ELCoM
ELECTRICAL COMPUTER MECHATRONICS

تقدم لجنة ElCoM الاكاديمية

تلخيص لمختبر:

أنظمة مضمنة

Embedded system

Internal Lab (Emb)

```
#include "P16F84A.inc"
```
→ مكتبة من خلالها بقدر أستخدم اسم الـ reg بدال الـ address تاعه

```
Cblock 0X20
X1
X2
endc
```
→ من خلالها بعرف Variable بقدر أستخدمها داخل الـ Code تاعي

كيف تبلش من الـ address يلي أنا بحدده

X1 → 0X20
X2 → 0X21

address للـ Variable

```
N1 equ H'20'
```
→ هون بعرف الـ Constant اسم وقيمة باستخدام الـ equ

N1 → [20] in Hexa

```
org 0x00
```
→ هون من خلال org بحدد لأدرس معين، بحيث أي inst. بعدها رح تتنفذ من عند الـ address يلي حطيته

```
Banksel TRISA
```
→ Banksel بتغير من Bank لآخر
→ هون بنقلني لـ Bank1

```
Banksel PORTA
```
→ هون بنقلني لـ Bank0

عن طريقة كتابة اسم reg موجود في الـ Bank يلي بدي أروح عليه

STATUS reg:-

IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C

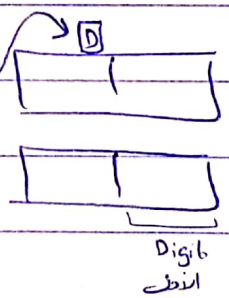
bit 7 bit 0

هاد كله عبارة عن reg الـ 8bit كل bit عبارة عن flag معين

$\overline{TO} \rightarrow 1$
$\overline{PD} \rightarrow 1$
} always

Z → Zero flag → 0 / 1 → set

C → Carry flag
DC → Digit flag



Carry عبارة عن
Digit أول

RP0 → 0 → Bank0
1 → Bank1

* destination

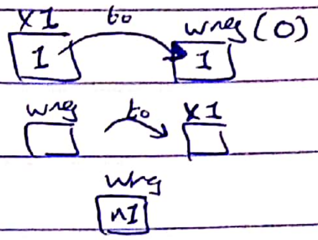
O/W → 0

I/F → 1

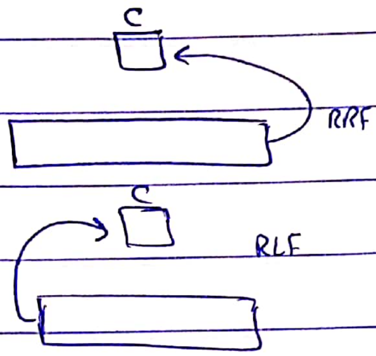
destination

ADDWF	X1 , 0	Wreg = X1 + Wreg
ADDLW	n1	Wreg = Wreg + n1 (constant)
ANDWF	X1 , 1	X1 = Wreg . X1
ANDLW	n1	Wreg = Wreg . n1 (constant)
CLRF	X1	يحذف القيمة بال reg بال
CLRW		يحذف القيمة بال wreg بال
COMF	X1 , 1	عبارة عن 1's complement يقلب ال 1 لـ 0 وال 0 لـ 1
DECF	X1 , 1	X1 = X1 - 1
INCF	X1 , 1	X1 = X1 + 1
* DECFSZ	X1 , 1 →	X1 = X1 - 1 وبعد ما تنقص بتشوف إذا 1 = Zflag يتعمل skip عن next inst
* INCFSZ	X1 , 1 →	نفس الشي بس صوب X1 = X1 + 1 وبعدها بتتأكد من Zflag
IORWF	X1 , 1	X1 = X1 + wreg
IORLW	n1	wreg = wreg + n1 (constant)
XORWF	X1 , 1	X1 = X1 ⊕ wreg
XORLW	n1	Wreg = wreg ⊕ n1

MOVF X1, 0
MOVWF X1
MOVLW n1

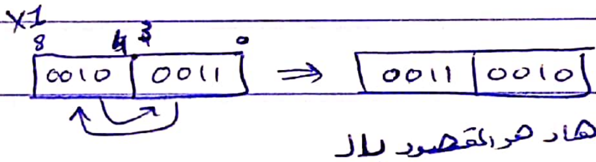


RLF X1, 1
RRF X1, 1



SUBWF X1, 1 — X1 = X1 − wreg
SUBLW n1 — wreg = n1 − wreg

SWAPF X1, 1

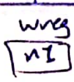


هاد هو المقصود بال swapf

NOP — No operation (بنفضل وضعها قبل الـ
عادة ما تتغير قيمة أي reg)

BTFSC X1, 0 → عبارة عن if بيتأكد إذا الـ bit المحدد يساوي 0 بيعمل skip عن next instruction
(0 = رقم الـ bit)
BTFSS X1, 1 → عكسها هاي إذا يساوي 1

GOTO Label — بينتقل لـ Label (address)
Call Label — هون بنستدعي subroutine وهي مجموعة من inst
RETURN — بننهي فيها الـ subroutine
RETLW n1 — بننهي فيها " " لكن بعطي قيمة لـ wreg



Ex:-

— R = V1 * 5

```
#include "P16F84A.inc"
    cblock 0x30
     R
     V1
     Counter
     endc
org 0x00
    movlw d'15'
    movwF Counter
    clrw

Again
    AddwF V1,0
     DECFSZ Counter,1
          goto Again
    MovwF R
    end
```

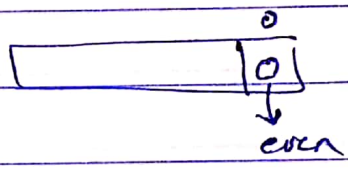
Ex:

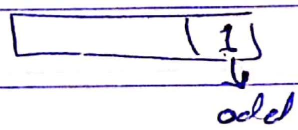
```
#include "P16F84A.inc"
    Cblock 0X20
    V1

    endc

BTFSS V1, 0
goto L1
goto L2
L1
    movlw 0
    goto finish
L2
    movlw 1
finish
    nop
    end
```

اذا فيه 1 ← BTFSS

أعمل skip عن next ins

0 ← أول bit





Ex8-

```
#include "P16F84A.inc"
  cblock 0x20
  X1
  X2
  X3
  Var
  endc
  MovF X1, 0
  SubWf X2, 0        → X2 - X1
  Movwf X3

  btfss X3, 0
    goto L1
  goto L2
L2
  Movlw b'1111 1111'
  Movwf var
  goto finish
L1
  Movlw b'0000 0000'
  Movwf var
finish
  nop
  end
```

Ex:

```
#include "P16F84A.inc"
    Cblock 0x30
        Var1
        Var2
        Var3
        endc
    Movlw 10
    Movwf Var1
    Movlw 13
    Movwf var2
    Movf Var1, 0
    Movwf Var3
    Movf Var2, 0
    Movwf var1
    Movf Var3, 0
    Movwf Var2

    nop
    end
```

- look up table

هي نوع من انواع subroutine

يتم استدعائها بـ Call namesubroutine

لكن يلي بيزها انه الها special return وهي عبارة عن ← retlw k

كيث بترجع قيمة k

لـ wreg قبل لتعمل

return

* إذا هي

- block of data that is held in the program mem
- used to retrieve values depending on the input they receive

هي ترجع value حسب الـ input يلي بتوصلها

```
MOVLW 3
CALL MylookUpTable
.
.
```

before call the lookup table W = 3

assume that this lookup table start from the location 100

address

```
100  MylookUpTable
       addwf PCL, F
101  0 nop
102  1 retlw 0x0A
103  2 retlw  b '10011110'
104  3 retlw  23
105  4 retlw  18
106  5 retlw  54
```

index = w reg

PCL store the address of the next instruction to be executed .

PCL=PCL+W

PCL= 101+3=104

before return from subroutine , w becomes 23

Index	Value
1	0A
2	9E
3	23
4	18
5	54

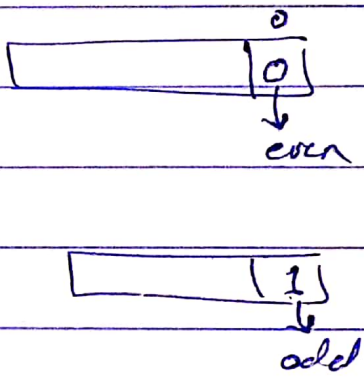
Ex:-

— R = V1 * 5

```
#include "P16F84A.inc"
    cblock 0x30
     R
     V1
     Counter
     endc
  org 0x00
   movlw d'15'
   movwF Counter
   clrw

Again
   AddwF V1,0
    DECFSZ Counter,1
          goto Again
   MovwF R
   end
```

Ex:

```
#include "P16F84A.inc"
    cblock 0x20
      V1

    endc
```



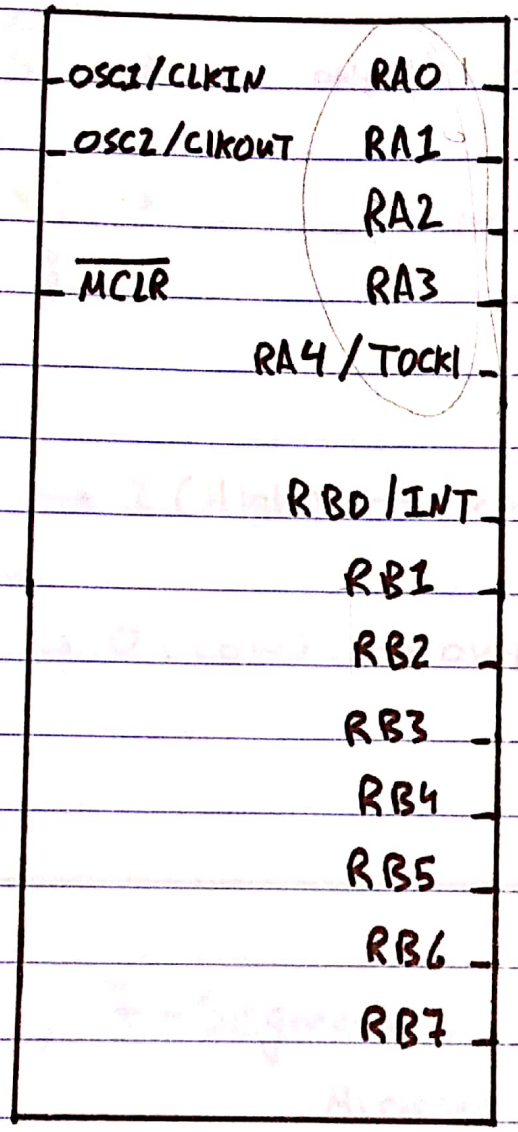

```
    BtfSS V1, 0      ← اذا في 1 اعمل skip عن next ins
                     → رقم bit
    goto L1
    goto L2
L1
    movlw 0
    goto finish
L2
    movlw 1
finish
    nop
    end
```

Ex8-

```
#include "P16F84A.inc"
    cblock 0x20
    X1
    X2
    X3
    Var
    endc
    MovF X1, 0
    Subwf X2, 0        → X2 - X1
    Movwf X3

    btfss X3, 0
        goto L1
    goto L2
L2
    Movlw b'1111 1111'
    Movwf var
    goto finish
L1
    Movlw b'0000 0000'
    Movwf var
finish
    nop
    end
```

Ex:

```
#include "P16F84A.inc"
    Cblock 0x30
     Var1
     Var2
     Var3
     endc
    Movlw 10
    Movwf Var1
    Movlw 13
    Movwf var2
    Movf Var1, 0
    Movwf Var3
    Movf Var2, 0
    Movwf var1
    Movf Var3, 0
    Movwf Var2

    nop
    end
```



portA : 4 pins

PortB : 8 pins

* بدي أقدر أتعامل مع ال pins عشان لازم two important reg

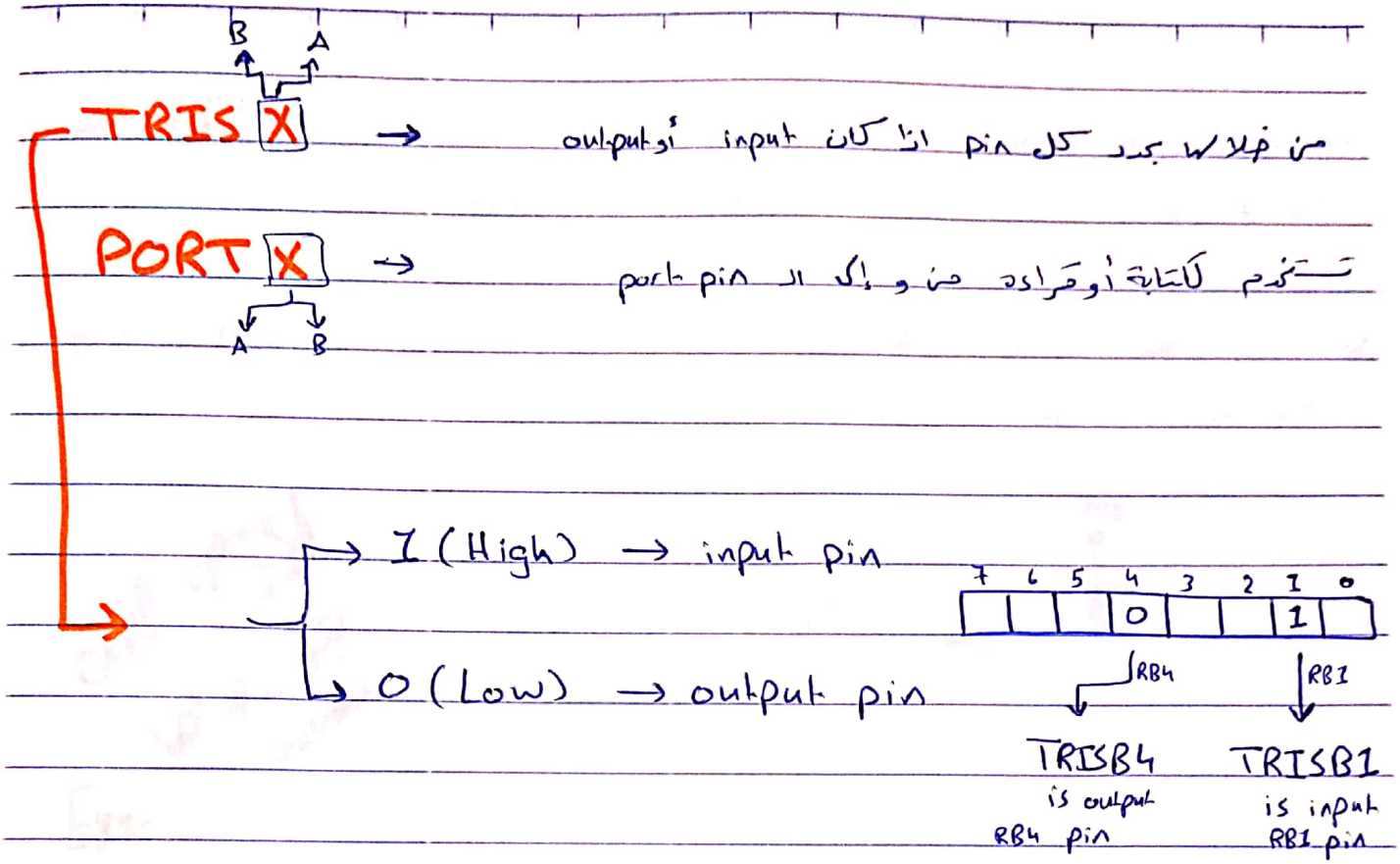
1 → in
0 → out

1. PORTA , PORTB
2. TRISA , TRISB

input

TRISA بستخدمهم مع portA

TRISB بستخدمهم مع PortB



من خلاله يحدد كل pin اذا كان input أو output

تستخدم لكتابة أو قراءة من و إلى ال port pin

+ LEDs , 7-Segment , motors , LCDs (write)

هي عبارة عن Microcontroller's output

+ switches , push buttons , sensors , keypad , LCDs (read)

هي عبارة عن Microcontroller's input

Exs-

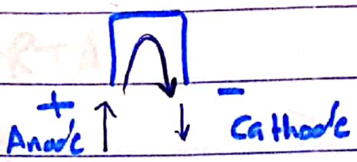
```
MOVLW 0X0F
MOVWF TRISA
```

→ 4 3 2 1 0
0 1 1 1 1
→ (RA0-RA3) is input
RA4 is output

```
MOVLW b'11001010'
MOVWF TRISB
```

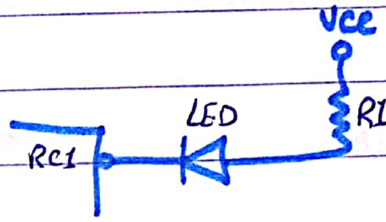
→ RB1, 3, 6, 7 is input
RB0, 2, 4, 5 is output

- LED



بحوي لا تنتقل Current
من + إلى ال -

* لازم نوصل مع ال LED ← resistance لا تتجاوز قيمتها ال (220Ω to 1KΩ)
حتى نحمي ال LED من ال burn



clrf TRISA ↓ بتخزن جواها 00000

Ex:-

```
#include "P16F84A.inc"
    Cblock 0x30
     V1
     V2
    endc

   org 0x00
   Banksel TRISA
     Movlw 0x000
     Movwf TRISA
   Banksel PORTA

Display
```

RA0,1,2,3,4 → output

- انتقلت لـ Bank1
- وضعت قيمة 0
- لكل ال pin عشان يفهم انها output
- انتقلت لـ Bank0

Display ← Label

```
Movlw 2
Movwf PORTA     →   [0|0|1|0]   (bits 3 2 1 0)
Call delay      →   استدعاء subrotiane

Movlw 9
Movwf PORTA     →   [1|0|0|1]   (bits 3 2 1 0)
Call delay      →   استدعاء subrotine

goto display    →   ترجع لـ label
```

```
Delay                          ] subrotine
    Movlw 0XFF                 ] Delay
    Movwf v1
    Movwf v2
Loop2 —
    Decfsz v1,1
      goto loop2

    Decfsz v2,1
      goto loop2
    Return

nop
end
```

• وظيفتها تعمل له Delay
الشغل ده يعتبر سريع جداً

Movf PORTA, 0

↓

عشان آخد قيمة الـ input بتاعي في TRISA

255 * 255

00000000 − 1 = 255 (Hex)

0001
0010
0100
1000

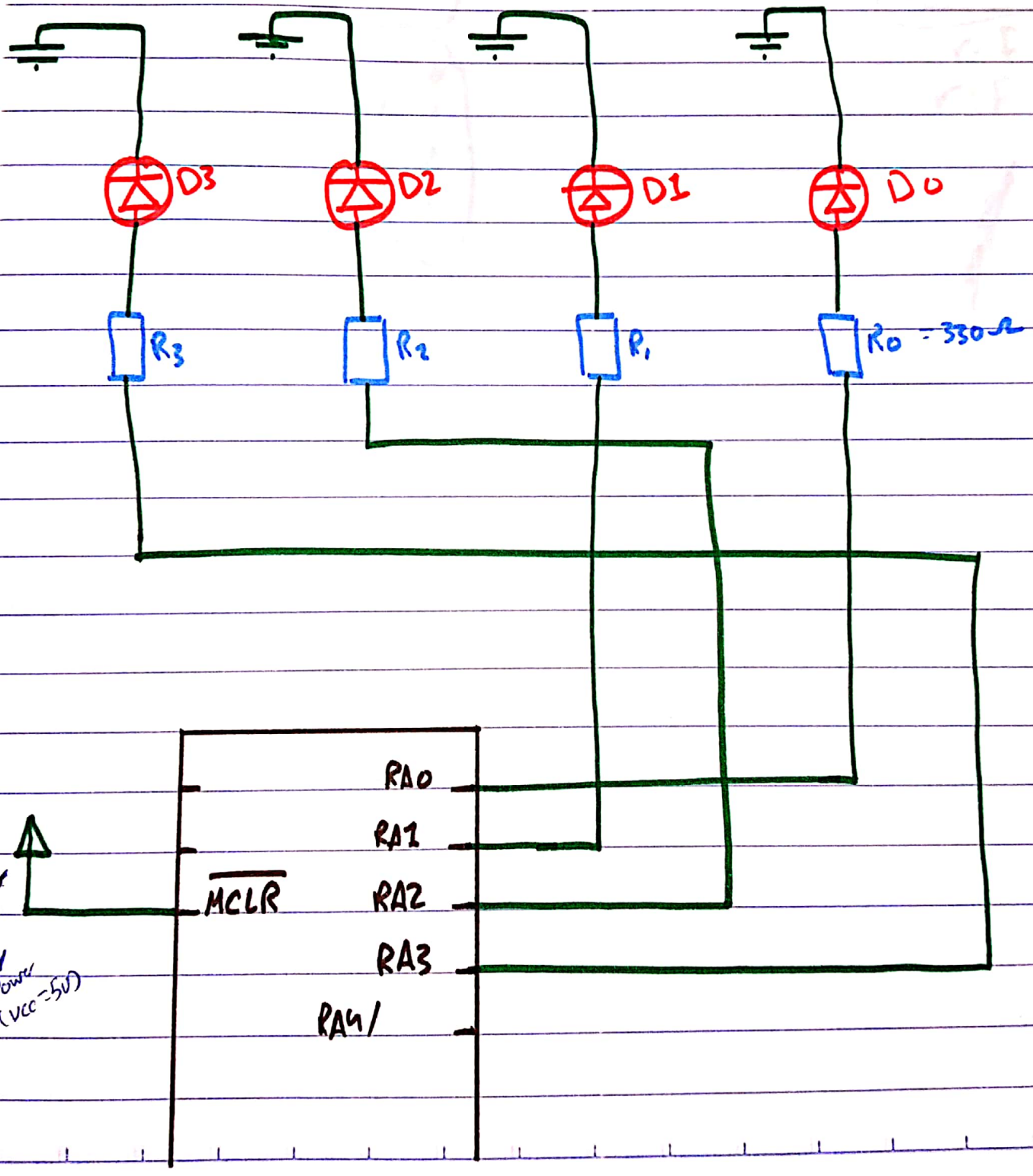
ok

Scanned with CamScanner

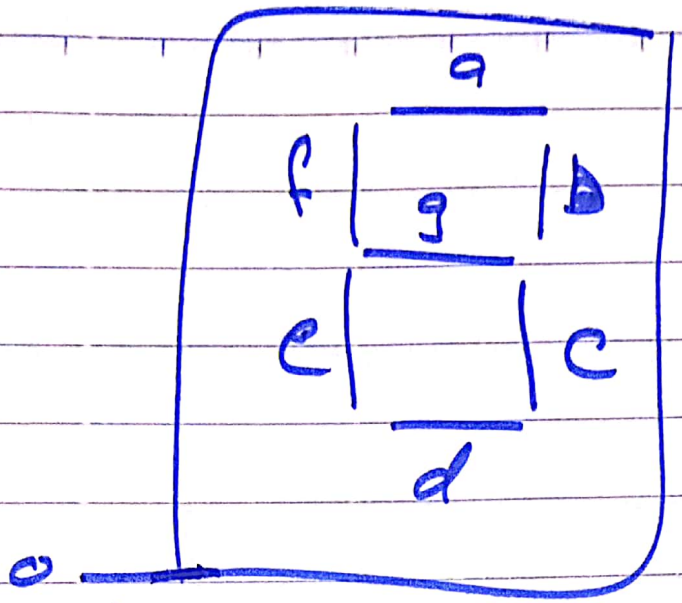
هاد التمثيل الـ ckt للمثال السابق ، لما أبعت

D3 D2 D1 D0 ← 2 لـ أربعة
ON off ON off

ON off off ON ← 9 لـ أربعة



ok



a	RB0
b	RB1
c	RB2
d	RB3
e	RB4
f	RB5
g	RB6
X	RB7

CathodE → لازم أودي 1 لـ كل عشان يضوي

abcdefg
1101101 → يضوي

=2
↓
a b g e d