MULTIMODUL

1. Codul de apel este codul scris înainte de apelarea unei funcții.
2. Codul de intrare este codul scris la începutul unei funcții apelate.
3. Codul de ieșire este codul scris la finalul unei funcții apelate.

În momentul apelării unei funcții sunt necesari anumiți pași pt. ca programul să funcționeze optim.

1. <u>cod apel</u>
   - salvare resurse volatile (pp. că toți regiștrii sunt resurse volatile → ne folosesc în funcție) → PUSHAD (îi salvăm pe stivă)
   - transmitem parametrii: regiștrii se transmit automat, pp. că mai dăm un parametru (o valoare din memorie, a) → PUSH dword [a]
   - efectuare apel cu salvare adresă de revenire → CALL funcție
2. <u>cod intrare</u>
   - se crează un stackframe nou → PUSH EBP
     MOV EBP, ESP
   - alocăm spațiu pt. variabilele locale (variabile pt. modulul separat)
     → SUB ESP, nr_octeți
     ↳ ex. pt. EAX vor fi 4 octeți
   - salvare resurse nevolatile (salvăm regiștrii care nu țin de apel)
     ex.: eax nu ține de apel, dar vrem să-l folosim → MOV [EBP-4], EAX
3. <u>cod ieșire</u>
   - restaurare resurse nevolatile MOV EAX, [EBP-4]
   - eliberare spațiu de variabile locale ADD ESP, nr_octeți
   - eliberare cadru stivă MOV ESP, EBP
     POP EBP
   - revenire din funcție ~~în sub~~ (RET) și scoaterea de pe stivă a parametrilor
     ADD ESP, 4*1 (pt. cazul prezentat)

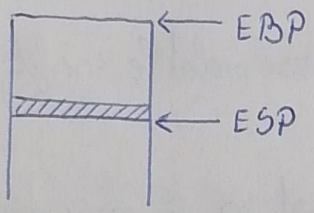
în programul principal: restaurăm resursele volatile POPAD

STIVA

Stiva este compusă din două părți: bază (EBP) și vârf (ESP)

Când scoatem un elem. de pe stivă, ESP crește cu 4 octeți (pop), se salvează vârful stivei în variabila dată ca parametru (pop parametru).

Când adăugăm un elem. pe stivă, ESP scade cu 4 octeți (push), iar la [ESP+0] va fi valoarea dată ca parametru (push parametru)



Rolul stivei este de a crea ~~loc~~ spațiu de variabile locale și de a transmite parametrii care nu sunt regiștrii

RESPONSABILITĂȚI

Codul de apel este responsabilitatea celui care apelează.

Codul de intrare/ieșire este resp. celui care este apelat.

CONVENȚII

Convenția CDECL în ASM este să transmită parametrii pe stivă și să salveze rezultatul în EAX.

Convenția STDCALL este să transmitem param. pe stivă în ordine inversă și apelantul eliberează stiva. Se declară cu _ înainte toate funcțiile care au alte convenții de apel.

8 feb. 2018

2. a) i) 61 | 2
1 | 30 | 2
0 | 15 | 2
1 | 7 | 2
1 | 3 | 2
1 | 1 | 2
1 | 0

$\Rightarrow 61 = 111101_b \Rightarrow$ nr. minim de biți pt. reprezentare e 6

$= 0_h$

ii) -62

$|-62| = 62 = 0111101_b$

↳ bit semn

$\Rightarrow$ nr. minim de biți pt. reprezentare e 7

$\Rightarrow C_2(62) = \underline{1}\,000010\,b$

↳ bit semn

iii) 130 | 2
0 | 65 | 2
1 | 32 | 2
0 | 16 | 2
0 | 8 | 2
0 | 4 | 2
0 | 2 | 2
0 | 1 | 2
1 | 0

$\Rightarrow 130 = 1000\ 0010_b = 82_h$

$\Rightarrow$ nr. minim de biți pt. reprezentare e 8

iv) -129

$|-129| = 129 = 01000\,0001$

↳ bit semn

$C_2(129) = \underline{1}\,0111\ 1111 = 17F_h$

↳ bit semn

$\Rightarrow$ nr. minim biți pt reprezentare e 9

129 | 2
1 | 64 | 2
0 | 32 | 2
0 | 16 | 2
0 | 8 | 2
0 | 4 | 2
0 | 2 | 2
0 | 1 | 2
1 | 0

b) xor ah, ah → pune 0 în ah

cwde → pune în partea high din eax 0 → eax va fi | 0 0 | 0 al |

add ebx, eax → ebx va fi ebx + al

mov al, [ebx] → al va fi [ebx + al] → echivalent cu xlat

3. a) * mov ax, 1000h

mută în ax pe $1000h = 16^3 \cdot 1 = 4096$ signed și unsigned

* mov bl, 1000b + 10b

mută în bl pe $1010b = 0Ah = 10$ signed și unsigned

* div bl → face ax : bl - al rest ah

=> 4096 : 10 = 409 rest 6

↓

nu încape în al => division overflow => runtime error (programul se oprește)

b) * mov ah, 0bch

mută în ah pe $0bch = 16^1 \cdot 11 + 16^0 \cdot 12 = 188$ unsigned = (188 - 256) = -68 signed

* mov al, 0deh

mută în al pe $0deh = 16^1 \cdot 13 + 14 = 222$ unsigned = (222 - 256) = -34 signed

* add ah, al

adună ah + al = 188 + 222 = 410 > 255 => CF = 1 (nu încape în ah)

=> în ah va fi 410 - 256 = 154 unsigned = (154 - 256) = -102 signed

∩

[-128, 127] => OF = 0

) * mov ax, 1001h

mută în ax pe 1001h = $16^3 \cdot 1 + 16^0 \cdot 1 = 4096 + 1 = 4097$ signed și unsigned

* mov bx, 1111b

mută în bx pe 1111b = 000Fh = 15 signed și unsigned

* imul bl

=> face al * bl cu semn și salvează în ax

=> în ax este 1001h => în al este 01h = 1 signed și unsigned
în bx este 000Fh => în bl este 0Fh = 15 signed și unsigned
-> în ax va fi 1 * 15 = 15 = 000Fh

15 începe pe byte => byte * byte = byte => CF = OF = 0

d) * mov dh, 62h

mută în dh pe 62h = $16^1 \cdot 6 + 16^0 \cdot 2 = 98$ signed și unsigned

* mov ch, 200

mută în ch pe 200 unsigned = (200 - 256) = -56 signed = C8h

200 | 16
192 | 12 | 16 => 200 = C8h
8 | 0 | 0
12

* sub dh, ch

scade dh - ch = 98 - 200 = -102 $\notin [0, 255]$ => CF = 1

98 - (-56) = 154 $\notin [-128, 127]$ => OF = 1

apoi explicație concept depășire