **ar fi trebuit să fie exact invers repartizate celor 2 interpretări**, însă nu este așa, cele 2 operații matematice de mai sus (adică cele 2 interpretări asociate scăderii din baza 2) fiind ambele incorecte dpdv matematic. Ca urmare și ca reacție a mP 80x86 la această situație vom avea **CF=1** și respectiv **OF=1**.

Tehnic vorbind, microprocesorul setează OF=1 doar în una din cele 4 situații prezentate mai sus (2 situații pt adunare și respectiv 2 situații pt scădere) plus încă o situație pt înmulțire care va fi explicată în cele ce urmează.

Operația de înmulțire NU furnizează depășire la nivelul arhitecturii 80x86, spațiul rezervat pt rezultat fiind suficient pentru ambele interpretări. Totuși, pt a nu rămâne neutilizate flag-urile CF și OF în cazul înmulțirii s-a luat decizia ca în cazul în care în cadrul operației de înmulțire dimensiunea rezultatului se întâmplă să fie identică cu cea a operanzilor ( $b*b = b$ ,  $w*w = w$  sau  $d*d = d$ ) flag-urile CF și OF să fie setate ambele la valoarea 0 (« no multiplication overflow »,  $CF = OF = 0$ ), iar dacă avem în mod real una dintre situațiile  $b*b = w$ ,  $w*w = d$ ,  $d*d = qword$ , atunci  $CF = OF = 1$  (« multiplication overflow »).

Cel mai grav efect al unei situații de depășire se manifestă în cazul împărțirii : în cazul acestei operații, dacă câtul obținut nu încapă în spațiul rezervat (spațiul rezervat de către asamblor fiind byte pentru împărțire word/byte, word pentru împărțire doubleword/word și respectiv doubleword pentru împărțire quadword/doubleword) atunci se va semnala situație de « depășire la împărțire » cu efectul ‘Run-time error’ și cu emiterea din partea sistemului de operare a unuia dintre cele 3 mesaje echivalente : ‘Divide overflow’, ‘Division by zero’ sau ‘Zero divide’.

În cazul unei împărțiri care se efectuează corect, adică fără a se semnala depășire, CF și OF sunt nedefinite. Dacă avem însă depășire, programul « crapă », execuția 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 și OF...

**w/b → b      1002/3 = 334      = situație de depășire (overflow) în cazul împărțirii – fatal – Run time error**  
(‘Divide overflow’, ‘Division by zero’ sau ‘Zero divide’)

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

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

În consecință e nevoie SIMULTAN de două flaguri diferite care să se ocupe fiecare de una din cele 2 interpretări posibile în baza 10:

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

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

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

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

Pentru că spre deosebire de adunare și scădere, care funcționează la fel în baza2 , indiferent de interpretare (cu semn sau fără semn) înmulțirea și împărțirea CU SEMN și FARA SEMN funcționează diferit în cazul cu semn comparativ cu cazul fără semn !!

Ca urmare la adunare și la scădere nu există necesitatea de a preciza ANTERIOR desfășurării lor cum dorim să fie interpretați operanzii și rezultatul, deoarece cele 2 operații oricum funcționează la fel BINAR indiferent de cum dorim să le interpretăm. Este suficient să ne decidem ULTERIOR efectuării operației cum dorim sa fie interpretați operanzii și rezultatul.

In schimb înmulțirea și împărțirea NU funcționează BINAR la fel în cele 2 interpretări, aici existând necesitatea de a preciza ANTERIOR unei înmulțiri sau împărțiri cum dorim să fie interpretați operanzii, iar acest lucru se face tocmai prin precizarea MUL și DIV (dacă dorim operanzi fără semn) sau respectiv IMUL și IDIV (dacă dorim operanzi cu semn).