ASC - Curs 4 | 27.10.2022

$S \leq_k V \iff \begin{cases} S \neq \emptyset \\ (\forall) v_1, v_2 \in S,\ v_1 + v_2 \in S \\ (\forall) k \in K, (\forall) v \in S,\ k \cdot v \in S \end{cases} \iff \begin{cases} S \neq \emptyset \\ (\forall) k_1, k_2 \in K, (\forall) v_1, v_2 \in S \\ k_1 v_1 + k_2 v_2 \in S \end{cases}$

Asc → Code Table

B8 → MOV eAX iv → Immediate value ← [code]

BA → MOV eDX iv

50 → PUSH eAX

52

[Mode R/M] → register/memory

— it has to decide if it is a R/M

Mem

SIB → 2 am formula

How to express an offset?

index

a(7)

base

scale: byte → 1, word → 2, dword → 4, 2word → 8

[] → optionalitatea (dif from assembly)

formula for computing addresses

offset_address = [base] + [index × scale] + [constant]

[] → dereferencing operator (the value from the address) (indications for the CPU)

[base] → any general register (which ∈ execution unit ex: EAX, EBX, ...)

[index × scale] —— n —— no ESP → it can't be an index register (restrictions)

↓

a constant: 1/2/4/8

[constant] → any constant value able to be determined (identified at assembly time)

offset - sp - four elements

! EXAMEN

nu se încadrează offset-formula → both of the operands are expressed as register operands so the formula can't be applied

mov eax, ebx

mov eax, [ebx] (offset-formula)

pointer variable expressed by register

EAX

copied

00F748B9

EBX

→ the content of EBX

[V] [ebx]

memory addressing operands

in the memory

17 ← the value

00F748B9

mov eax, [ecx + 2*EDX + 7]

base index scale constant

mov eax, 23

mov eax, [23] } → bug (olly debbuger)
→ (correct!) but it's a bug in the debugger

mov eax, [ecx + 2 * esp + 7] → WRONG = SYNTHAX ERROR

index reg. index order can be switched (ESP ≠ index register)

! EXAMEN 10 DEC: mov eax, [ebx * 2] (no base; no constant)

mov eax, [ebx * 3] ⇔ mov eax [ebx + ebx * 2] so it's valid

↓ the scale can't be 3

or 9, 1248, ...

7 doesn't work

! mov eax, [ebx + ebx * 2 + ebx * 2]

→ Hamyone can be present in one exemplar

SIB } 4 byts
Displacement

Scale Index Base

B i S
[base] + [index * scale]

!!!

CS EIP
can be modified by the programmer with jumps

mov eax, [ebx + 4 * ebx + 7]

mov eax, [V-7]

mov eax, [V] → address

mov eax, [7]

mov eax, ebx+23 → FORBIDDEN
↓
reg

Mod R/M
R. opcode

EIP → instruction pointer (column 1)
(offset of the executing program)

mov [EAX], EBX → CORRECT

mov [EAX], [EBX] → SYNTHAX ERROR
(only 1 may be stored in RAM memory)

mov (eax), 23 → CORECT
↓
byte / word / --

↑ ⟺ mov eax, [ebx*8 - ebx] WRONG

(you can't have a "-"
you can only add)

mov eax, [v]

↳ we may address prepared (that's why the formula can work)

! if there isn't a syntax error ⇒ it obeys the offset_formula

offset = how far of the begs

the content of a variable = variable

the offset of a variable = constant det in ass. time

mov eax [V]
jump mai_nicole → mov eax, [0042B7A9]
jump [0084] ↓ (go 84 bytes below)

The segment part

SEG offset

INDIRECT ADDRESSING [base] + [index * Scale]

this formula is applied for computing offsets

DIRECT ADRESSING → with the name of the variable

- compliment fata-dez
- Repr
-
-

ASC - curs 5

03.11.2022

2's complement. Discussions and examples

representation — interpretation

1 0 0 1 0 0 1 1

9 3 h

→ unsigned = 147

→ signed = − 2's complement of 10010011 = −109

base 16 = zipped ~~representation~~ notation

↳ also a representation (another version of base 2)

a) 1 0000 0000

Admissible representation intervals

N positions (bits) → 2^N values

Theory / Exam:

$[0 \dots 2^N - 1]$ - unsigned interpretation

$[-2^{N-1}, 2^{N-1} - 1]$ - signed interpretation

- on 1 byte

$[0, 255]$ - unsigned interpretation

$[-128, 127]$ - signed interpretation

- on 1 word

$[0, 65535]$ - u.i.

$[-32768, 32767]$ - s.i.

a) 1 0000 0000 −
1001 0011
0110 1101 } 109
6 D h

b) 0110 1100 +
1
0110 1101

c) ~~binary repr. of a neg. number~~ → the best binary configuration of the 2's complement of an initial value

0110 1101

d) fastest one

!!! The sum of the absolute values of the 2 complementary binary configurations / representations is the cardinal of the set of values representable on that size.

256 − 147 = 109

! identify the key words in every question / definition (dacă nu, restanță)

EXAM:

Which is the signed interpretation of

a) 1001 0011 (c) → a) 0110 1101
b) 93 h → b) 109
c) 147 (10) (c) → c) 6D h
d) +147

c) none of the above

0110 1101

iii) → none of the above !

because in the question it says „interpretation", and 147 is already an interpretation

(it should've been a representation*)

-109 is not the 2's complement of 147

in practice we use that

-109 and 147 2's comp values

is [not] reciprocal

? $\begin{pmatrix} 109 \\ -147 \end{pmatrix}$

a) $\overline{0xxx\ ---}$ → m. sig.

↘ s. sig. the same $+\overline{abc}$

$[0,255] \cap [-128, 127] = [0, 127]$

b) 1— Which is the representation for $-\overline{abc}$

c) $\overline{1xxx\ ---}$ → m. i. $+\overline{abc}$

↘ s. i. (-) 2's complement on the initial configuration

The values in signed ?

$-$ (2's compl of $\overline{1xxx\_\_\_}$)

d) Which is the representation for $\ominus$DEF ?

- the 2's complement of $\overline{1xxx\_\_\_}$

(the answer is wrong)

we don't have a 147 representation !

[-128, 127]

→ it has to be something > byte.

CORRECT ANSWER the 2's complement of the unsigned extension on an immediate superior size of the initial $\overline{00\ldots\ 1xxx\_\_\_}$ configuration

CORECT ANSWER

0000 0000 1001 0011

↓

1111 1111 0110 1101 = FF6D$_h$

2's complement -147

e) Which is the binary representation that makes complementary 109 and -147 (for one)

-109 the co

! Which is the minimum number of bits on which we can represent a number [3]

→ intervals admissible ___.

→ det N ! (from the [0, $2^N - 1$]

1 bit → $2^1 = 2$ [0,1]

2 bits → $2^2 = 4$ [0,3] u.i

[-2, +1] s.i

ANS: 2 BITS (Value 11)

! ~~W~~ ——— " ——— [-3]

3 bits → 2^3 [0,7]

[-4, 3]

ANS: 3 BITS (Value ~~11~~

011

[101] -3 (in the unsigned is 5] $101_{(2)}$

ASC - Curs 6

10.11.2022

ASSEMBLY LANGUAGE BASICS

- main task assembly = generating bytes

Pointer arithmetic

1. The only operation that can exists between 2 pointers is SUBTRACTION (because it makes sense)

$ -

↓

the point where I am at the current moment

! Any segment has its own location counter

(2-p) → the number of bytes between 2 pointers

2. Adding a constant to a pointer

a[7] = *(a+7) (the content of (a+7))

p+7 (pass an array constants as an index)

3) Substracting a constant

p - 7

→ ~~What if 5 go out of it?~~

~~4)~~

in masm

q + p → doesn't report a syntax error

↳ it doesn't mean addition of 2 pointers

a[7] = *(a+7) = *(7+a) = 7[a]

p+7 → POINTER DATA TIME

q-p → SCALAR VALUE

[base] + [index * scale] ⊕ [const]

v db 17

a dw 232, -17, 'xy'

b dd 12345678h

add EBX, [EBX+ECX*2+v-17] ✓

mov EBX, [EBX, ECX*2+a+b-7] ~~syntax error~~

sub [EBX+ECX*2 + a+b- 7]

mov EBX, [EBX, ECX*2- Y -7] - syntax error

ASC - curs 4

17.11.2022

Location counter ~~has the type pointer~~ is an integer number of type pointer

masks

4 AND
- X AND 0 = 0
- X AND 1 = X
- X AND X = X
- X AND ~X = 0

1 OR
- X OR 0 = X
- X OR 1 = 1
- X OR X = X
- X OR ~X = 1

^ XOR
- X xor 0 = X
- X xor 1 = ~X for complementing
- X xor X = 0
- X xor ~X = 1

10 dec

12 ex de initializare a unui registru cu 0

```
AND EAX,0
SUB EAX,EAX
XOR EAX,EAX
mov EAX,0
shl EAX,...
```

```
a db 17,-1,'xyz'
b dw 40001,-128,'13'
mov eax,![a]
mov eax,a
mov eax,![a+17]
mov eax,![b-a]
mov eax,[a+17]
mov eax,!7
```

mov eax, -7

mov eax, 0

mov eax, !ebx - syntax error

aa equ 2

mov ah, !aa

mov ah, ~(v17)
ffffff~h

mov axh, v7

value ~(v value) fff....f

v db

a dw

b dd

---

push v stack ← offset (v)

push [v] ✗ (it hasn't dimensions, which has to be 16/32 bits)

mov eax, [v] ✓

push [eax] ✗

push 15 ✓ - push dword 15 (exception in masm)

pop [v] ✗ (you don't specify the size of the tagged n)
pop v ; 2=3 (stupidity) nu e atribuibil
pop ✓ eax

pop (eax) ✗

pop 15 ✗

mov [v], 0 ✗ (doesn't know if it is a byte--)

push [15] - should be ok, but

push [v] ↕ mov [SS:ESP], [v]

n

dword

ASC - curs 8

23.11.2022

~~Variable = (address, content)~~ - NU

The concept of a variable in assembly language

Variable : (name, set of alias, reference, value) - cuadruplu

(optional)

(address)

if you have a ref ⇒ you have a value

type, (most important)

visibility (domain (scope)) → local / global variables

lifetime, memory class (extent) → temporal var.

(auto, register, static, extern)

space variable (text)

interval de prog unde în care acea variabilă este accesibilă

↳ DOV (Domeniu de vizibilitate)

→ the space alocated for a variable is cleared ⟹ variable = dead

→ variable = register (for efficiency)

→ auto = mem class will adapt

- local → alocated in stack
- global → — " — global data segment

→ heap → dynamic variables

→ variabila pointer ≠ variabila dinamică

→ p = new (----) (size of)

p - pointer to help managing the dynamic variable

→ dynamic variable → accesed by a pointer

→ free(p)

→ Dynamic variables don't have names.

static, extern → very important

STATIC

import - export

(static) int f ⇒ funcția devine locală

↳ face locală acelui modul funcția respectivă

- var trăiește atâta timp cât DoV = activ

  lifetime (linked) DoV

GDS → global variables + static variables

global data segment

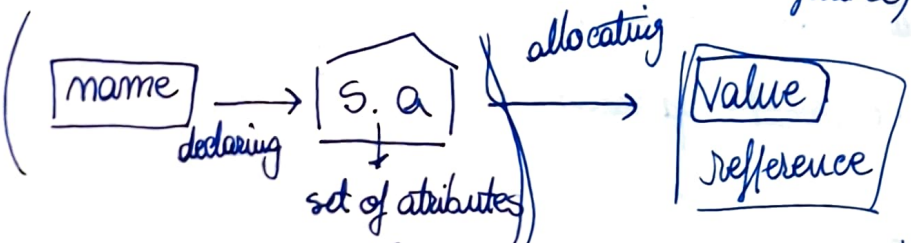
EXTERN

linkeditor - checks if ~~the~~ you have the same name for 2 variables (the same variable for 2 modules)

definition = declaring its atributes + allocation

unique — not unique — unique

(also a duplicate if you declare int a, float a)



- formal parameter (int a, int b, int c) → variables that have a set of atr., memory but don't have a value

TIMES = directiv that can also be applied for instructions

EQU = constant def.

Segment data

i1 db 0,1,2,'xyz' ✓ (the assembler generates bytes '.' : 00 01 02

db 300, "F" + 3 ✓ (variable) ≠ (46) + 3 = 49 (300 = 12C)

nu încape într-un octet

va lua ult. 787776 octeți din reprezentare unității de size

a2 TIMES 3 db 44h ✓ 44 44 44

a3 TIMES 11 db 5,1,3 ✓ 33 bytes (5 1 3 5 1 3 ...)

offset[a2] = 8 (from the start (0 1 2 ...))

a4 db a2+1 (it doesn't fit a byte ⇒ it will be a pointer)

a41 dw a2+1, 'bc'

a42 dw 1009h

a5 dd a2+1, 'bcd'

a6 TIMES 4 db '13'

a61 TIMES 4 dw '13'

'1' | '3' → concatenare de 2 octeți

'ab' | 'a' | 'b'

→ generate the same ∇∇∇ (10 elec)

'1' | '3' → deja un word.

a7 db a2

a8 dw a2

a9 dd a2

a10 dq a2 - 0000... .0

a11 db [a2]

a12 dw [a2]

a13 dd [a2]

a14 dq [a2]

a15 dd eax

a16 dd [eax]

syntax error!
(cannot appear as ~~not~~ evaluable expressions at assembly time)

!!! offsets on 16 bits or 32 bits but NEVER on bytes ⇒ syntax error

ASC - Curs 9

24.11.2022

→ exe. file → NOT the assembler
↳ generated by OBJ file
→ generated by the assembler

→ The linker checks if a variable is defined twice

segmentarea = programatorii, pe calculator nu-l intalnesc segmentarea

mov EDX, [CS:C] → no syntax error

∀ all 4 (CS, DS, ES, SS) start at the same address
FS, GS are not part of this ↑ (introduced on 32 bits)

(3 rules CS, SS, DS)

mov eax, [ebx + esp] - SS ✓ (index, base)
mov eax, [esp + ebx] - SS ✓
mov eax, [ebx + esp*2] - syntax error ✗ (esp can't be an index, can be only base register)
mov eax, [ebx + ebp*2] - DS ✓
mov eax, [ebx + ebp] - DS } the order
mov eax, [ebp + ebx] - SS
mov eax, [ebx*2 + ebp] - SS
mov eax, [ebx*1 + ebp] - SS (because EBP = base)
mov eax, [ebp*1 + ebx] - DS (EBP - index, EBX - base)
mov eax, [ebx*1 + ebp*1] - SS (EBX = index, EBP = base) } because we have 1
mov eax, [ebp*1 + ebx*1] - DS (EBP = index, EBX = base)
mov eax, [ebp*1 + ebx*1]
mov eax, [ebp*1 + ebx*2] - SS (EBP = base (sets first the base, but when it goes further and sees *2 → sets EBX = base)
mov eax, [ebp] - SS (EBP = base)
mov eax, [4*ebp] - DS (no base → 3 rule)

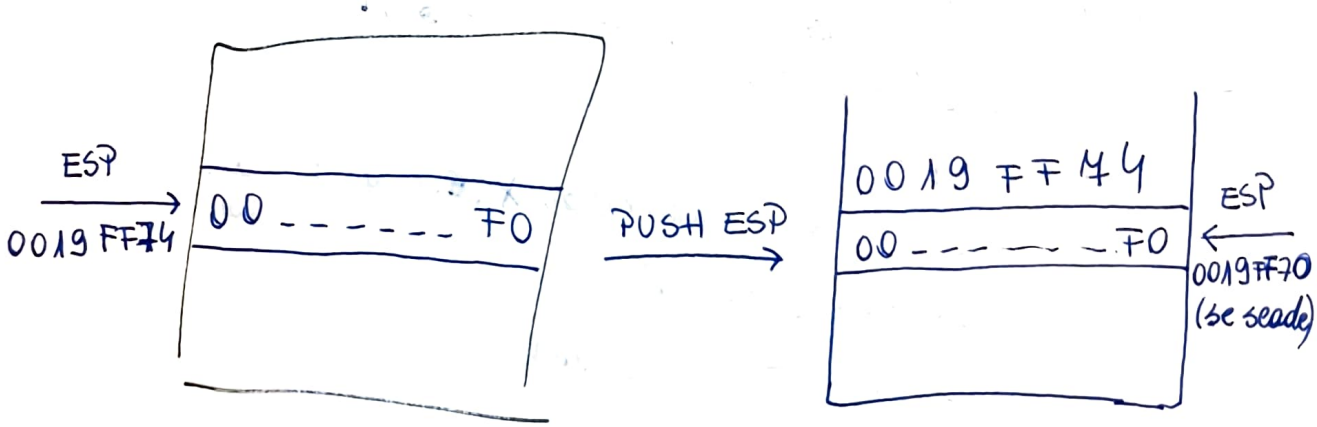
→ implicit operand - exists, but not seen

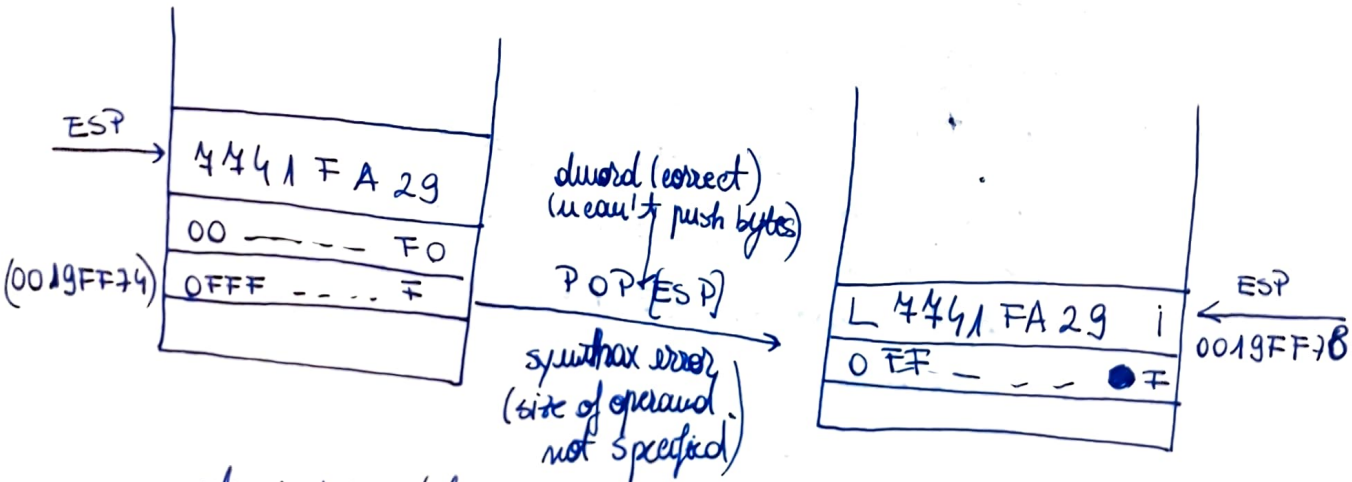
→ explicit operand - from input

→ L-value

→ push source → dest = stack
stack = implicit operand

XLAT → translation





→ evaluezi operandul

→ actualizezi ESP-ul

→ actualizezi operandul

DF48ABFh (+'a' - 10)

AL = 7

↓ (XLAT) overwrites AL with the ascii code of AL (7)
(as an index)

'4'

relative to DS

ES XLAT → then is relative to ES (if not specified ⇒ DS)

registrii ⇒ adresare indirecta

2 DATA SEGMENTS - ES, DS

LEA - load effective address

lea eax, [v] ⇒ mov eax, v (only for direct addressing operands!)

↳ for manipulating addresses in the case of indirect addressing

! lea ~~esp~~ eax, v (why not like this?) because the 2am formula wouldn't work

lea eax, [ebx + v - 6]

ce e eu rosu. → nu ~~vreau~~ ma cele

1) operands interpretation (signed/unsigned)

? interpretation doesn't exist for a processor (only base 2)

only 2 situations

→ 1...
1...
0...
} u can't add 2 neg. numbers and obtain a pos. one (OF = 1)

→ 0...
0...
1
} same (OF = 1)

→ overflow in multiplication:

m positions in base 10
n positions in base 10
---
result - (m+n)

! You never can't have overflow in multiplication

b*b → b
w*w → w

OF = CF = 0

b*b → w
w*w → dd

OF = CF = 1

10 dec: care dintre instr. de mai jos seteaza OF=0, CF=1 (TRICKY)
(at multiplication u can't have diff values at these flags)
only OF=CF

= mul
= imul
- mul

CORECT - none of the above / no overflow

! division by 0 = OVERFLOW (because → ∞)

Curs 10

mov [a], eax

movsx ah, [v] → we cannot fill → SYNTHAX ERROR

movzx ax, [v] ; movzx ax, byte [v] (op size not specified)

mov eax, [v] - syntax error (may be a byte/word, so the ass. doesn't know)

movzx eax, -1 - syntax error

movsx eax, v → syntax error → not a memory address op, or a register ~~reg or const~~ it is seen as a constant (it should be between [])

! address is a dword

mov eax, V

mov ax, V

segment data

Salt

0040 1000h

⋮

Dest

0040 2008h → Dest

fictional subtraction

$d \overset{?}{<} s \Rightarrow d - s \overset{?}{<} 0$

(> / <)

! cmp and test are not comparing instructions (they only do fict. subtr.)

(the comparison is made only when we reach the jumps

→ they just prepare the "area")

"N" - from RET N - remove N bytes from the stack (multiple of 4)

4 std (apelantul)

! std - January

Duinou:

mov eax, 89

- - - - -

jmp Maideparte:

Resd 1000h → Mai aproape: jmp Duinou:

Maideparte:

mov ebx, 17
(add ebx (edx)
Loop Duinou

dec ecx
jnz ecx

→ short jump is out of range
→ max 127 în loop
→ loop → evoare (luia Loop Duinou)
→ consider signed

→ long jump → not restricted to any number
→ short jump → restricted to 127

'x' → size of = 1 character
"x" → size of = 2 ['x' '\0'] string

a 6 dd "123" → exactly this order in memory and completes with 0 till the multiple of 4 (word, dword, etc)

'23' (=) '2', '3'

mov → little endian thing

ASC - curs 11

12.11.2022

Separate compilation means that in a program composed of n text files (source code files) many of them is compiled SEPARATELY at diff moments in time. The final executable file will be obtained as a final step linking together these binary .obj files using the link editor as a tool.

Using directives like #include does not mean sep. compilation and you will not obtain multimodule programming.

* program curs bitdefender

! Exam Which are the shared resources between these 2 modules?

→ concatenare

→ getFinal

→ eax, ebx

∀ 3 possible categories of shared resources

→ the call instruction uses the stack

R: ~~static~~ symbols, eax, ebx and the stack (because call instr pushes the address on the stack and ret...)

→ eax, ecx - volatile resource

→ non volatile resources should be saved

the implementation of calling a subroutine in computer science (title)

Steps: (~~calling a~~ subroutine call)

① Call phase → call code - C

② Entry phase → entry code - C

③ Exit phase → exit code - asm

Call code

everything in asm is our responsability

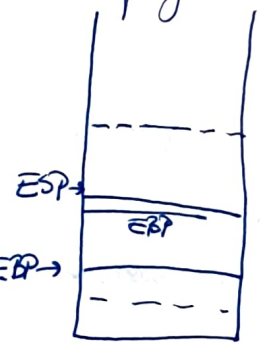
1) Save volatile resources - EAX, ECX, EDX, eflags
2) Just to be careful
3) Passing parameters
4) Perform the ~~code~~ call

[printf] because the acces, it will take only the address

Exam
! phases, explain each, could to să intervină programatorul

Entry code - most complex

a) Building the new stack frame



→ what involves to define a new stack frame ?

EBP has to move (in the place of ESP), but ESP - overwritten so it has to be saved on the stack

```
push EBP
mov EBP, ESP
```

b) Reserving space for local variables

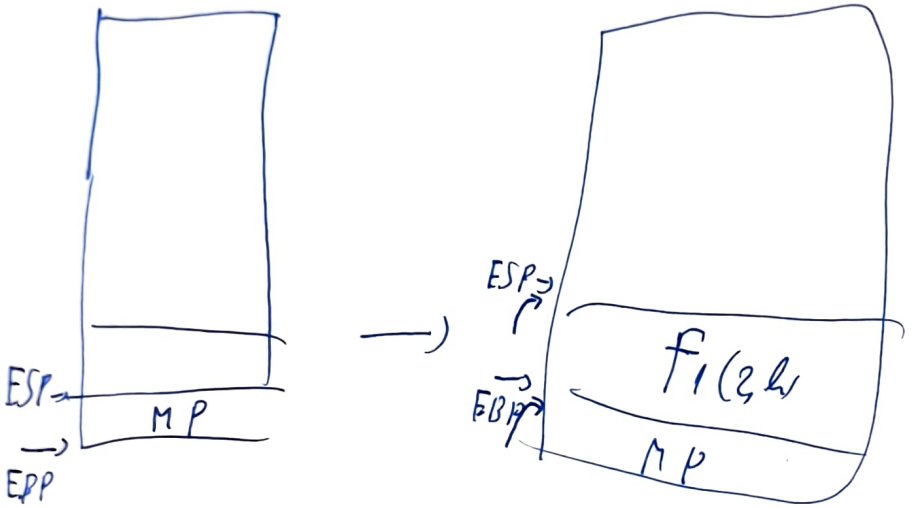
```
sub esp, nr-of-bytes
```

c) Save the non-volatile resources which know that will be affected

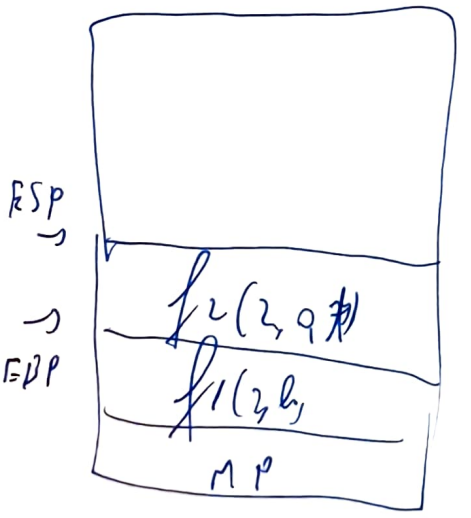
specific offset

Exit code

a) Restore the non volatile resources
b) Release local variable space
c) Deallocate the stack frame
d) Returning step



EBP, ESP always moves to the new function on the stack!!!



- sign bit
- sign extension.
- base 2 representation
  ↳ interpretation for a human being