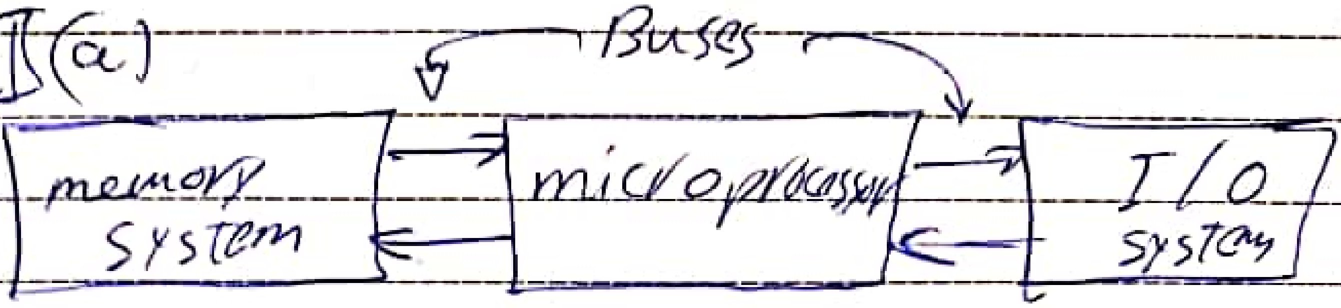
final 2011

II (a)



real mode:-

- is devide memory for segmant
- It is original mode.
- Memory is accessible by any program
- no multiple program run in memory at the same time.
- use only 1 MB of memory.
- where Dos operate.

protected mode:-

- use all memory - support virtual M.
- where windows operate.
- Memory isn't accessible by any program running in memory.

FTU

PIXELOGIC

$\boxed{2}$ (a) Ax 0100
bx FCFF

(b) to convert from 2's complement to decimal

- represent the hex number in binary.
- look for the MSB if ⌈0 → positive ⌊1 → negative

- if positive convert binary to decimal
- if negative optain the 1's complement of binary then add 1

0100
0100 = 00000001 00000000
MSB = 0   number is positive
= 256

FCFF

FCFF = 1111 1100 1111 1111
MSB = 1   the number is negative

Not(1111 1100 1111 1111) = 0000 0011 0000 0000
+1 = 0000 0011 0000 0001
number = -~~7~~69

3

(a) nibble ⇒ 4 bit   Ax ⇒ 16-bit

ال inst بيعمل set(1) هو OR

OR AX, 0F000H

الرقم عشان يعملهم [illegible]
كابيتال

(b) xAND0 = 0   ال inst بيعمل clear هو AND

AX = xⓧxⓧ xⓧxⓧ xⓧxⓧ xⓧxⓧ
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

AND AX, 0AAAA

(c) XOR 1   ال Inst بيعمل invert هو XOR

AX = ⓧxxx xⓧxx xxⓧx xxxⓧ

XOR, AX, 8421 H

(d) ال Inst بيعمل change لل sign هو ال NEG
يطلع ال 2's complement

NEG AX

[6] ~~org 7F~~ [4] e

```
org 100h
mov cx,97
mov si,0

fib:
mov ax, list[si]
mov bx, list[si+4]
add ax, bx
add ax, list[si+8]
mov list[si+12], ax
add si, 4

Loop fib
ret
list dd 0,0,1 97 dup(0)
```

[5] ECX ⇒ 30 FD1C

~~EA97 ⇒ A97E~~

EAX ⇒ FA 75 1C 56

SS ⇒ 1C 2e9

12
9F
1B
FA
75

(4) [3]

PIXELOGIC

$\boxed{5}$ SS=C300

SP=F200

ECX= 30 FD 1C 56

EAX= A9 7E 42 B8

push cx (push 1C 56)

pop EAX

1C	SP= F198
56	
75	SP= F200
FA	
1B	SP= F202
9E	
12	
CC	

(a) ECX ⇒ 30 FD 1C 56

(b) EAX ⇒ 1C 56 75 FA

(c) SS ⇒ C300

(d) SP = F200-2+4= F202

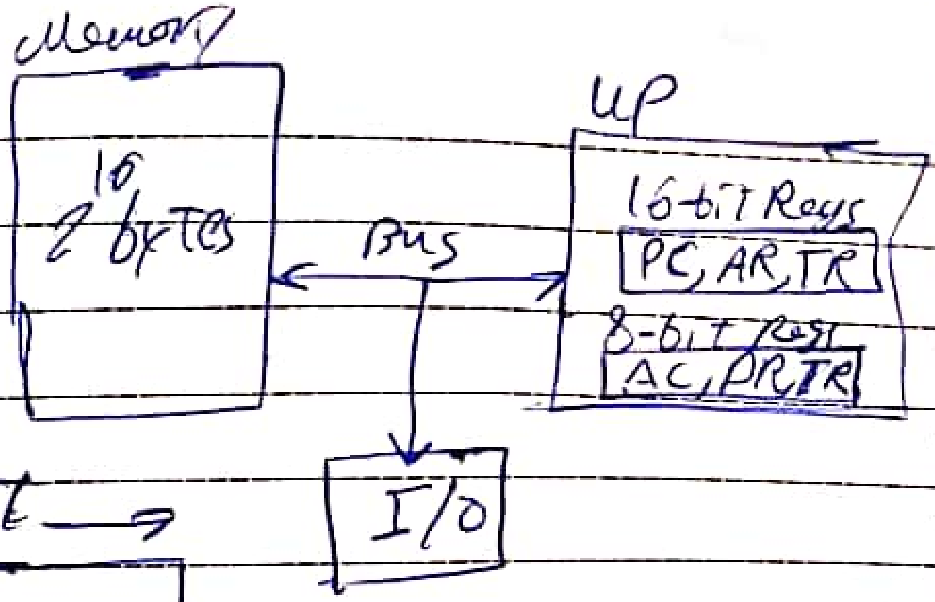
Push 2 Pop 4

(e) EA= SS*10 + SP

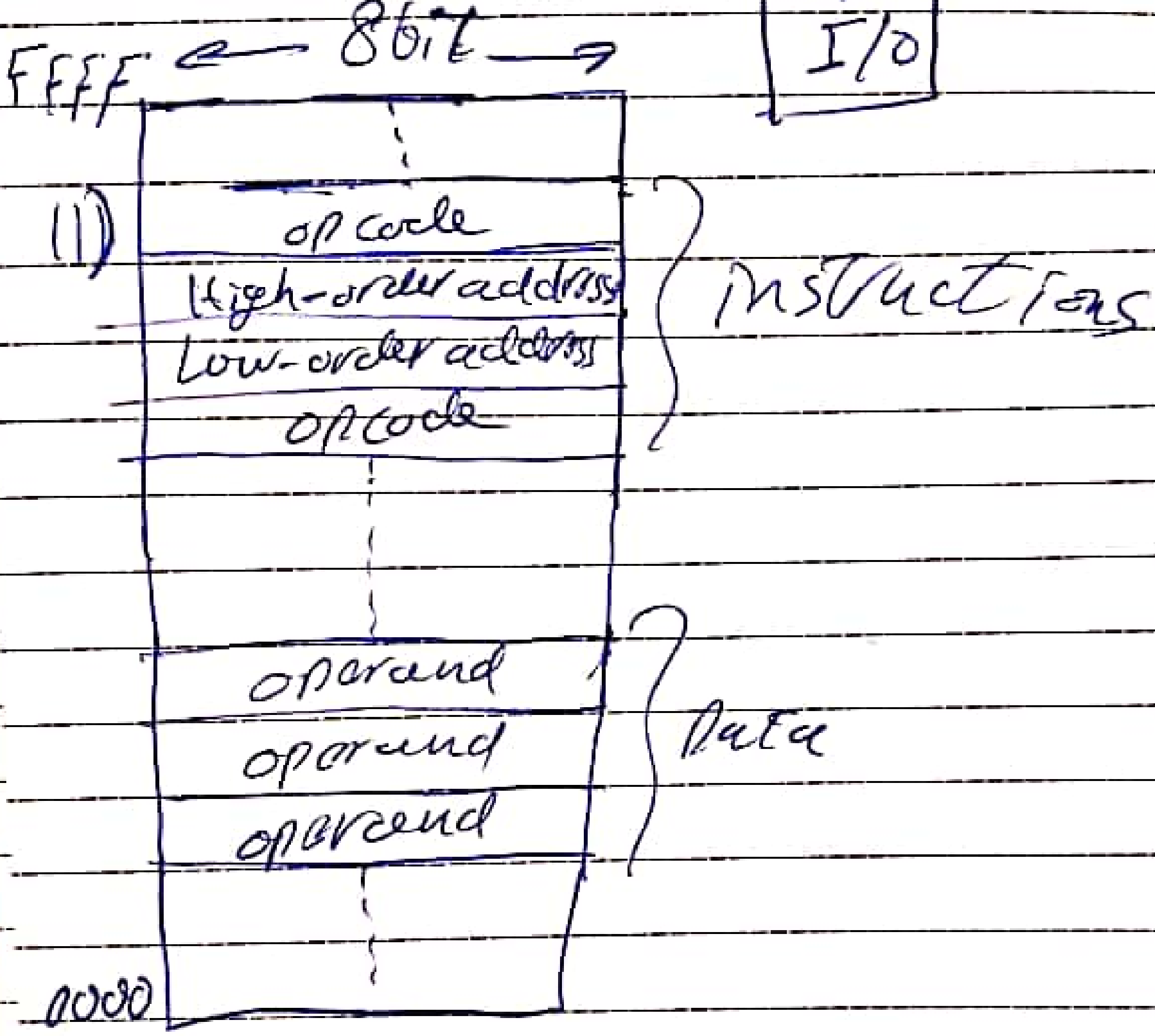
= C3000+F202 = D2202H

(f) 1B 9E 12 CC

(i)



(ii)



(iii) step 1 ⇒ fetch the instruction

① AR ← PC ② IR ← M[AR]

③ PC ← PC+3

Step 2 ⟹ decode the Inst. & fetch the operand

④ AR ← IR[15-8], Decode IR[7-0]

⑤ DR ← M[AR]

(e) SHL AX, 4

(f) BT AX, 5
or TEST AX, 0010H

2012

2 1GB of data = 2^{30} = 4000 0000 H
starting ~~AB~~ Location = 1000 0000 H

(a) 32-bit base = start of segment 1000 0000H
20-bit Limit = Last 5-hex digits of (max Length - min Length)
~~M~~ax = FFFF FFFF H assumed
Min Length = 1000 0000 H

20-bit Limit = FFFFF - 10000 = EFFFF H
+ G bit of descriptors 1 [illegible]

1 GB = 2^{30}

(b) ~~start of data~~ 1000 0000 H
then data Limit is 4000 0000 H - 1
= 3FFFFFFF H

End of data = 1000 0000 + 3FFFFFFF = 4FFFFFFH

3 Q.2 Roll

4 Q.3 Roll

5 Q.5 Roll

6 Q.6 Roll

hw 12344

2014

[I] (i) ASCII Code:-

represent alphanumeric characters in the memory.

> it is a 7-bit code, the eight bit holds parity in some system ~~for~~

example: A = 65 = 1000001   a = 97 = 1100001

BCD: binary code decimal

information is stored in packet or unpacked forms

- packed ⇒ data is stored as two digits, per byte   91 = 1001 0001
  2digits — byte

- unpacked ⇒ data is stored as one digit per byte
  ex — 91 = 00001001 00000001
  2digit (— 2bytes →

(ii) BB - 58

(iii) a) 12 / 34   b) A1 / 22   c) B1 / 00

16

2 (i) any Location in ~~mem~~ the memory system.

(ii) it is a selector that selects the descriptor from a descriptor Table It also sets privilage Level of the request and choose global or local table.

(iii) $8192 = 2^{13}$

(iv) starting = A00000 H
ending = A01000 H

3 Q.5 2011

5 (i) scan string byte
Al compared with the byte contents of the extra segment memory location addressed by DI

(ii) the D flag bit selects whether SI/DI are increment (D=0) or decrement (D=1)

(iii) an equal condition or if CX decrement to zero

* if content of AL equal content of ES:DI

(iv)

```
MOV DI, OFFSET LIST
MOV CX, 3adH
Cld
MOV AL, 66H
REPNE SCASB
```

4 MAIN PROGRAM

```
            PUSH N
            PUSH VAL
            CALL SUBROUTINE SEARCH
SUBROUTINE SEARCH
            POP R             ;The top of the stack
                      PC       content value of the PC
            POP $R_{VAL}$
            POP $R_V$
            CLEAR $R_3$       ; $R_3$ ← 0 this register
            - - - - -            is set to 1 checked
            BRANCH-IF-EQUAL NOT-Found
Found:      MOVE      , $R_3$   ; value VAL found
            $R_0$                  at position $R_3$-1
NotFound    POP $R_{TEMP}$
            RETURN SEARCH
```

2015

[1] Q3 2011

[2] Q.2 2011

[3] Q 5 2011

[4] (a) 100 H = 256 bytes

(b) the stack is cyclic so. SP will become FFFF after hitting zero.

the physical address of the addressed data is 10000+FFFF=1FFFF

(C) The difference is that the AND changes the destination operand whereas the TEST does not. TEST only affects the condition of the flag register.

2016

1 Q.2 2014

2 Q.2 2011

3 Q.4 2013

4 (a) DAA → decimal adjust after addition

DAS → decimal adjust after subtraction

(b) AAA → ascii adjust after addition
AAS → " " " subtraction
AAM → " " " multiplication
AAD → " " before division

(c) the only differance is that the logical product is lost after TEST.

(d) NOT is one's complement
NEG is two's complament

~~5~~

(e) AL is compared with the byte contents of the extra segment memory Location address by DI

5 (a) X MOV CX,1234

(~~b~~ (b) : (e) ✓

(f) X MOV AX,BX

(g) X MUL AL

(h) : (j) ✓

ral nw [illegible]

2017

1 (a) ~~the main differance between a near and a far call is the distance from the call and the type of call and return that assembles.~~

(b) ~~The near return retrieves the return address from the stack and places it into the instruction address register.~~

(c)
```
        SUMS    PROC    NEAR
                MOV     EDI, 0
                ADD     EAX, EBX
                JNC     SUMA1
                MOV     EDI, 1
       SUMS1:   ADD     EAX, ECX
                JNC     SUMS2
                MOV     EDI, 1
       SUMS2:   ADD     EAX, EDX
                JNC     SUM3
                MOV     EDI, 1
      SUMS3:
      SUMS  ENDP
```

[2] Q 2 2011 (16)

[1] (a) Near Call

memory

AFFFF	
AFFFE	00
AFFFD	03

Stack

SP before call = FFFF

SS before call = A000

IP before call = 0003

11003	
11002	procedure
11001	
11000	

10004	
10003	
10002	0E
10001	FF
10000	Call

near call

far call

memory

AFFFF	
AFFFE	01
AFFFD	00
AFFFC	00
AFFFB	05

STACK

11003	
11002	procedure
11001	
11000	

10004	11
10003	00
10002	00
10001	02
10000	call

far call

SP before call FFFF

S.S before call= A000

IP before call= 0005

*B 3 bytes Long, the first byte contain the op code and the second and the third contain the displacement

* places the content of the IP on the stack

- Like other jump bracs it can call aprocedure stored in any memory Location in the system.
- is 5-byte instructions this contain op code bytes 2 and 3 contain the new content of the IP bytes 4 and 5 contain the new content of CS
- places both IP and CS on the stack

(b) the return instruction remove a 16-bit number from the stack memory

SP → AFFFF	
AFFFE	00
AFFFD	03

Stack

11003	RET
11002	
11001	
11000	

Now RET

SP before = FFFD
SS before = A000
IP before = 1004

10004	
10003	return here
10002	0F
10001	FF
10000	CALL

[3] (a) starting Location = base address
= ~~#~~ 01000000 H
ending Location = starting + Limit
= 01000000 + 0FFFF = 0100FFFF

(b) DS = 0020H = 32 (dec)
each descriptor is 8 bytes Length
32 / 8 = 4
0020 accessed 4 global descriptor

[4] (a) The instruction doesnot specify the size of the data addressed by BX and can be corrected with BYTE PTR, WORD PTR DWORD PTR or QWORD PTR

(b) DL = 81   BH - DL = 81

S = 1   A = 0
Z = 0   P = 0
C = 0   O = 1

(c) no instruction is avilable to add to asegment register (DS).

(d)

SUB AH,AH ~~AH=0~~ AH=0

MOV AL, '6' ~~AL=36H~~ AL= 36H

ADD AL, '7' AL= 6D H 6D

AAA for AAA AL= ~~6D~~ = 0110 1101

OR AL, 30H

ال lowest nibble أكبر من 9 بالتالي هنضيف 6

نضيف 1 ع ال AH و نصفر ال ~~nibble~~

ال highest nibble ع ال AL

AH=1 0000 1101

0110

0011

AH=1H AL=3H /الناتج صحيح

OR AL,30H → AL=33H

---

5 e)

12	A1	B1
34	22	00

2018

1 (a) physical Address = SS*10 + SP

(b) Lower range => starting = SS*10

(c) upper range = ending address
SP*10+ FFFFs

2

9	← SI=9
19	← 5
17	← SI=7
11	← SI=6
16	← SI=5
18	← SI=4
10	SI=3
14	← SI=2
12	← SI=1
15	Gradas [SI]

AL	0	12	22	38	55	64
CX	5	5	4	3	2	1
SI	1	3	5	7	9	11

62

3 (a) Q.4 2019

(b) address in table = 42 X 4 = 108H

offset low = [108] = 2A

" High = [109] = 33

segment low = [10A] = ~~4A~~ 3C

" High = [10B] = ~~3C~~ 4A

address = 4A3C : ~~2A33~~ 332A = 4A3C * 10 + 332A

(4) Q.4 2016

PIXELOGIC™

SS = 100 starting Location = 10000

ending location = 10000 + FFFFs

(a) FFFF - 0100 = -- H

(b) 1000

Stack cycle

لو خلص ال full كده هيبدأ من الأول فكرة الساعة

loc starting location

final 2019

1

a Book 7

- 16 bit Microprocessor
- execute instructions in 400 ns (2.5 MIPs)
- 1M byte of memory
- small 4 or 6 byte instruction cache

b Book 146

Proc → * indicate the start of a procedure
* must be followed with NEAR or FAR
* specifies and automatically saves any register used within the procedure

EndP → indicate the end of a procedure

c

[XLAT] Book 138

- Converts the content of AL register into a number stored in a memory table
- It first adds the content of AL to BX to form a memory address within the data segment, it then copies the content of this address into AL

example → Book 138

[LEA]

Loads 16 or 32 bit register with the offset address of the data specified by the operand

example → LEA BX, [DI]

Loads the offset address specified by [DI] (content of DI) into BX

②

ⓐ

* AND DX, BX

→ register addressing Mode

* JMP JMPTAB[BX]

→ Base plus index addressing mode

* ADD DX, 15

→ Immediate addressing mode

* CMP WORDPTR [BX+DI], 10

→ Base plus index addressing mode

* MOV IVAL [DI+4], CX

→ Index plus DISP addressing mode

ⓑ

addresse = segment register *10h + offset

all register 8 bit

address = 12 bit

~~memory = 2^{8} = 4KB~~

~~example 10 segment , FF offset~~

~~address → 100~~
~~+ FF~~

~~1FF → 12 bit address~~

بالتالي

address = 8 bit *10 + 8 bit

→ 12 bit + 8 bit

→ 12 bit [illegible]

* size of the total address = 2^{12} = 4KB

* size of offset = 2^{8}

3

the body of the loop will execute 4 times (Cx = 4)

Ax = 0 + 7 + 1 = 8

Ax = 8 + 6 + 0 = 14

Ax = 14 + 5 + 0 = 19

Ax = 19 + 4 + 0 = 23 = 17 H

Cx = 0

SI = 22

ax	0	0+7+1 8	8+6 14	14+5 19	14+4 23
Cx	4	3	3	2	1
SI	0	2	4	6	8

⇒	0	7
	1	0
⇒	2	6
	3	0
⇒	4	5
	5	0
⇒	6	4
	7	0

ax = 23 = 17H Cx = 0 SI = 8

4

a) the interrupt vector table contain 256 Four byte entries. Containing CS:IP interrupt vectors for each of the 256 Possiple interrupts the table is used to locate the interrupt service routine addresses For each of those interrupts

offset segment	interrupt 0
—	interrupt 1
—	
—	
offset segment	interrupt 255

4

(b) address in table = 2×4 = 8H

offset low = [8] = 16 offset high = [9] = 05

segment low = [10] = DA segment high [11] = 09

address = 09DA : 0516
= 09DA*10 + 0516
= A2B6

5

a

real Mode
each segment is 64KB and the total memory is 1MB
therefore $2^{20} / 2^{16} = 16$ segment
protected mode
13 bit selectors, therefore $2^{13} = 8192$ descriptors
but there is a bit which selects local or global
descriptor tables, therefore 8192 + 8192 = 16384 segment

b

```
MOV CX,0
MOV AX,1
IF AX <= 3EB
 print "Enter
CALL SCAN NUM
MUL CX
ENDIF

DEFINE-SCAN_NUM
```