TEORIE FLAGURI

Un flag este un indicator reprezentat pe un bit. O configurație a registrului de flaguri indică un rezumat sintetic al execuției fiecărei instrucțiuni. Pentru x86 registrul EFLAGS are 32 biți, dintre care sunt folosiți uzual numai 9 : CF, OF, SF, ZF, DF, IF, TF, AF, PF. Flag-urile se impart în 2 categorii : cele care raportează efectul generat de ultima operație efectuată (CF, OF, PF, AF, ZF, SF) și cele cu efect ulterior setării de către programator (CF, TF, IF, DF).

Carry flag este un flag de transport, semnalează depășirea în cazul interpretării fără semn. Are valoarea 1 în cazul în care în cadrul UOE s-a produs un transport în afara domeniului de reprezentare al rezultatului și valoarea 0 în caz contrar.

exemple: 1. add
```
mov al, 100
mov ah, 200
add al, ah
```
la finalul secvenței CF=1 pt. că 100+200>255 (marginea superioară a intervalului admis pe octet)

2.
```
mov al, 100
mov ah, -1
add al, ah
```
la finalul secvenței CF=1 pt. că 100+(-1) = 100+(256-1) = 355>255

Ca și regulă generală pe octet, un nr. <0 se transformă în nr. fără semn scăzând acel nr. din cardinalul intervalului de reprezentare = 256. ([0,255] interval octet)

exemple : 1. sub
```
mov al, 100
mov ah, 101
sub al, ah
```
la finalul secvenței CF=1 pt. că 100-101 = -1 < 0

2.
```
mov al, 100
mov ah, -1
sub al, ah
```
la finalul secvenței CF=1 pt. că 100-(-1) = 100-(256-1) = = -155 < 0

Overflow flag este tot un flag de transport, dar acesta semnalează depășire în cazul interpretării cu semn. Are valoarea 1 în cazul în care în cadrul UOE s-a produs un transport în afara domeniului de reprezentare al rezultatului și valoarea 0 în caz contrar.

Ca și regulă generală pe octet, dacă $x \in (127, 255]$ îl aducem în intervalul de reprezentare scăzând 256 din el.

! Adunarea sau scăderea cu 256 nu schimbă CF și OF pe octet.

exemple: add

1. mov al, 100
   mov ah, 100
   add al, ah

   OF = 1 pt. că $100 + 100 = 200 > 127$

2. mov al, -100
   mov ah, 156
   add al, ah

   OF = 1 pt. că $-100 + 156 = -100 + (156 - 256) = -100 + (-100) = -200 < -128$

exemple: sub

1. mov al, 100
   mov ah, -100
   sub al, ah

   OF = 1 pt. că $100 - (-100) = 100 + 100 = 200 > 127$

2. mov al, 100
   mov ah, 156
   sub al, ah

   OF = 1 pt. că $100 - 156 = 100 - (156 - 256) = 100 - (-100) = 200 > 127$

Pentru operația de înmulțire CF = OF = 1 dacă rezultatul obținut nu încape în același interval cu cei 2 operanzi ($b * b = w$, $w * w = d$, $d * d = q$)

exemple: cu semn.

```
mov al, -1
mov ah, 2
imul al
```

CF = OF = 0 pt. că $-1 * 2 \in [-127, 128]$

fără semn:

```
mov al, -1
mov ah, 2
mul al
```

OF = CF = 1 pt. că $-1 * 2 = (256 - 1) * 2 = 255 * 2 > 255$

La operația de împărțire, dacă rezultatul nu încape în spațiul de reprezentare alocat se produce run-time error, programul se oprește, deci e irelevantă valoarea din flag-uri.

Zero flag indică dacă rezultatul ultimei operații este sau nu 0. Totuși, acest flag nu se setează la înmulțire și împărțire indiferent de rezultat! Astfel, pentru adunare și scădere ZF = 1 dacă rezultatul UOE este 0 și ZF = 0 altfel.

Exemple: 1. mov al, 15
mov ah, 241
add al, ah

ZF = 1 pt. că 241 + 15 = 256 = 256 - 256 = 0 (256 nu încape pe un byte, deci se salvează doar ultimii 8 biți)

2. mov al, -1
mov ah, 255
sub al, ah

ZF = 1 pt. că -1 + 255 = (256 - 1) - 255 = 255 - 255 = 0

Sign flag indică bitul de semn al rezultatului ultimei operații efectuate. Rezultatul va fi adus la intervalul de reprezentare admisibil. (ex.: pe octet la $[-128, 127]$).

exemple: 1. mov al, 10
mov ah, -1
add al, ah

SF = 0 pt. că $10 + (-1) = 9 \in [-128, 127] > 0$

2. mov al, -1
mov ah, -1
add al, ah

SF = 1 pt. că $(-1) + (-1) = -2 \in [-128, 127] < 0$

Pentru imul, SF = 1 numai dacă înmulțim nr. poz cu nr. neg., altfel SF = 0.
Pentru mul va fi val. bitului de semn din al/ax/dx.

Există mai multe instrucțiuni pentru setarea flag-urilor:

1. Carry Flag

CLC - clear carry flag ⇒ CF = 0
STC - set carry flag ⇒ CF = 1
CMC - complementează carry flag ⇒ dacă CF = 0 ⇒ CF = 1
dacă CF = 1 ⇒ CF = 0

2. CLD - clear direction flag ⇒ DF = 0
STD - set direction flag ⇒ DF = 1

3. Flag-ul de întrerupere: ⌀ putea fi setat doar sub 16 biți, sub 32 de biți programatorului luându-se această posibilitate

CLI => IF=0

STI => IF=1

Flag-ul de întrerupere: folosit pentru secțiuni critice, atunci când IF=1 se oprește rularea oricărui alt proces. Nu poate fi setat sub 32 biți.

Trap flag: dacă are valoarea 1 mașina se oprește după fiecare instrucțiune. Nu poate fi schimbat.

Auxiliary flag: reține valoarea transportului de la bitul 3 la bitul 4.

Direction flag este folosit la lucrul cu șiruri. Dacă DF=0 parcurgerea se face de la stânga la dreapta, altfel parcurgerea e de la dreapta la stânga.

Există 2 instrucțiuni speciale pentru a încărca, respectiv a scoate tot registrul de flag-uri de pe stivă: PUSHF / POPF. De asemenea, există două instrucțiuni care încarcă, respectiv scot registrul de flag-uri de pe biții 0,2,4,6,7 ai registrului AH: LAHF, respectiv SAHF.

18 feb. 2019

2. a) x dw -256, 256h

* -256 se transformă în baza 2, apoi în 16

|-256| = 256 = 2^8 = 1 0000 0000

C2(256) = 1111 1111 0000 0000 = FF 00

=> în memorie va fi 00|FF

* 256h = 0256h => în memorie 56|02

y dw 256|-256, 256h & 256

* 256|-256 le transformăm în baza 2

256 = 2^8 = 1 0000 0000 adică pe word va fi 0000 0001 0000 0000

-256 = 1111 1111 0000 0000

=> 256|-256 = 1111 1111 0000 0000 = FF 00 => în memorie va fi 00|FF

* 256h & 256 le transformăm în baza 2

256h = 0000 0010 0101 0110

256 = 0000 0001 0000 0000

=> 256h & 256 = 0000 0000 0000 0000 => în memorie va fi 00|00

z db $-z, y-x
db 'y'-'x', 'y-x'

* $-z = 0 => în memorie va fi 00
* y-x = 4 => în memorie va fi 04
} => 00|04

* 'y'-'x' = le scade codurile ASCII => 'y'-'x' = 1 -> în mem. va fi 01
* 'y-x' = va pune separat codurile fiecăruia -> în mem. va fi 'y'|'-'|'x'

a db 512>>2, -512<<2

* 512>>2 shiftarea la dreapta e de fapt o împărțire cu 2^ cu cât se shiftează
  -> 512 : 2^2 = 128 = 2^7 = 1000 0000 => în memorie va fi 80

* -512<<2

  |-512| = 512 = 2^9 = 1 0000 0000

  C_2(512) = 1111 1111 0000 0000 (pe word)

  => -512<<2 = 1111 1100 0000 0000 = FC 00

  luăm doar un byte => în memorie va fi 00

b dw z-a, !(z-a)

* z-a = -6

  |-6| = 6 = 0110

  C2(6) = 1111 1111 1111 1010 = FFFA => în mem. va fi FA FF

* !(z-a) = !(-6) = 0 => în mem. va fi 00 00

c dd ($-b)+(d-$), $-2*g+3

* $-b = 4

  d-$ = 4 (doar un el. de acolo e valid)

  => 4+4=8 => în mem. va fi 08 00 00 00

* înmulțirea de pointeri nu e validă! syntax error

d db -128, 128^(~128)

* -128 = -128+256 = 128 = 2^7 = 1000 0000 = 80h => în memorie va fi 80

  ↓

  adunarea sau scăderea cu 256 nu schimbă numărul pe octet

* 128^(~128) = 1111 1111 = FF => în memorie va fi FF

  ↓

  nr ⊕ complement nr

c) times 2 resw 6

times 2 dd 1234h, 5678h

* times 2 resw 6

00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00|00

1 word

6 words

* times 2 dd 1234h, 5678h

34|12|00|00|78|56|00|00|34|12|00|00|78|56|00|00

b) mov bh, 7fh

7f = 0111 1111

cmp bh, al ⇒ scădere fictivă bh - al ⇒ dacă bit semn al = 1 ⇒ CF = 1

altfel ⇒ CF = 0

! shiftările nu se transmit de la ah la al

rcr ah, 1 ⇒ pune pe prima poz din ah CF (rotate carry right ⇒ completează cu CF)

sar ah, 7 ⇒ shift aritmetic right (completează cu bitul de semn)

⇒ în ah va fi bitul de semn al lui al

⇒ echivalent cu movsx ax, al sau cbw

3. a) mov eax, 200

200 | 2
0 | 100 | 2
0 | 50 | 2
0 | 25 | 2
1 | 12 | 2
0 | 6 | 2
0 | 3 | 2
1 | 1 | 2
1 | 0

⇒ 200 = 1100 1000 = C8h

signed și unsigned e la fel în eax (200 începe pe dword)

mov ebx, 254h

254h = $16^2 \cdot 2 + 16^1 \cdot 5 + 16^0 \cdot 4 = 256 \cdot 2 + 80 + 4 = 596$ (signed și unsigned)

bl

idiv bl ⇒ 200 : 84 ⇒ al = 2 și ah = 32

⇒ ah = 32d = 20h al = 2d = 2h

la împărțire nu se modifică CF și OF

b) mov ax, 256h

$ax = 256_h = 16^2 \cdot 2 + 16^1 \cdot 5 + 16^0 \cdot 6 = 512 + 80 + 6 = 598$ signed și unsigned pe word

mov dx, -1 => dx = FF FF

$|-1| = 1 = 0000\ 0000\ 0000\ 0001$

$C_2(1) = 1111\ 1111\ 1111\ 1111 = FFFF_h = 65\,535$ unsigned

-1 signed

add ah, dh

ah = 02h

dh = FFh => 0000 0010 +
1111 1111
$\overline{1\ 0000\ 0001}$ => CF = 1 (avem transport fără semn)

OF = 0 ($-1 + 2 = 1 \in [-128, 127]$)

=> în ah va fi 01h = 1 decimal signed și unsigned

c) mov ax, ~(16h | 32)

16h = 0000 0000 0001 0110

32 = 0000 0000 0010 0000

=> 16h | 32 = 0000 0000 0011 0110 => ~(16h | 32) = 1111 1111 1100 1001 = FF C9 h (ah, al)

$= 16^3 \cdot 15 + 16^2 \cdot 15 + 16^1 \cdot 12 + 16^0 \cdot 9 = 65\,481$ unsigned

$65\,535 - 65\,481 = -54$ signed

$C9 = 16 \times 12 + 9 = 201$ unsigned = $(201 - 256) = -55$ signed

mov bx, 2000h >> 4

$2000_h = 2 \cdot 16^3 = 8192 = 0010\ 0000\ 0000\ 0000$

↳ signed și unsigned

=> 2000h >> 4 = 0000 0010 0000 0000 = 0200h => bh = 02h

înmul bh => al * bh cu semn = $-55 * 2 = -110$ signed = $65\,535 - 110 = 65426$ unsigned

C9h = 201 fără semn = $(201 - 256) = -55$ cu semn

02h = 2 cu și fără semn

CF = OF = 0 pt că b*b a încăput pe byte ($-110 \in [-128, 127]$)

mov ax, 21 << 7

21 | 2
1 | 10 | 2
0 | 5 | 2
1 | 2 | 2
0 | 1 | 2
1 | 0

-> 21 = 10101 , iar pe word = 0000 0000 0001 0101

-> 21 << 7 = 0000 1010 (ah) 1000 0000 (al) = 0A 80h

$= 16^2 \cdot 10 + 16^1 \cdot 8 = 2560 + 128 =$

$= 2688$ signed și unsigned

mov bh, 10h ^ 3

10h = 0001 0000
3 = 0000 0011
-> 10h ^ 3 = 0001 0011 = 13h $= 16 \cdot 1 + 3 = 19$ signed și unsigned

sub bh, ah

al = 1000 0000
bh = 0001 0011
-> bh - al = 0001 0011 - 1000 0000 = 1001 0011 = 93h $= 16 \cdot 9 + 3 = 147$ unsigned =
nu-am împrumutat => CF = 1 $= 147 - 256 = -109$ signed
+ - (-) = - => OF = 1

e) mov e x, ebx <=> lea eax, [ebx]