Expressions & using operators

Operators perform computations only with constant SCALAR values computable at assembly time

working with bits is a mechanism   * operations are done on 64 bits

2 Exceptions : – adding / subtracting a constant from a pointer
- offset computation formula

* ~ is one's complement -> turns 1 into 0 & 0 into 1

! logic negation (0 - false , 1 - true)

<< shift left ; >> shift right   ← operators , NOT instructions

& bit by bit

^ exclusive or

| or ← lowest priority

! $5 | 6 + 7 \& 8 = (5|6) + (7\&8) = 7 + 0 = 7$ NO!

↳ $5 \,|\, (\overset{3}{6}\overset{1}{+}7)\overset{2}{\&}8 = 5 \,|\, 13 \& 8 = 5 | 8 = 13 = 0Dh$

* at exam use parantheses to force the result you want

Bit shifting operators

expression must be computable & evaluatable at assembly time

expression >> how_many   and   expression << how many

! there is no operator for rotate with carry BECAUSE the value of CF is not evaluatable at assembly time

```
mov ax, 01110111b << 3 ;  AX =
mov bh, 01110111b >> 3 ;  BX =
```

the same { & - bitwise AND operator
AND - instruction

$\boxed{X \text{ AND } 0 = 0}$ → most imp
↳ clc

X AND 1 = X

X AND X = X

X AND ~X = 0

* AND is useful for forcing the values of some bits to 0

| - operator	X OR 0 = X	x OR x = x
OR - instruction	$\boxed{X \text{ OR } 1 = 1}$	x OR $\overline{x}$ = 1

↳ STC

* OR is useful for forcing the value of some bits to 1

^ - op.	X XOR 0 = X	x XOR x = 0
XOR - instr	$\boxed{x \text{ XOR } 1 = \overline{x}}$	x XOR $\overline{x}$ = 1

↳ CMC

* XOR is useful for obtaining the complement

Question: Present 15 diff. methods for initializing with 0 the content of a regist.

```
MOV  eax, 0
AND  eax, 0
XOR  eax, eax
 ⋮
     Homework
SHR  32
SHL  32
mov  eax, val ^ val
```

~

a	d?	17, -1, 'a'
b	d?	200, -5, 'xy'

mov eax, ![a]

↑ Syntax error

[a] is a content <u>not</u> determinable at assembly time

mov eax, [!a]

↑ Syntax error

NOT a <u>scalar</u> value

mov eax, !a

↑ still Syntax error

* applying ! to something that is not a scalar is not permissible

```
mov eax, !(a-7)
mov eax, !(b-a)   ✓   → gives a scalar
mov eax, !b - a
mov eax, !7       →   eax = 0
```

! 0 → eax = 1

mov eax, ! ebx → the values of registers are not scalars determinable at assembly time

* ! REGISTERS do not have addresses

aa equ 2
mov ah, ! aa → a label declared with equ does not have an address associat

```
mov ah, ~7   ; ah = F8 h
mov eax, ~7  ; eax = FF FF FF F8 h
```

7 = 00....00 111b

~7 = 11 ... 11 000b

Question : Do F8h on 1 byte & FFFFFFF8h represent the same value in base 10?

A : DEPENDS on signed / unsigned interp.

unsign : F8h = 248
signed : F8 h = −8

two comp. values on 1 byte

```
mov ah, 17 ^ ~17
        X XOR X̄ = 1
```
⇒ ah = 0FF h

```
mov ax, val ^ ~val
```

How does the NASM assembler interprets Datatypes?

types of assembler : TASM / MASM / NASM

↑ turbo ↑ microsoft ↑ netwide

Type operators and operands data types

v	db	...
a	dw	....
b	dd	....

```
push v    ; pushes onto the stack the offset of v
push [v]  ; * in TASM / MASM
```

[v] ↓ syntax error

operation type not specified

push word [v]
dword

mov ah, v → means con –

in NASM : mov ah, [v]

→ push offset v

if: mov ax, v → syntax error, types do not match

* see the audio

in NASM no variable has no datatype, they represent the starting address of a memory area

we use db ... only to have a proper initialization

[base + index * scale + constant] 32 bits formula

in 32 bits programming the STACK is organised on double words

* you cannot push a BYTE or QWORD

have to tell the assembler if you want to push a word or dword

* specifying the size of the expression that follows

byte / word / dword / qword

mov eax, [v] ; go to v and take 4 bytes from the starting point v

* NASM has NO DATATYPES

push 15
↳ a constant is always pushed as a DWORD

pop [v] ; operation size not specified
WORD
DWORD

pop v ; v is an offset / constant ⇒ syntax error
v is NOT an L-value (not smt assignable)
like 2 := 3 ⟶ both are R-values

pop [eax] ; operation size not specified
WORD
DWORD

pop 15 ; not an L-value

pop [15] ; memory violation error
offset sp. formula
with only the constant

* any offset has to be, at the end, a far address

3 rules of associating

CS → ? jmp labels !
ESP/EBP as base → SS
DS otherwise

pop DWORD PTR DS: 15

mov [v], 0 ; operand size not specified
byte, word
dword

There is no situation where you need to use QWORDS

mov [v], byte 0 ; same output OK ✓
div [v] ; operation size not specified
imul [v+2] ; op. size not specified

mov a, b invalid combination, a is NOT an L-value
mov [a], b ← operation size not specified ; offset
word, dword
mov a, [b] a cannot be an L- value

! mov [a], [b] YOU CANNOT have instructions where BOTH operands are from the memory

mul v CAN only have a register or the contents of memory area

mul [v] ; operation size not specified

$\boxed{\text{mul reg / mem}}$

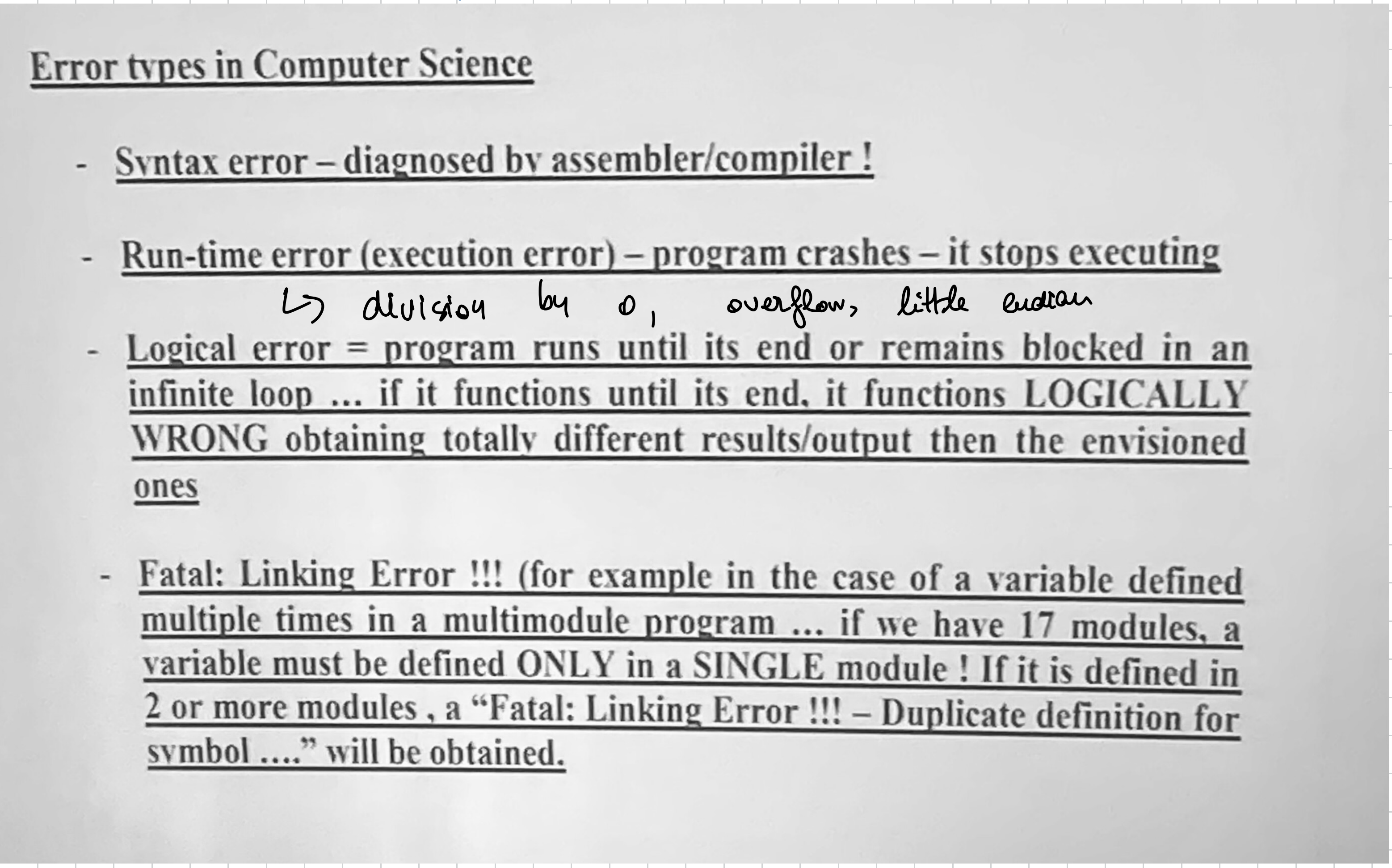
the general case of a memory location is an expression, not the name of a variable

mul eax ✓
mul [eax] ; operation size not specified

mul 15 ; not mul reg/mem → this is a constant

pop byte [v] ; you cannot pop a BYTE, only WORD or DWORD
pop qword [v]

* in DEC 9th you must know Error types



Error types in Computer Science

- Syntax error – diagnosed by assembler/compiler !
- Run-time error (execution error) – program crashes – it stops executing

  ↳ division by 0, overflow, little endian
- Logical error = program runs until its end or remains blocked in an infinite loop ... if it functions until its end, it functions LOGICALLY WRONG obtaining totally different results/output then the envisioned ones
- Fatal: Linking Error !!! (for example in the case of a variable defined multiple times in a multimodule program ... if we have 17 modules, a variable must be defined ONLY in a SINGLE module ! If it is defined in 2 or more modules , a "Fatal: Linking Error !!! – Duplicate definition for symbol ...." will be obtained.

.obj files are the output of the assembler / compiler

↳ LINKER – tool provided by OS to link between modules

↳ .EXE files

see last slide.