28.01.2022

Subject I

**Define and explain the concepts of call code, entry code and exit code in the general context of programming languages, after which analyse how are they reflected at the level of linking a C module with a x86 assembly language module.**

**Explain the involved tasks of each step, giving source code adequate schematic examples to clarify the involved responsibilities and needs.**

**Who is responsible for generating these codes and when exactly?**

**Why is needed the assembly language involvement in working with these concepts?**

**Describe the CDECL and STDCALL call conventions, explain where are they used.**

**Present and explain the communication mechanism between modules involved in a multimodule ASM-ASM programming and ASM-C respectively.**

Subject II

a)

a1 dd ‘0abcdefh’, 0abcdefh

**a1: ‘0’ ‘a’ ‘b’ ‘c’ ‘d’ ‘e’ ‘f’ ‘h’ | EF CD AB 00**

a2 dw ‘0abcdefh’, 3|6

**a2: ‘0’ ‘a’ ‘b’ ‘c’ ‘d’ ‘e’ ‘f’ ‘h’ | 07 00**

3 = 0011b

6 = 0110b

3|6 = 0111b = 7h

a3 dw $-a2, a2-a1

**a3: 0A 00 | 0C 00**

$-a2 = 10 = Ah = cati db ocupa a2

a2-a1 = 12 = Ch = cati db ocupa a1

a4 db 129>>1, -129<<1

**a4: 40 | FE**

129 = 1000 0001b

129>>1 = 0100 0000b = 40h

-129 = 0111 1111b

-129<<1 = 1111 1110b = Feh

a5 dw a2-a4, ~(a2-a4)

**a5: F2 FF | 0D 00**

a2-a4 = -14? = 2h

14 = 0000 1110b => -14 = 1111 0010b = F2h

~(a2-a4) = 0000 1101b = 0Dh

a6 dd $+a2-1, !a2

**syntax error (adunare de vectori + negare de adresa, nu e scalara)**

a7 dd 256h^256, 256256h

**a7: 56 03 00 00 | 56 62 25 00**

256 = 0001 0000 0000b

256h = 0010 0101 0110b

256h^256 = 0011 0101 0110b = 356h

a8 dd ($-a7) + (a9-$), -256

**syntax error (a9 not declared yet)**

a9 dw -255, -128

**a9: 01 FF | 80 FF**

255 = 1111 1111b

-255 = 0000 0001b = 1h

128 = 1000 0000b

-128 = 1000 0000b = 80h

a10 times 4 dw 128h, -128

**a10: 28 01 | 80FF | 28 01 | 80 FF | 28 01 | 80 FF | 28 01 | 80 FF**

a11 db a3

**error db not accepted (not on 16 or 32)**

a12 dw a3 = 16 10? = # of elements of a1 and a2? = 22 = 16h

**a12: 16 00**

for a3 -> 1016h = 4118

for a2 -> 100Ch = 4108 (a3 – 10)(# of elements of a2)

for a1 -> 1000h = 4096 (a3 – 22)(# of elements of a2 and a1)

b)

push ebp ; restores the base of the current stackframe

mov ebp, esp ; new stackframe

pop eax

shl eax, 1

sub eax, 5

mov ebx, eax

add ebx, eax

mov esp, ebp

I literally have no idea

Subject III

An overflow is a mathematical situation/condition which expresses the fact that the result of an operation didn’t fit the reserved space for it. At the level of the assembly language an overflow is situation/condition which expresses the fact that the result of the LPO didn’t fit the reserved space for it OR does not belong to the admissible representation interval OR that operation is a mathematical nonsense in that particular interpretation (signed or unsigned).

a)

mov ax, 0100h AX = 0100h = 256

mov bx, 1000+10b BX = 03EAh = 1002

idiv bl AX = 0EF5 ??? I don’t get why

**no overflow**

b)

mov ah, 0cdh AH = CDh

mov al, 0ebh AL = EBh AX = EDEBh

add ah, al AH = AH + AL = 12\*16 + 13 + 14\*16 + 11 = 440 = 1B8h

but 1B8h doesn’t fit in AH so:

**AH = B8h**

**overflow**

c)

mov ax, 1010h AX = 1010h

mov bx, 1111b BX = 000Fh

mul bl AX = AL\*BL = 10h\*0Fh = 16\*15 = 240 = F0h

**AX = 00F0h**

**no overflow**

d)

mov dh, 200 DH = 200 = C8h

mov ch, 62h CH = 98 = 62h

sub dh, ch DH = DH – CH = 200- 98 = 102 = 66h

**DH = 66h**

**OF = 1 so overflow but idk why**