ASC 6

Parts of the assembler

The type of an offset is a pointer type

When you perform “p-7”, you obtain a scalar value/an immediate constant. This still remains a pointer.

The location counter has the pointer type AND is an integer number

The assembly language is simbolic(we put symbols, mnemonics and declare values) ¸but the machine language is not.

The operators are used and evaluated only by the assembler. You will see a table/array with operators that are valid in NASM, to the assembler, not to TASM or processor.

When is every operands evaluated?

Immediate operand – evaluated IMMEDIATELY (scalar type)

Registers operand - cannot be known at assembly time or loading time or linking time, only at run time. Only when the program starts can Oly look at the values within memory and registers.

Direct adress operand (pointer) - the assembler can evaluate it’s value only as an offset. That is the only part you know, you don’t know.

Immediate operand

Every adress allocation is fixed by runtime.

Registers operands

* Can be used as direct operands or a memory operand

The diff of a call and a jump

* A call is a jump to the starting point of a procedurre. But before jumping it saves the returning adress

NASM doesn’t define a procedure, just puts a label and calls. It is defining a logical procedure. Not even fake procedures (like in tasm) exists, there only exists a call function which can allow you as a program to perform a procedure (it basically opens a paranthesys to do your task and when it ends it closes the paranthesis)

Relative to a segment you are returning the value to any operand. When the assembler finishes it’s part, it will output an obj (object) file. All the obj files are taken independently from where they come from(python, c, c++) by the linker and checks their validities. There are specific errors that cannot be known at compiling time, like linking error. If it is alright, the linker will put all obj files at the same place. The actual physical adress is computet at loading time

When you have an isolated offset without the segment part, the offset will associate any segment part based on these rules:

* CS for code labels target of the control transfer instruction
* If in the SIP part you are using the offset specification formula and are using EFP AND ESP as a base, then you admit that you are working with a stack so you are working with an offset that is in the stack.
* If you are not in the condition of (!), (2) you automatically put DS

Using OPERATORS:

* Operators are a mechanism used in defining operands.

[base] + [index \* scale] + [const]

* + are operators
* You cannot determine at assembly time bases, indexes and scales
* THE OPERATORS ACCEPTED BY NASM
  + + unary
  + – unary
  + ~ - complement.
  + ! unary
  + \* binary
  + + binary
  + ! binary
* BIT SHIFTING OPERATORS:
  + ROL
  + ROR
  + SHL
  + SHR
  + Etc

AND (instruction)

& (operator)

x AND 0 = 0

x AND 1 = x

x AND x = x

x AND ~x = 0

|

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

x XOR x = 0

x XOR ~x = 1

WAYS TO MOVE 0

MOV EAX 0

AND EAX,0

SUB EAX, EAX

XOR EAX EAX

a db 17, -1, ‚xyz’

b dw 10001, -128, ‚13’

mov EAX, ![a]; the contents of a variable is not someting that can be determined at assembly time => SYNTAX ERROR

mov EAX, [!a]; Syntax error

mov eax, !a; Syntax error

mov EAX, !(a+7); Syntax error

mov EAX, !(b-a); b-a is a scalar value, so IT IS OK

mov EAX, ![a+7]; Syntax error

mov EAX, !7; EAX = 0

mov EAX, !0; EAX = 1

mov EAX, ~7; EAX = 11.......111000 ; EAX = ffh

mov EAX, !EBX; Syntax error

aa equ 2

mov AH, !aa; It works because it is a scalar value

mov Ah, 17^(~17); AH = ffh

v db ......

a dw ......

b dd ......

push v; OK, stack <- offset of v

push [v]; <-> mov [ss:esp](dword), [v]; SYNTAX ERROR, you don’t know if the target is on 32 or 64 bits

mov eax, [v]; SYNTAX ERROR,

push [eax];

push 15; push dword 15

push [15]; SYNTAX ERROR – memory access operand so

pop [v];

pop v;

pop [EAX];

pop 15; it is a constant and you cannot assign it

mov [v], 0;

mov a, b

mov [a], b

mov AH, b; INCORRECT; you cannot put an offset on 8 bits

mov AX, b; CORRECT

mov EAX, b; CORRECT

mov a, [b]; a is not something assignable

mov [a], [b]; no instruction in assembly can have 2 explicit operands from memory