Asc 8

OBJ FILES – ASSEMBLER

EXE FILE – LINKER

MZ – exe sigature (the initials of a guy from 19??) Mark Zvicovski

STEPS FROM SOURCE CODE TO RUN-TIME

1. Syntactic checking - assembler
2. Obj Files are generated – assembler
3. Linking phase – linker
4. Running the exe
5. The loader is looking for ram and when it finds it it loads the exe file and performs Adress relocation (every adress must be adjusted with info regarded the segment adress which can be done only at loading time)
6. Last step

Data segment begins with 00401000

Code segment begins with 00402000

Segment code

Start:

Jmp Real\_start – 00402000

A db 17 – 00402002

B dw 1234h 00402003

C dd 00402005

Real\_start:

...

Mov eax, c; eax = 402005

Mov edx, [c]; mov edx. DWORD PTR DS: [402005]; normally this means that edx will be assigned with the dword of offset 00402005 taken from DS; my data is not in data segment but code segment so shouldn’t it be from cs?

Utilizarea segmentelor au sens pt noi ca programatori si reprez segmente logice. Ce era logic in programe trb sa fie si in fizic (16 biti).

On 32 bits we are using data adresses represented by segment and offset. The operating system is ignoring it and it maps everything on a huge segment containing only the offset. Toate se vor alinia la acelasi inceput iar singurul lucru care conteaza este 0.

EXAMPLES:

Mov eax, [ebx + esp] – SS because we have esp (cannot be an index, must be only base register) RULE 2

Mov eax, [esp + ebx] – SS because we have esp (cannot be an index, must be only base register) RULE 2; it doesn’t matter in which order you adress

Mov eax, [ebx + esp \* 2] – cannot use esp as index register => syntax error

Mov eax, [ebx + ebp \* 2] – DS; ebp but used as index

Mov eax, [ebx + ebp] – ambiguity, the assembler cannot know exactly, it should apply only the one criteria that can be understood, the order => DS

Mov eax, [ebp + ebx] - ambiguity, the assembler cannot know exactly, it should apply only the one criteria that can be understood, the order => SS

Mov eax, [ebx \* 2 + ebp] – SS because ebx \* 2 and base register si ebp

Mov eax, [ebx \* 1 + ebp] – 2 possibilities: to try to think that ebx \* 1 is ebx from the start => SS, but the idea is in fact that the assembler will see that you use \* as scale in case of ambiguity, so it will think that you as a programmer sugest that you use an index => SS

Mov eax, [ebp \* 1 + ebx] - DS

Mov eax, [ebx \* 1 + ebp \* 1] – the compiler already decides that ebx \* 1 is index, so when it reaches ebp \* 1, it will be clear that that is the base => SS

Mov eax, [ebp \* 1 + ebx \* 1] - DS

Mov eax, [ebp \* 1 + ebx \* 2] – so it first checks that is \* 1, so he decides ebp is index, but then he sees ebx \* 2 that is surely an index so then he goes back to the first one and sees that is multiplied by 1 so it decides that second one is index => SS

Mov eax, [ebp] - SS

Mov eax, [4\*ebp] – DS (?)

Kill me

A face in nasm cu assemblorul e ca si cum ai avea o relatie toxica

Nu stii nimic

Nimic nu are sens

Daca incerci sa intelegi suferi

RELATIE. TOXICA.

CHAPTER 4: ASSEMBLY LANGUAGE INSTRUCTIONS

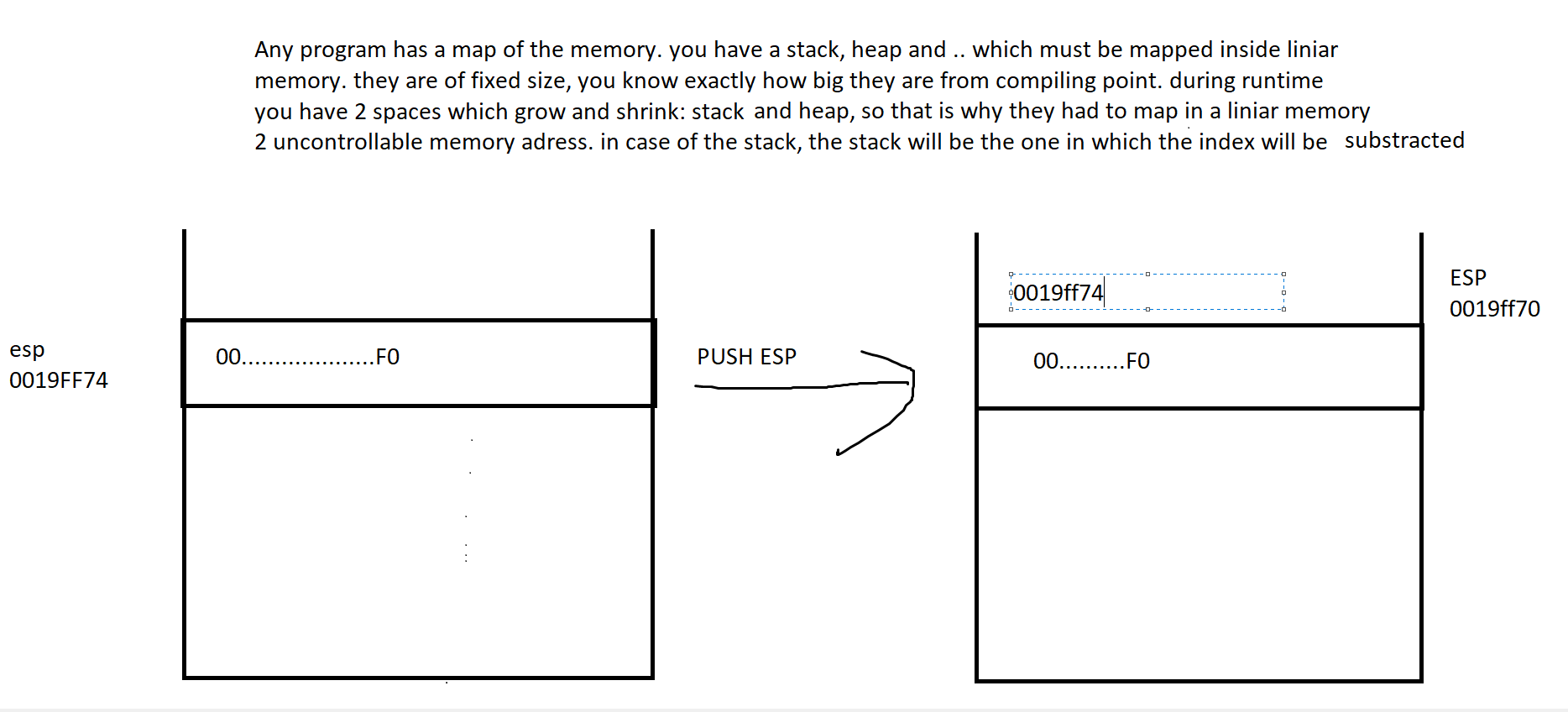
Data transfer instructions.

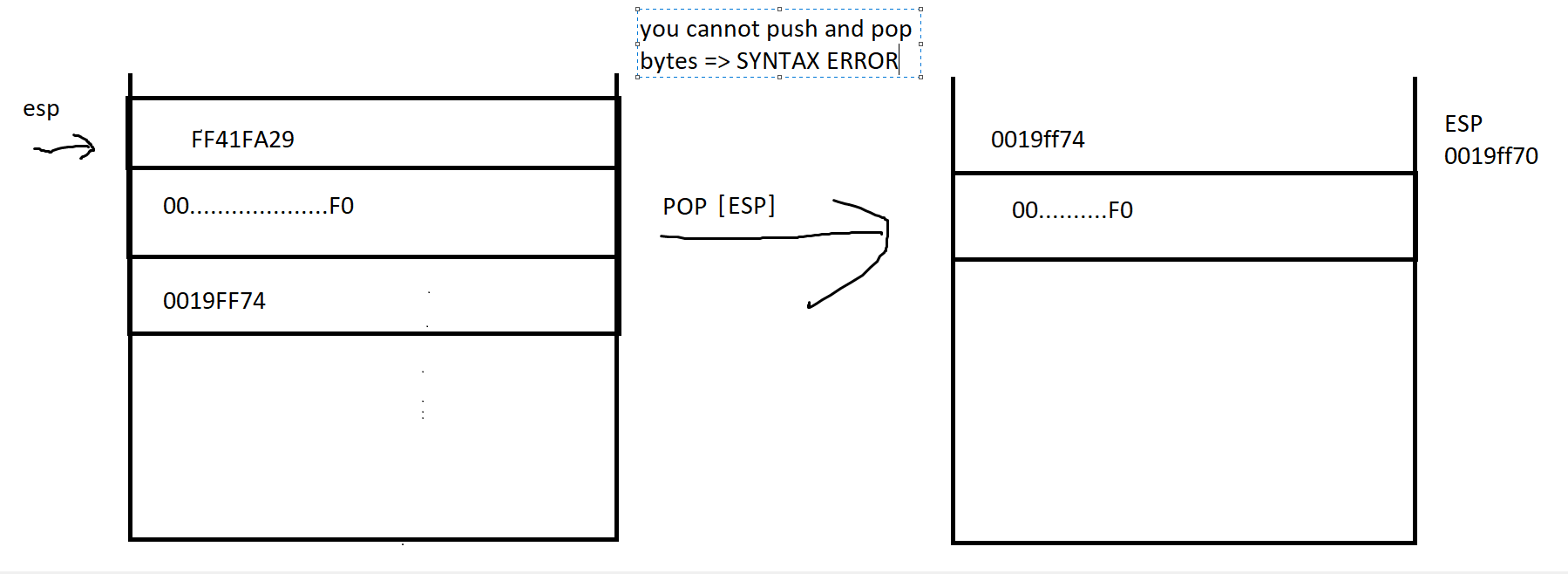
Xchg - Swap procedure – interchange the value of 2 quantities without a 3rd item. If you interchange them , both of them must be assignable (l value)

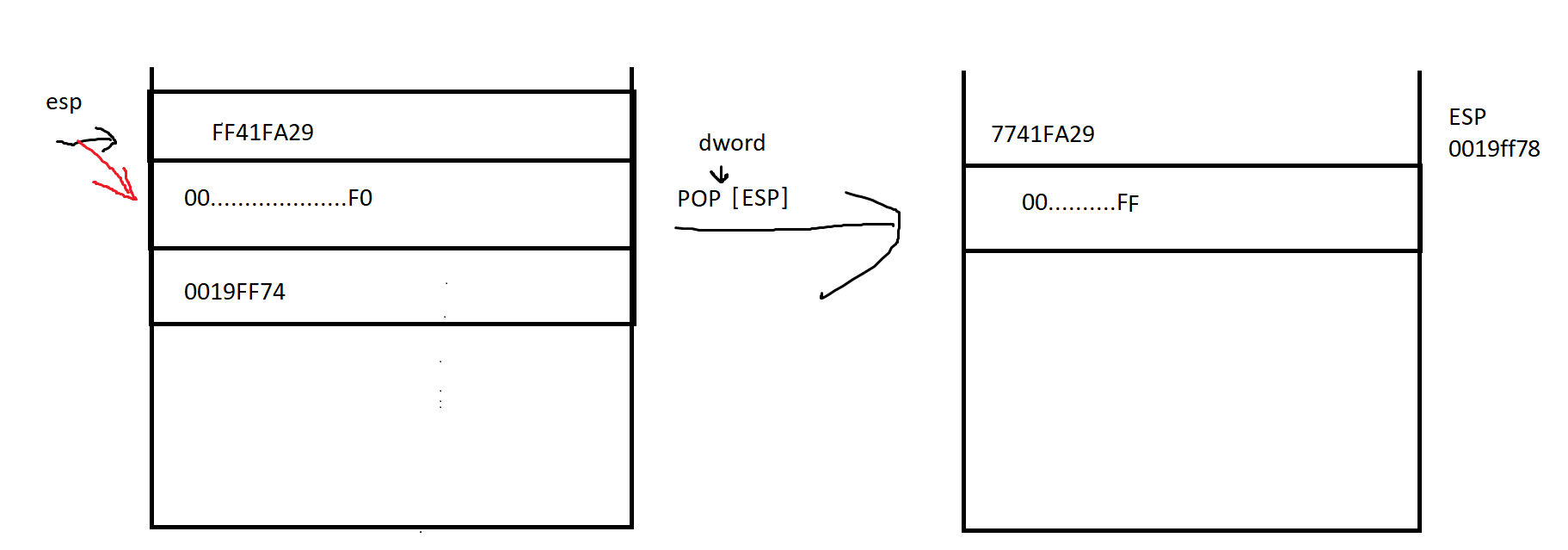
Implicit operands – those that exist

Explicit operands – those that i write

Std – instructions without any existing operands







Lea eax, [v] ⬄ mov eax, v

The reason for which leas was introduces was that only with an instruction like this can you take indirect adresses

Flags instruction (ce e cu rosu nu ne cere) yey

OVERFLOW

* From a physical point of view, it is very clear (mainly through carry flag)
* In case of overflow flag, the idea is that the flag refers to the signed interpretation and you can infer a certain value from overflow flag only by putting an admissable memory variable
* Example:
  + 1001 0011 + 1001 0011 = 1 0100 0110
  + 147 + 179 = 326 (unsigned)
  + 93h + B3h = 1 46h (hexa)
  + -109 + -77 = -186 (signed)