تلخيص لغة تجميع ومعالجات دقيقة

إرادة - ثقة - تغيير

CH 2

Parallel processing → That we have Two units work at the same time to increase speed of processing

First one bus interface unit that fitch instruction from memory & store it in place close to micropross.

secound one execution unit that execute the instruct

data bus → carry data info

address bus → carry address info

→ components in BIU

1. segment Registor → base address
2. Instruction pointer (IP) → offset address
3. Address generation adder → convert logical address to physical address (32 bits → 20 bits)
4. Bus control logic.
5. instruction queue → place to store instruction

→ components in EU

1. ALU
2. status & control flag
3. general-perpose Registers
4. Temporary-operand Registers.

* we have 14 Register inside microprocessor each one's size 16-bits

1. Four segment Registers → base address

CS, DS, ES, SS → base address (stack segment)

CS → base address code segment

DS → base address data segment

ES → base address Extra segment

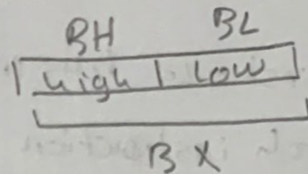
[T]

[3] ~~Instr~~ instruction pointer (IP) → offset address

[4] Four data Registers

AX , BX , DX , CX
Accumulator, Base, Data, Count

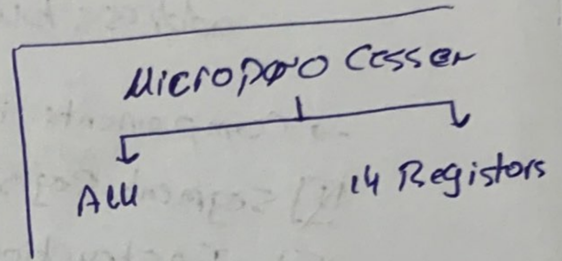
Base ↓ offset address for data seg / Extra



[5] Two pointer Registers → offset address for stack segment

BP , SP → Top of the stack

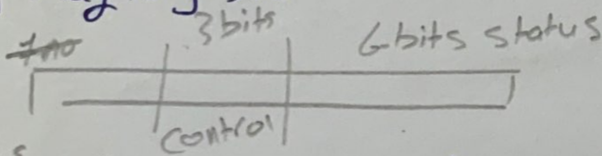
BP → any location within the stack



[6] Two index Registors

SI, DI → offset address for data & Extra segments

[7] Flag Registor



7 bits not used

→ Memory → contain 4 - segments each one have 64 k location

Memory → 1M locations each one store 8bits

$\#$ of location $= 2^{\#\text{ of address bit}}$

$\#$ of address bits $= \log_2 \#$ of location

[1] Code segment { base (CSR), offset (IP) }
↳ to store codes & instructions

[2] Data segment { base (DSR), offset (SI, DI, BX) }

[3] Extra segment { base (ESR), offset (SI, DI, BX) }
↳ to store the data that need to be process

[4] stack segment { base (SSR), offset (BP, SP) }
↳ to store status info

Scanned with CamScanner

→ Even OR odd addressed Word

Aligned word — even address

↳ LSB (0, 2, 4 & 6 --) (for the address)

MisAligned word — odd address

↳ LSB (1, 3, 5 -- for the address)

→ double word (32 bits = 4 bytes)

(0, 4, ---)

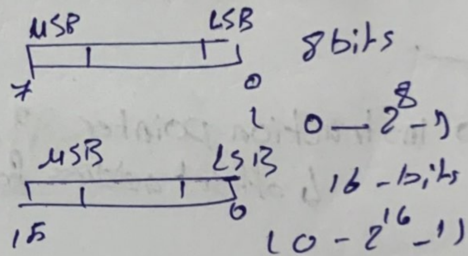
address multiples of 4 → Aligned

not " " " → MisAligned

→ data Types

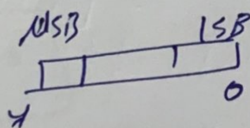
1. integer
   - → signed OR unsigned
   - ↳ Byte-wide OR word-wide integer

* unsigned



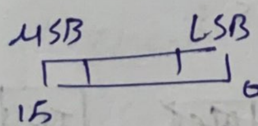
* signed

MSB → 0 → +ve ; ↳ 1 → -ve



8 bits (-2^{8-1} - $2^{8-1}-1$)

MSB → 0 → +ve ; ↳ 1 → -ve



16 bits (-2^{16-1} - $2^{16-1}-1$)

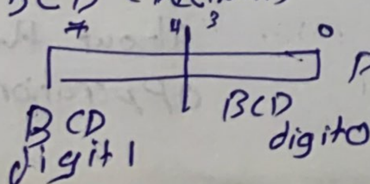
-ve number → 2's complement

1. toggle
2. 100H - □

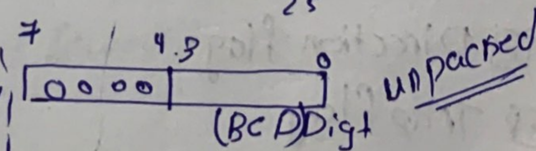
1+

25

2. BCD (Decimal)



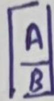
packed | unpacked

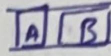


[3] symbol

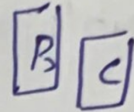
(ASCII) → وجولدا Dec او HEX وبشوف شو رمزها من الجدول وبس ❤

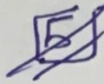
→ memory segmentation.

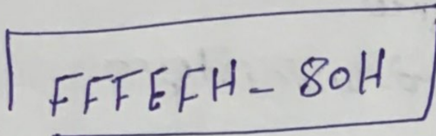
[1] Contigous


[2] adjacent


[3] disadjacent

[4] over lapping






FFFEFH - 80H → هاد الحد هادا اللي بقدر استخدمه بس من ال memory

---

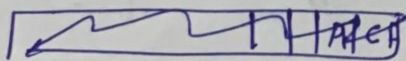
→ instruction pointer → point to the next instruction to fitch
↳ offset address for CS

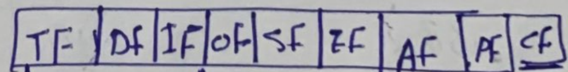
0000H → FFFFH

→ intruction exection queue

max size → 8088 → 4
↳ 8086 → 6

→ status & control flags.





control flags
[1] Interrupt flag
[2] Direction flag
[3] Trap flag

status flags ↓
gives condition about the last opperation

Scanned with CamScanner

→ status flags

[1] carry flag (CF) {
- → 1 اذا كانت مطلع carry برا آخر bit
- ↳ 0 اذا ماطلعش carry برا آخر bit

[2] zero flag (ZF) {
- → 1 اذا كانت ال Result = zero
- ↳ 0 Result ≠ zero

[3] sign flag SF بص لل MSB {
- → 1 MSB = 1
- ↳ 0 MSB = 0

[4] overflow flag OF {
- → 1 carry in ≠ carryout
- ↳ 0 carry in = carryout

[5] parity flag PF بص # of ones لأول 2bits {
- → 1 → # of ones even
- ↳ 0 → # of ones odd

[6] Auxiliary flag AF {
- → 1 اذا طلع borrow/carry في أول 2bits
- ↳ 0 اذا ماطلعش — — —

[physical address = shifted Base (Base0) + offset address] → 20bits

بيستخدم في ال logical address 32bits

| aliases address |

↳ more than one logical address have the same physical in the same segment

→ stacks segment

* differentes between stack segment & others

[1] deal with word data only

[2] starting store data from lowest to highest

Scanned with CamScanner

→ MOV Instruction

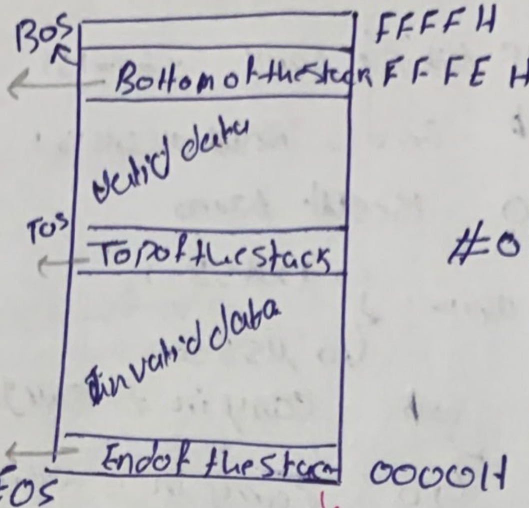
stack

→

highest address word

last data location in which data has pushed

lowest address word



Size = 64 byte

& 32 word

امت ميد

# of Invalid Words

$$= \frac{TOS - EOS}{2}$$

* طلب هادي الرجعة باكيد

→ address SB

POP → inc by 2 byts

→ ابو ميد

push → dec address SB by 2 bytes

Scanned with CamScanner

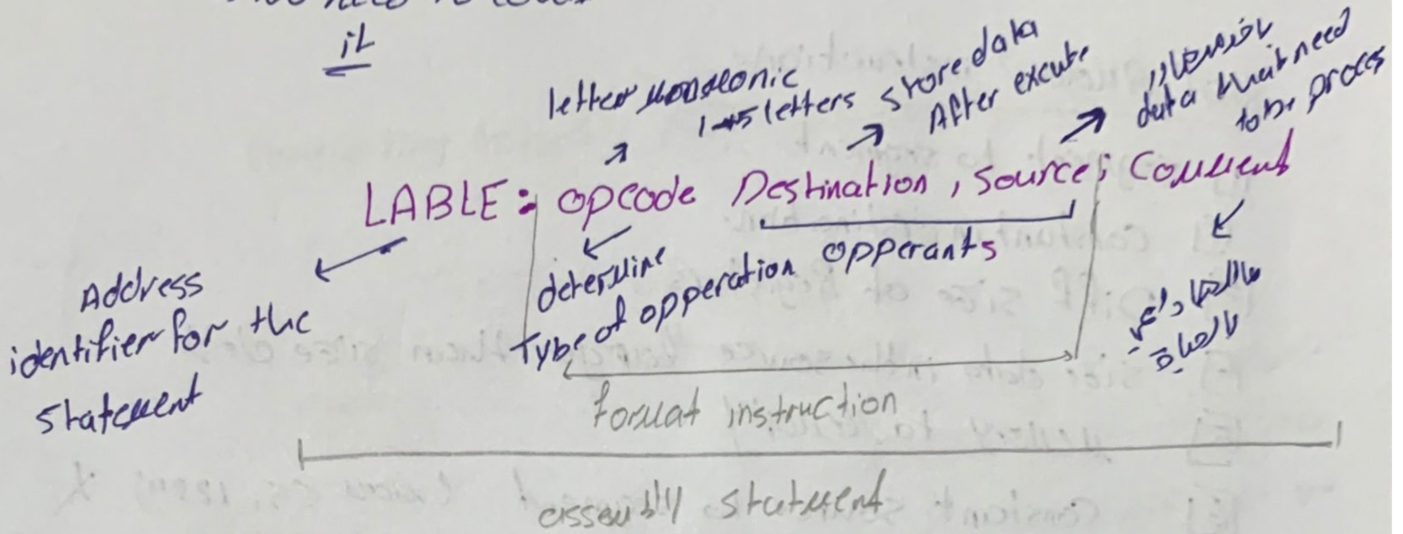
CH3

Instructions

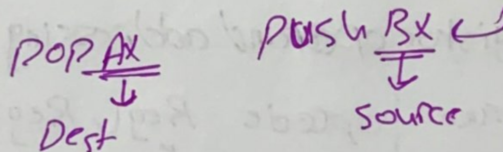
directive (Assembler)
2 convert assembly to machince code

↳ No need to coved it

exulable « Microprocessor »
it's need to convert from Assembly to machine code



لو كان عندي one opperant يحدد هو Dest أو Source حسب ال opcode



---

assembler (assembly code → machine code)

combiler (any language → machine code)

— low level language → use physical component in the processor

high level language → use general variables · that's closer to human

* Both of them must be convert to machine code

* low level language faster than other one

Scanned with CamScanner

→ MOV Instruction

Memmonic	Meaning	Format	Operation	Flags
MOV	Move	MOV D,S	(S) → (D)	none

يعني: Dest ال source ال من data ال copy بعمل

NOT Allowed instructions

1. segment to segment
2. constant in destination.
3. Diff size of Registors
4. size data in the source larger than size dest
5. Memory to Memory
6. Constant source to segment (MOV CS, 1234) ✗

---

* Addressing modes

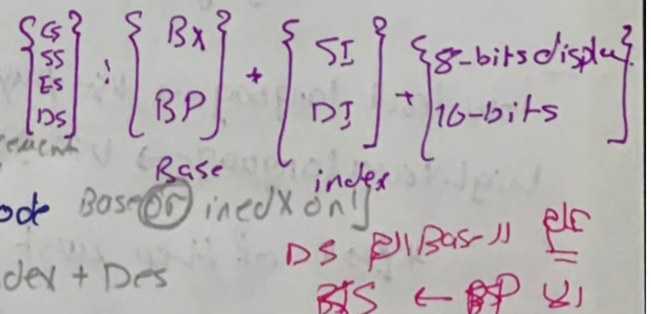
1. Registor operand addressing mode

both Dest & soruce are Registors | opcode Reg1, Reg2 (Reg1 → Dest, Reg2 → Source) — MOV BX, AX

2. Immediate operand addressing mode

Data is constant in source & Dest is Reg | opcode Reg, Imm — MOV AL, 15H

3. Memory addressing modes



↳ Direct addressing mode — displacement only
↳ Registor indirect addressing mode — Base or index only
↳ Indexed addressing mode — index + Dis
↳ Based addressing mode — Base + Dis
↳ Based-Indexed addressing mode — Base + index + Dis

Scanned with CamScanner

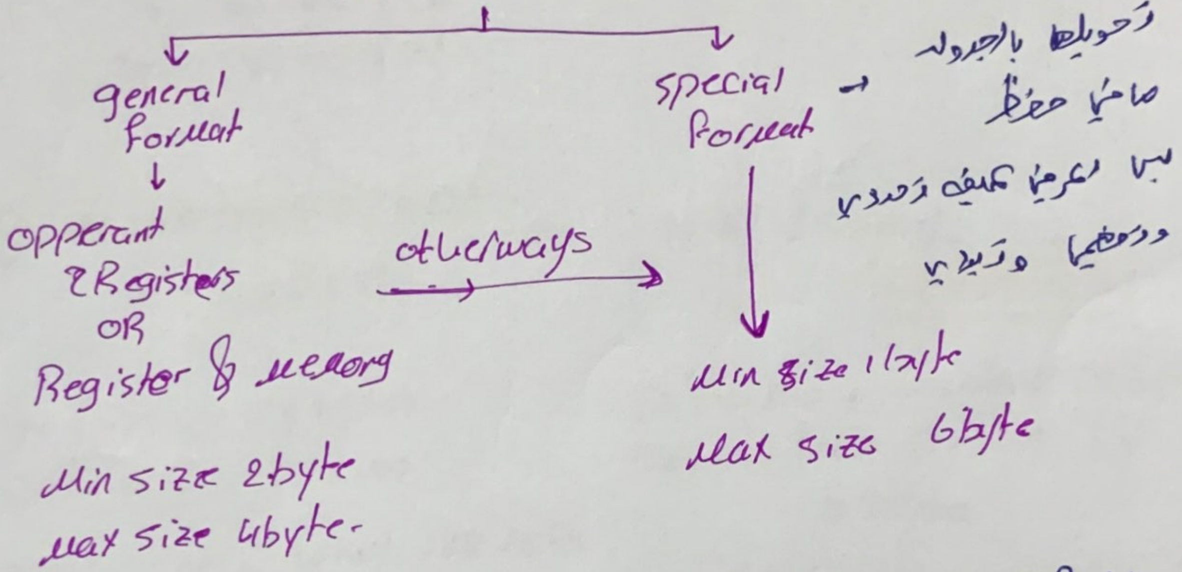
⇒ Advantage of hybrid length

1. allows for many addressing modes
2. allows full size (32-bit) immediate data & addresses

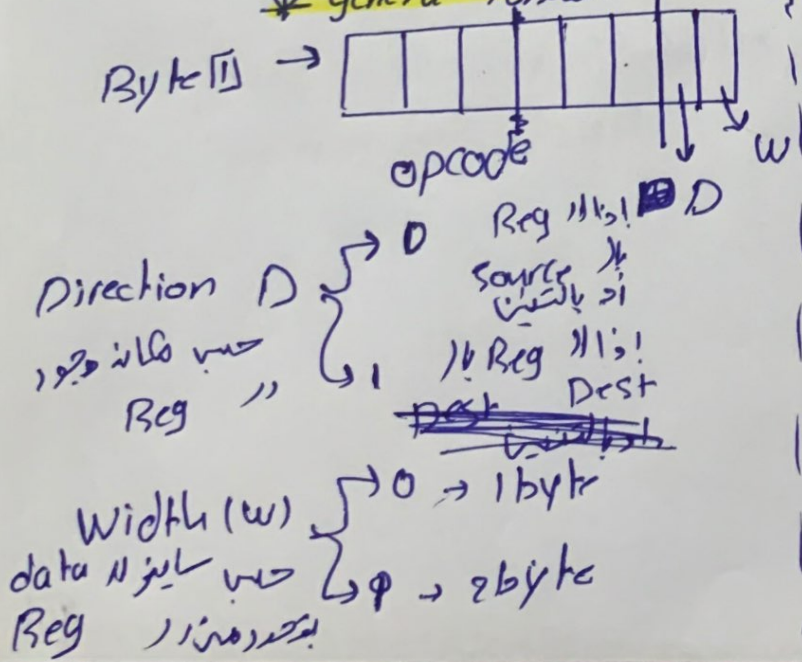
⇒ Disadvantage of variable length

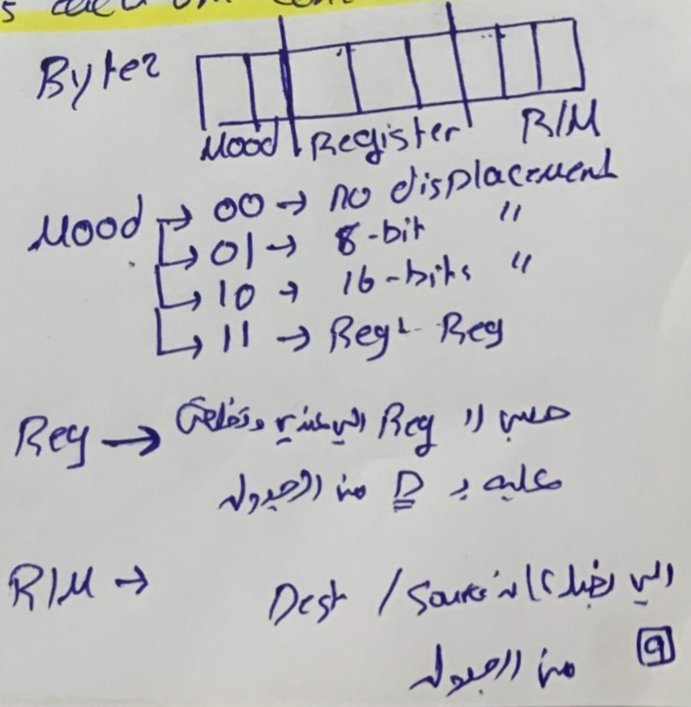
& need more complex assembler

Converting assembly language instruction to Machine Code



* general format → 2bytes each one contain 3 fields





→ DEBUG Program.

↳ An initial state when with the loading of DEBUG

IP → 0100

status Register flag → 0040 (interrup flag = 1)

[1] Register command (R) ⇒ Display & modify instructions

R [Register name]

R AX
↳ AX يعطيني قيمة ال
↳ لو بدي اعدل عليها بس باخذ القيمة الجديدة

R ( )
↳ بعرض كل قيم ال flags + Registors + Machine code

[2] Dump command (D) ⇒ Display / show

D [ADDRESS]

default DS

D → Display 128 bytes starting from DS:0100

D 1373:200 → Display 128 bytes starting from 1373:200

D 1F0 → Display 128 bytes starting from DS:01F0

D 200 300 → Display 128 bytes starting from DS:200 to DS:300

D CS:200 212 → Display 128 bytes starting from CS:200 to CS:212

3 ENTER command (E) → insert data 1byte

E ADDRES [LIST]

Defult → DS

E DS:100 FF FF FFFF

عشان اشوف الي بعدها (space) و (dash) الي قبلها

FFFF انتبه انه ما تكون [احسن عليها سؤال ميد] لازم FF FF (space)

* لو علت ENTER ASCII بدخلو ك ارقام بال Hex

"A" → 41

4 FILL command (F) → insert blocks of data

F starting address Ending address Address List

huge no. of locations

Defult → DS

$$\#of\ locations = (ending\ address - starting\ address) + 1$$

* لو حطيت 33 21 55 1F رح يحط جميع هدول ع كل locations

5 Move command (M) → بعمل copy / blocks of data from one part of memory to another

M starting address END address DEST Address

بين افتار منو — بداية الجزء الي بدي احط عليه

Defult → DS

6 Compare command (C) → بعمل مقارنة ولو انهم متشابهين ما بعرض شي والمختلفين بعرضهم

C starting address end address DEST address

Defult → DS

Scanned with CamScanner

7 Search command (S) ⟹ بتدور عن data معينة

S starting address end address Address list

ادور عليه → الي بدي search

8 INPUT command (I) ⟹ بطلعلي شو فيه ال Address

I ADDRES

9 output command (O) ⟹ بحط ال data insert

O ADDRES BYTE

input → Read
output → write

10 Hexadecimal command (H)

H NUM1 NUM2

NUM1 + NUM2 — ADD
NUM1 - NUM2 — SUB

* لو بدي احسب ال 2's complement H O □ ← مباشر طرح

* لو بدي احسب ال offset address shifted Base physical address H ← مباشر طرح

Scanned with CamScanner

11 UnAssemble command (U)

U [starting address [ending address]]

⇒ Convert from machine code to assembly code

default CS

12 Assemble command (A)

A [starting Address]

⇒ Convert from Assembly language to machine code.

default CS:100

13 TRACE command (T)

T [= starting Address] [NUMBER]

عدد ← NUMBER

مش ضروري

⇒ excute few # of instructions

default CS:IP

14 Go command

G [= starting Address [break point address Address list]]

↳ بعمل excute للـ program

↳ بقدر احطه بـ max لا اقصى 10

Default CS:IP

Scanned with CamScanner

5.1 : Data-transfer instructions (flags مابتتأثر دا) CH5

1 Move instruction

Mnemonic	Meaning	Format	Operation	Flag affected
MOV	Move	MOV D,S	(S) → (D)	None

بتنقل ال data من ال source للـ Destination

* مابتقدر تنقل من Mem ل Mem في MOV
* ومابتقدر تنقل seg ل const في MOV دايركت

بنستخدم Reg كوسيط بالنص

* نفس ال Not-Allowed instructions الموجودين بنتابر ch

Defult DS
عشان احسب address ال memory اذا احتجنا

2 Exchange instruction

Mnemonic	Meaning	Format	Operation	flag effected
XCHG	Exchange	XCHG D,S	(D) ↔ (S)	None

D → S
* بتنقل من Dest الى source
ومن ال source الى Dest
S → D
بدل بينهم يعني

Defult DS
عشان احسب ال Address memory اذا احتجت

* نفس ال Not allowed instructions الموجودين ب ch3
+ ممنوع يكون عندي const in source.

3 Translate instruction

Mnemonic	Meaning	Format	Operation	flags
XLAT	Translate	XLAT	((AL) + (BX) + (DS)0) → AL)	none

هلا لشوفي الفكرة هون انه بدي أحول من ASCII ل EBCDIC وهاد الTable مخزن بال memory

ال BX ← offset to the first element in the table

وال AL ← بكون عرفتها قبل مثلا mov AL, 41H

بجمع AL + BX + DS0 =
ال address الي بطلع

بروح ع ال memory دبشوف شو ال value تبعه
بتكون هادي ال new value for AL.

Scanned with CamScanner

4 load effective address & load full pointer instructions

Mnemonic	Meaning	Format	Flags effecte
LEA	loadeffectiveaddress	LEA Reg16, EA	none
LDS	load Register & DS	LDS Reg16, Mem32	none
LES	load Register & Es	LES Reg16, Mem32	none

1 LEA

LEA Reg16, EA

LEA Bx, [1234]

Bx = 1234

* بتعبير قيمة ال EA = Reg

2 LDS

LDS Reg16, Mem 32

First 16 bits → Reg16

Secound 16 bits → DS

LDS Bx, 12345678 H

Bx = 1234 H

DS = 5678 H

3 LES

LES Reg16, Mem 32

First 16bits → Reg16

sec 16bits → Es

عشان Defult
اسم memory // address
DS

* Advantage → we can load full pointer

5.2 :- Arithmetic instructions

I Addition instruction

• ADD

mnemonic	meaning	Format	operation	flags effected
ADD	Addition	ADD D, S	(D) + (S) → D	CF, SF, PF, ZF, OF, AF (all)

⇐ جمع عادي :)

• ADD With carry

mnemonic	meaning	Format	operation	flags effected
ADC	Add with carry	ADC D, S	(D) + (S) + (CF) → Dest	CF, SF, PF, ZF, AF, OF (كلهم)

CF → حسب الـ instruction الي قبلها

• Increment

INC	increment	INC D	(D) + 1 → Dest	OF, ZF, PF, SF, AF CF ✗ ما بتغير

• ASCI adjust for adition.

				flags effct
AAA	↓	AAA		CF, AF → only الباقيا ما بتغير

لازم قبلها يكون عندي عملية جمع عادية

الـ AAA بحوله , ASCII وبحطه بالـ ASCII ← بيخزنه بالـ Result AL, AX حسب الـ size

• Decimal Adjust for addition - DAA

نفس الـ AAA بس هون بحوله للـ Deci وبخزنه بـ AH يا AL

كل الـ flag بتغير الا الـ OF

Scanned with CamScanner

5.2 - subtraction instructions

Mnemonic	Meaning	Format	Operation	Flags effected
SUB	subtraction	SUB D,S	(D)-(S) → Dest	CF, ZF, PF, OF, AF, SF
SBB	subtract with borrow	SBB D,S	(D)-(S)-(CF) → Dest	كلهم
DEC	Decrement by 1	DEC D	(D) -1 → (D)	كلهم الا CF
NEG	Negate	NEG D	0 - D → (D) 1 → CF بصيغة الـ 2's comp	كلهم
AAS	ASCII adjust for SUB	AAS		CF, AF
DAS	Decimal adjust for SUB	DAS		كلهم الا الـ OF

نفس الكلام زي الـ بال Addition

3 Multiplication → مطلوب اعرف انهم بغيروا ال flags

1 MUL
↳ unsigned multiply

MUL S

$AL * S8 \rightarrow AX$

$AX * S16 \rightarrow DX\ AX$

بختار AX / AL حسب size (S)

بجاد الضرب مابتطلع على الاشارة وبكونه ضرب عادي (استخدم الكالكوليتر أسهل)

2 IMUL
↳ signed multiply

IMUL S

$AL * S8 \rightarrow AX$

$AX * S16 \rightarrow DX\ AX$

بختار AX/AL حسب size source

← هون لازم انتبه على ال signed حسب MSB
- 0 → +ve
- 1 → -ve → 2's complement

بعد ما احول ال -ve ل 2's complement #

بضرب وبعديها برجع باخذ 2's complement للجواب

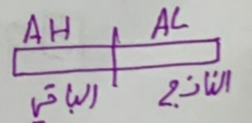
- اذا كان واحد بس -ve → AX = Final Result ال
- اذا التنين -ve / اذا التنين +ve → ال Result هو ال AX

4 Division → مطلوب اعرف انهم بغيروا ال flags

1 DIV
↳ unsigned

DIV S

$AX / S8 \rightarrow$



$DX\ AX / S16 \rightarrow$

AX → الناتج

DX → الباقي

الباقي = AX - (الناتج × S8)

الباقي = DXAX - (الناتج × S16)

استخدم عالكوليتر اسهل وعشان تعرف الباقي

2 IDIV
↳ signed

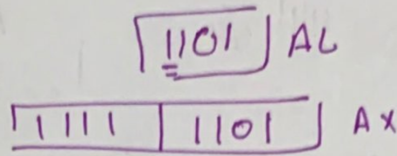
بشوف شو الاشارة حسب MSB
- 0 → +ve
- 1 → -ve → 2's complement

اذا واحد بس -ve الجواب بعد القسمة باخذله 2's

اذا ال -ve في مباشر 2's

* CBW (convert byte to word)

لما بشوف ال MSB لل AL وبخلي كل AH نفسها



* CWD (convert word to double word)

لما بشوف ال MSB لل AX وبخلي كل ال DX نفسها

AX = 00110001
DX = 00000000

5.3 logic instructions → كلهم بغيروا الflags الا NOT

1. AND

AND D,S D.S → D

0	0	0
0	1	0
1	0	0
1	1	1

X.1 → no change
X.0 → Reset (0)
Mask

2. OR

OR D,S (D)+(S) → (S)

0	0	0
0	1	1
1	0	1
1	1	1

3. NOT

NOT D ($\bar{D}$) → D
0 ⇆ 1

X+1 → 1 set
X+0 → no chang

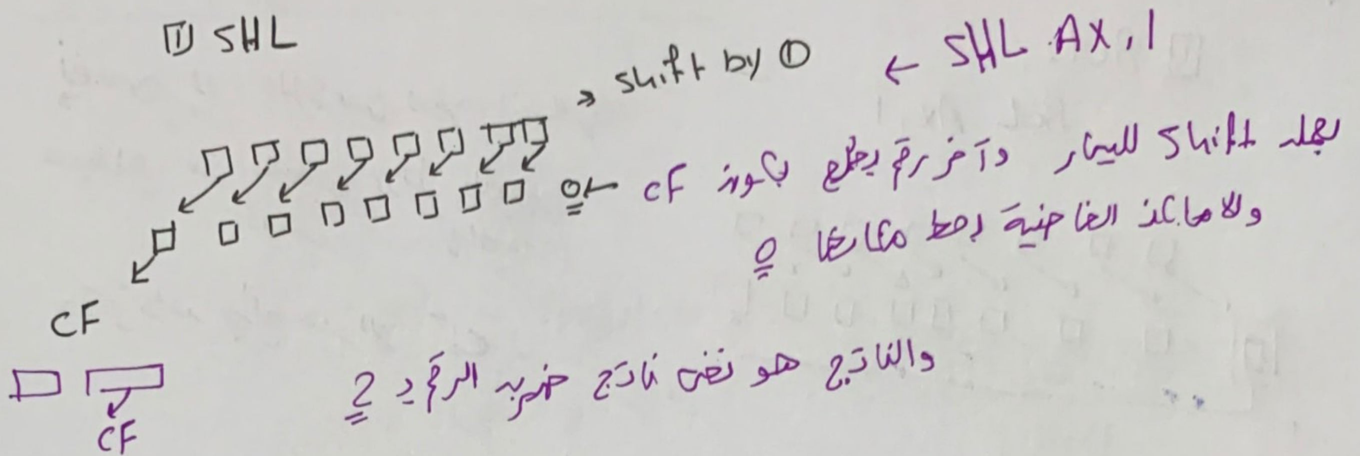
4. XOR

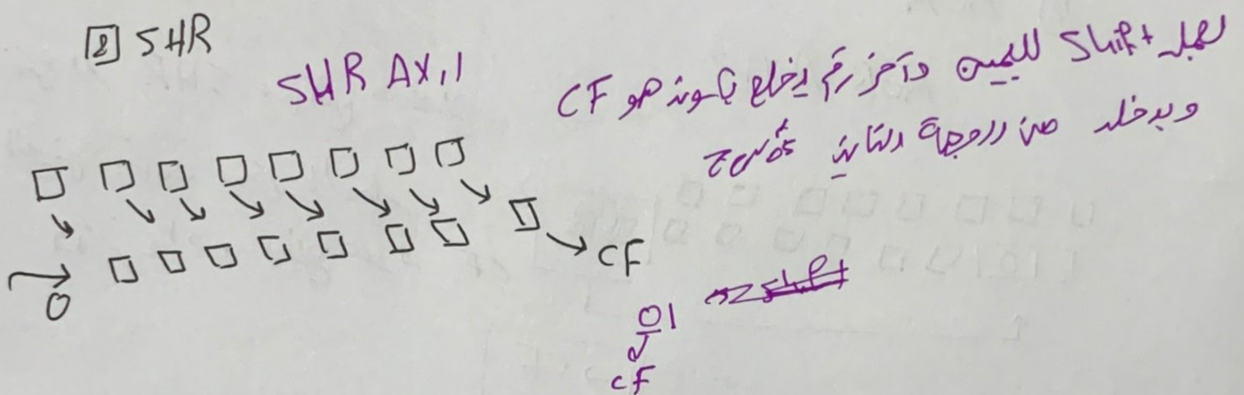
XOR D,S D ⊕ S

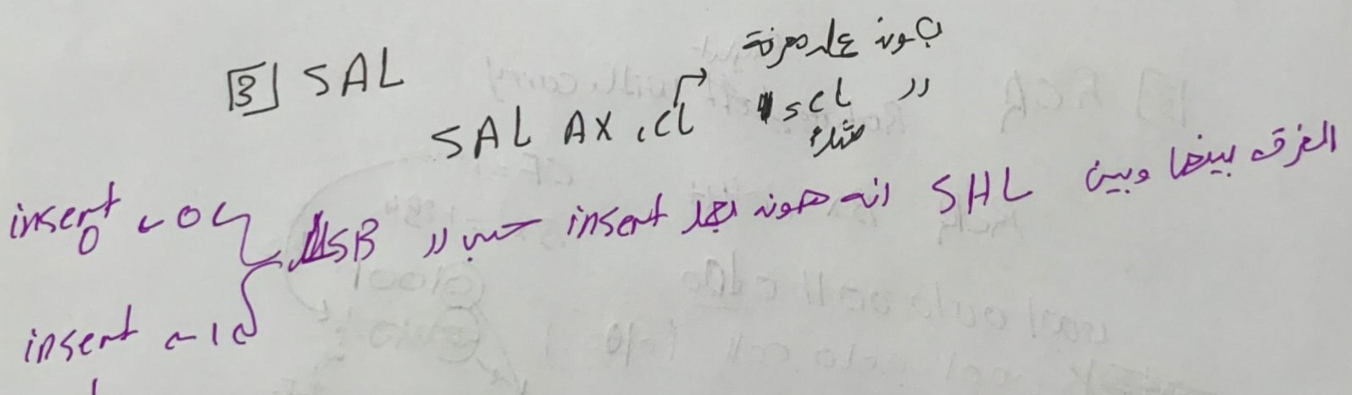
0	0	0
0	1	1
1	0	1
1	1	0

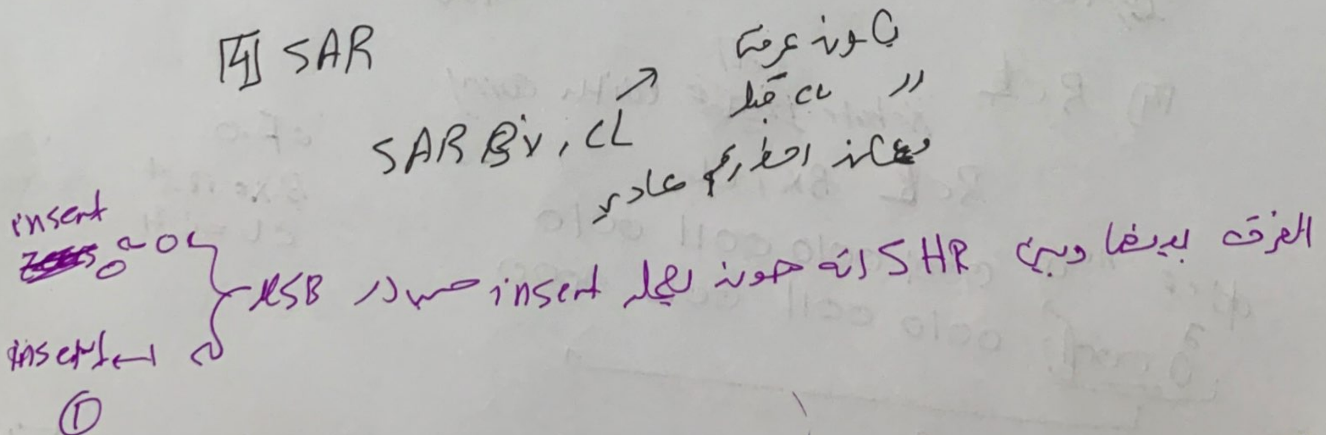
المتشابهين ← 0
المختلفين ← 1
one's → even
0 ← لو
one's odd → 1

5-4 shift instructions ⇒ بعني اعرف انه ال flags بتغيروا









Scanned with CamScanner

5.5 Rotate instructions ⇒ بتغير ال CF

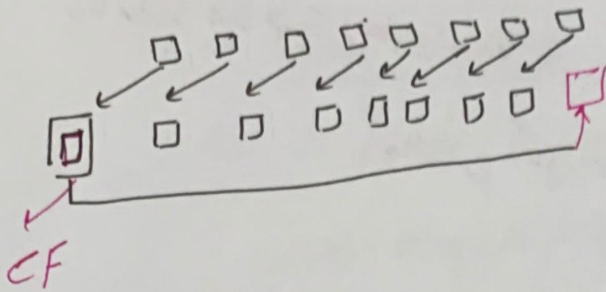
[1] RoL

RoL Ax, 1

نفس ال SHL بس الفرق انه اللي

حيطلع من اليسار يدخل باليمين

وآخر واحد بطلع هو ال CF



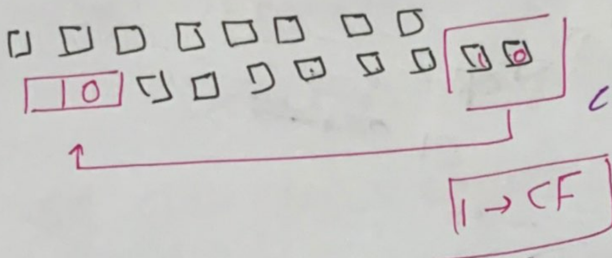
[2] RoR

RoR Ax, 2

* نفس ال SHR بس اللي حيطلع

من اليمين نفسه بيدخل باليسار

وآخر واحد بطلع هو ال CF

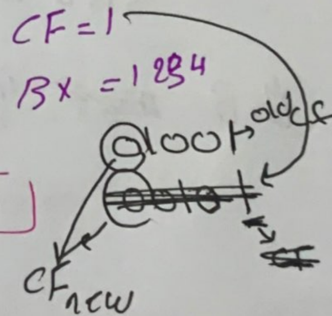


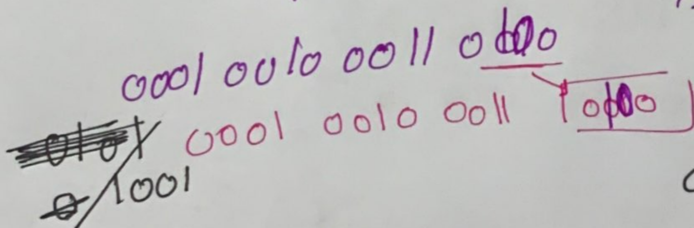
[3] RCR Rotate Right with carry

RcR BX, 04H

CF = 1

BX = 1234





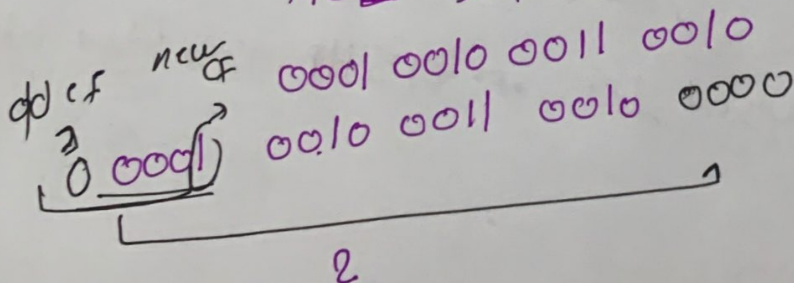
[4] RCL Rotate Left with carry

RcL BX, CL

cF = 0

BX = 1234

CL = 04

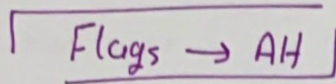


Scanned with CamScanner

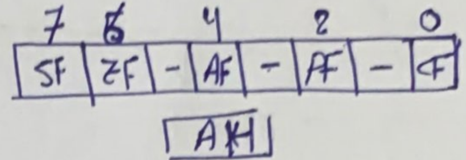
<u>CH6</u>

6.1 Flag - Control instructions ⇒ هدول ال instructions بتغيرها بال flags بطريقة مباشرة

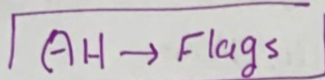
1. Load AH from Flags (LAHF)



بياخذ من ال flags وبحطهم بال AH
لا يعني قيم ال flags مابتتغير



2. STORE AH into Flags (SAHF)



بياخذ من AH وبيسيف بال flags
لا يعني قيم ال flags بتتغير

3. clear carry (CLC)
   ↳ CF=0

4. Set carry (STC)
   ↳ CF=1

5. Complement carry (CMC)
   ↳ $CF = \overline{CF}$
   0 ⇆ 1

* CLC followed CMC → STC
* STC followed CMC → CLC

6. clear Interrupt (CLI)
   ↳ IF=0

7. Set Interrupt (STI)
   IF=1

8. Clear direction (CLD)
   DF=0

9. Set Direction (STD)
   DF=1

10. → هوصلنا بس اجا ع بالي ♡

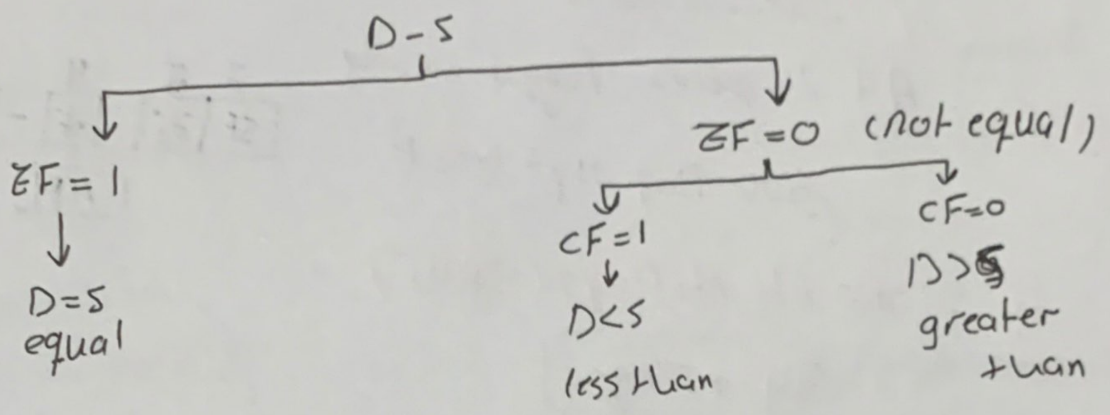
Scanned with CamScanner

6-2 Compare Instruction

CMP D,S

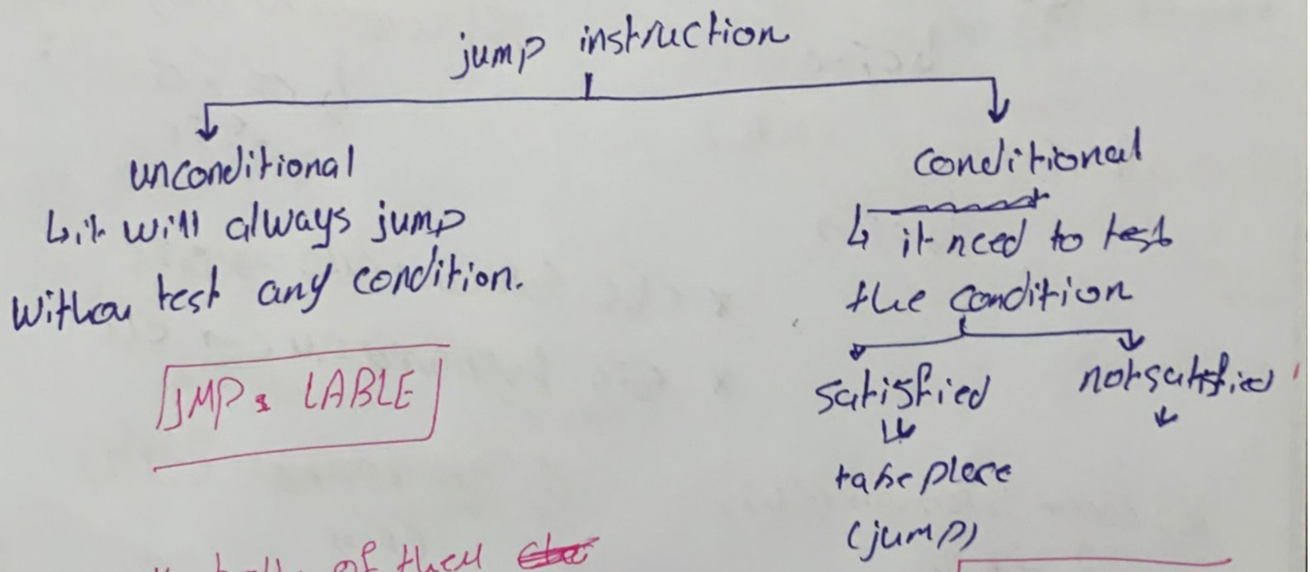
(D) - (S)

all flags will be effected
ZF, CF, PF, SF, AF, OF



6.3 Control flow & jump instruction

↳ change the flow of instruction



* both of them
↳ change IP in the same segment

* لازم يكون عندي LABLE بال program
عشان اعمل JUMP عليه

Scanned with CamScanner

JMP / JCC LABLE

Intrasegment Jump
↳ within the same code segment (change IP only)

↳ [1] short lable
(lable written by 8-bits)

$IP_{new} = IP_{old} + 8\text{-bits}$

↳ [2] near lable
(size ⇒ 16 bits

IP_{new} = near lable

↳ [3] Reg ptr 16

IP_{new} = Value of the Register

* ~~IP new~~ lable BX

For Ex (BX=100)

~~lable~~ lable=100

↳ [4] Mem ptr 16

IP_{new} → physical address بجيب ال

وباخذ ال 2 byte من ال data لحطهم في ال IP_{new}

Intersegment Jump
↳ to another code segment (change IP & CS)

↳ [1] far lable (intersegment)
32-bits immediate

0 - 16 bits → IP_{new}

17 - 32 bits → CS_{new}

↳ [2] Mem ptr 32 (intersegment)
عشان افرقه بينها وبين ال Mem ptr 16

JMP DWORD PTR [DI]

DI . DI+1 → IP_{new}

DI+2 . DI+3 → CS_{new}

هلا بدي أتعلم كيف أحوله من ال If-then-else ال C++
لـ Assembly code باستخدام JMP + JCC

بال Assembly بربط الكود حسب المطلوب وما في شي ثابت
رح أحطلك مثال واشرحه عشان تتذكري القاعدة

C++
if
==
else
==

if
A == B
B++
else
B-- ← بدخل حالها دائماً

```
CMP AX , BX      → افضل شي لا تعرف انو (=) ولا مين (CMP)
    JE   first   → اذا كانو equal حيروح على first
    JMP  END     → عشان يطلع من ال loop
    JNZ  secound → في حال ما كانو (=) حيكمل طبيعي
    JMP  END     ← عشان يطلع
secound : DEC BX → لازم أكون معرفها
first : INC BX   → لازم أكون معرفة first وحاطة فيها الي بدي انفذه لو طلع equal (if)
END : ---
```

(ديو حط لهون بعدين بيروح secound ... وحاطة ال else)

* أقرف تشابتر بالمادة ♡
احفظ سنوات ذخطل

→ loop program structure

I Repeat until structure

CL → عدد مرات التكرار
ورح تضل ال loop تتكرر لحتى
CL = 0
وطبعاً خلاله تكرارها بنفذ ال instructions الي موجودين بال code

* ال form الثابت

```
MOV CL, Count
Again : ---
        ---
        ---
        DEC CL
        JNZ Again
```


Scanned with CamScanner

[3] While do structure.

نفس فكرة ال Repeat في عداد ال CL بس الفرق

انه بال Repeat بكون الشرط مفروض بالاخر (JNZ gain)

هو (do while ، الشرط بالاول (JE next) لانه لما يوصل CL = zero

بتطلع من ال loop

JE → بعمل jump اذا كان zero

JNZ → بعمل jump اذا لا تساوي zero

JE → بعمل jump اذا كان equal

JC → بعمل jump اذا ال CF = '1'

6.4 → ما لخصته لأني ما فهمته ♡

6.5 the loop & the loop - handling

[1] loop : loop while not zero

CX ≠ 0 → برجع لجديد ال loop وكل مرة CX - 1

[2] loopE / loopZ → loop while equal

CX ≠ 0 repeat while count not zero

ZF → بشوف ناتج [ZF] اللي طلع من ال instruction اللي قبل loopZ / loopE

اذا CX ≠ 0 و ZF = 1 بعمل loop

[3] LOOPNE/LOOPNZ $CX \neq 0$ & $ZF=0$

ناتج من ال instruction اللي قبل LOOPNZ / LOOPNE

* لازم احط مع ال loop ← lable ويكون short lable

* اول شي بشوف شو ال loop المستخدم

وبتحقق من شروطه اذا متحققين بعمل ال loop اذا لا بطلع منه دايرة وصلى الله وبارك

6.6 String & String-Handling instructions.

قبل كل شي لازم نعرف ال (DF) Direction flag

DF
- 0 → بقرا للامام → forword ↓
- 1 → للخلف → back word ↑

وبعدل عليه من خلال

$DF=0$ ← CLD

$DF=1$ ← STD

[1] Move String (MOVS)

MOVSB / MOVSW

بنقل ال Data من (DS:SI) ل (ES:DI)

وبعدل على قيمة SI و DI

$(DS)0 + SI \rightarrow (ES)0 + DI$

$SI = F1 \text{ or } F2$

$DI = F1 \text{ OR } F2$

1 OR 2 → B / W

Byte ← 1 ، word 2

F → DF
- 0 ↓ للأمام
- 1 ↑ للخلف

[2] Compare / Scan string (CMPS / SCAS)

CMPSB / CMPSW

DS:SI - ES:DI

(DS)0 + SI - (ES)0 + DI →

DI = DI ∓1 OR ∓2

SI = SI ∓1 OR ∓2

طبعاً بيعني التغيير على ال ZF

ونواتج الطرح ما نخزنها

• Scan

SCASB / SCASW

AL OR AX - ES:DI

AL OR AX - (ES)0 + DI

AX → Word

AL → Byte

بيعني التعديل على ال flag

[3] Load string

LODSB / LODSW

DS:SI → AL OR AX

(DS)0 + SI → AL OR AX

بينقل من ~~DS ل DI~~ DS:SI ل AX OR AL

SI = SI ∓1 OR ∓2 → W / B

DF 0 / 1

طبعاً نفس قصة ال

[4] Store string

STOSB / STOSW

بينقل من AL OR AX ل ES:DI

AL OR AX → ES:DI

AL OR AX → (ES)0 + DI

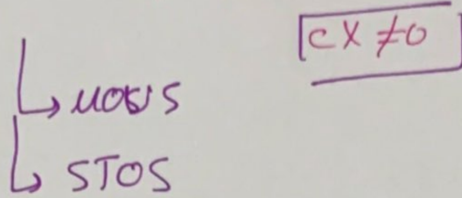
DI = DI ∓1 OR ∓2

Scanned with CamScanner

* Repeat string

ال Repeat بتجي مع وحدة من الي قبل.

[1] REP ⟹ Repeat while not end of string



لو اجت مع ال MOVS مثلاً

[1] بعمل ال MOVS وبنقل DS:SI → ES:DI

[2] CX − 1 = CX

[3] بشوف الشرط CX ≠ 0

- CX = 0 ← بوقف
- CX ≠ 0 ← بكمل

[2] REPE/REPZ ⟹ CX ≠ 0 ZF = 1

- CMPS
- SCAS

[3] REPNE/REPNZ CX ≠ 0 ZF = 0

- CMPS
- SCAS

* خلصنا أطول مادة بالحياة 😮

مصطفى العافية

Scanned with CamScanner