Embedded Systems

No.

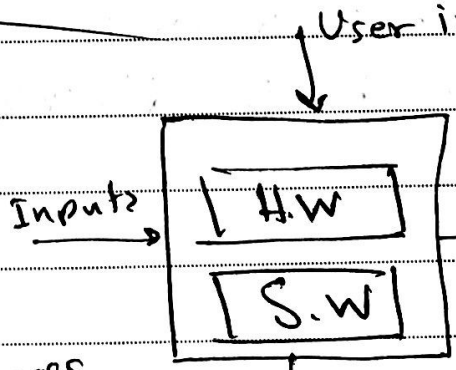
E.S. : Computer system embedded on non-computing device that performs one of few dedicated functions, usually with real time constraints.

↳ There is deadline, any action after this deadline are useless (timing & delays)

To understand any embedded sys:-

1- Inputs.
2- Outputs.
3- User interaction.
4- Link to other systems.

5- Hardware-
6- Software.



access to the code → Take backup

No access → ask for code → -ve response

black box

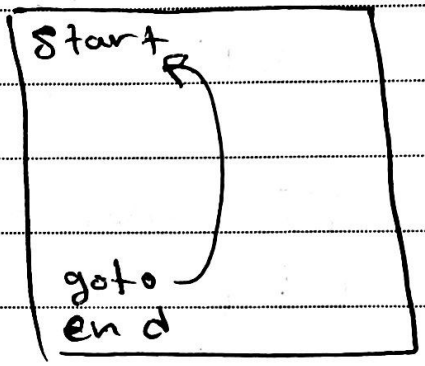
No.

Characterestics of E.S.:

1- R.C.

2- Software driven: all components in the E.S. are controlled by sw. (program)
"when to read inputs & what to do in outputs"

always running →



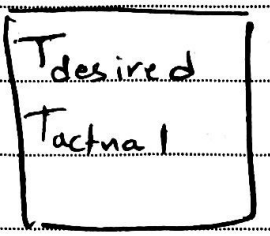
3- Reliable: almost 100% works correctly.

4- Real time.

5- Autonomous.

6- Work in diverse Environment variables.

No.



Ta > Td
turn on compressor.
else
turn off compressor.

M.C.: Computer on chip.
↳ All components of computer are existing on the M.C.
1- CPU: Control for all components. execut prog.
2- I/O: deal with external world.
3- Memory: Storage.
[1] Program: permenant.

[2] DATA: Volatile.
easier to write in terms of power and time.

4- ~~busse~~ buses: connect all components together.

No.

Code $\xrightarrow{\text{compile}}$ check syntax
generate machine code

$\xrightarrow[\text{on M.C}]{\text{down load}}$ program memory

$\xrightarrow[\text{on M.C}]{\text{run}}$ Data memory.

* On power up: The CPU starts executing the code stored at specific address called "Reset Vector".

* Program Counter: It refers to the instruction to be executed.
On power up PC= Reset Vector. (0000)

PC: holds the address of next instruction to be executed.
↑ auto increment.

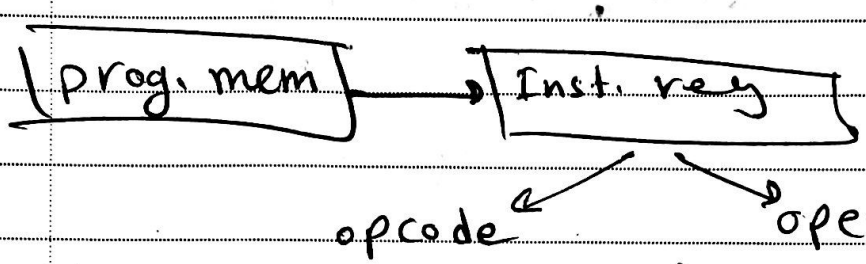
On powerup, PC= reset vector=0000
* The CPU fetches the instruction that its address is in the PC.

No.

→ * After the instruction is fetched: the CPU executes the instruction.

* While the CPU is ~~exect~~ executing the instruction, the next instruction is being fetched (PC is auto increment).

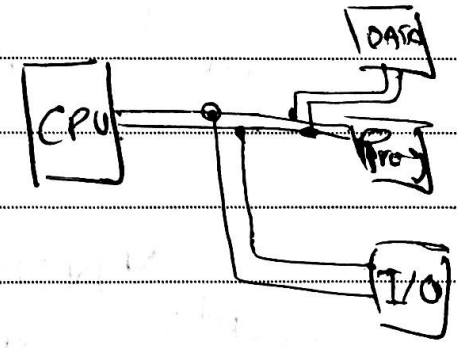
pipelining



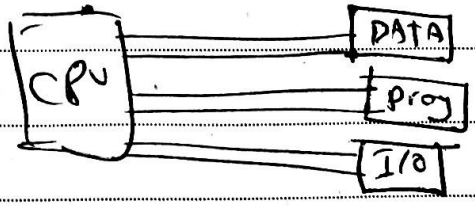
Based on bus connections:

1- Von Neuman Arch.

* fixed bus sizes



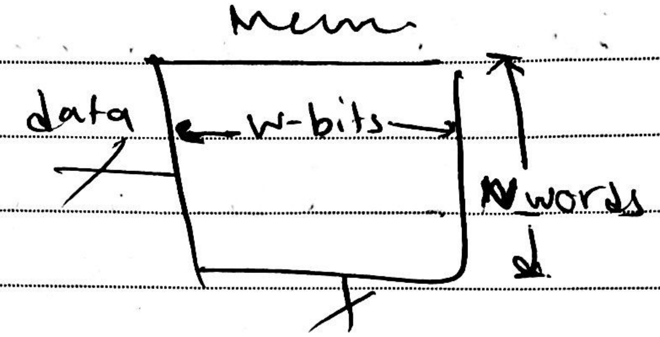
2- Harvard Arch.



* faster due to pipelining.

* variable bus sizes.

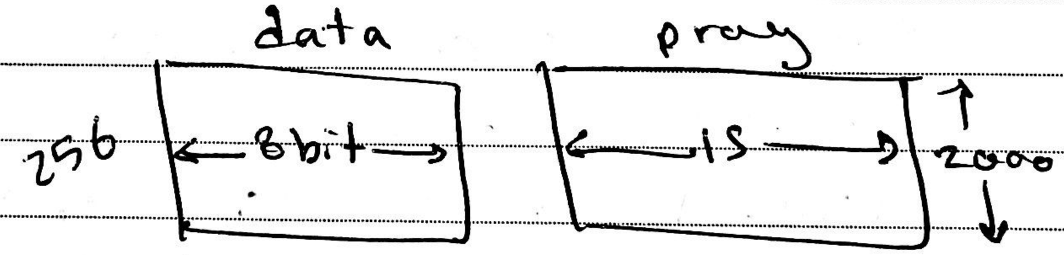
Word size (w): When you store a value, it reserves w-bits, when you read a value, it returns w-bits.



data bus $\geq W$

address bus $\geq \lceil \log_2 N \rceil$

e.g.



		VNA	Harvard
data mem	data	$\max\{8,15\}=15$	≥ 8
	add	$\max\{8,11\}=11$	$\geq \lceil \log_2 256 \rceil = 8$
prog mem	data	15	≥ 15
	add	11	$\geq \lceil \log_2 2000 \rceil = 11$

No.

* Instruction set Architecture:
List of instructions and their ~~details~~ detailed format from which you can write code (programs).

* When you write code, if the code does not adhere to the ISA; Syntax error that is detected by the compiler.

Two types for ISA:

1- Reduced instruction set computing (RISC)
- only few simple instructions.
- longer programs,
- slower compilation.
- faster execution (more effecient pipeline).

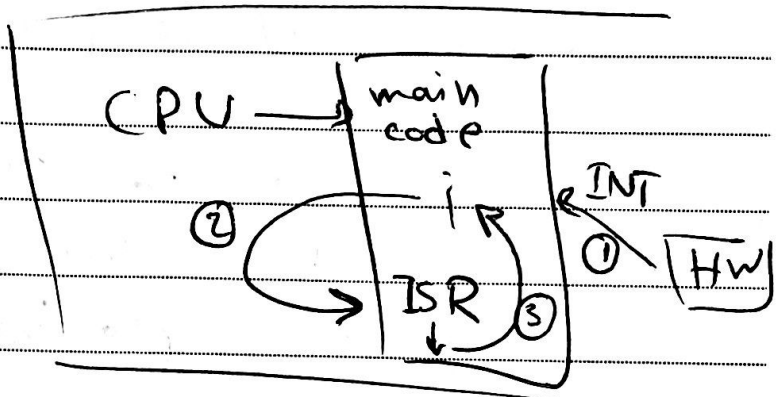
2- Complex instruction set computing ~~(CISC)~~ (CISC)
- many instructions including complex.
- Many addressing modes.

No.

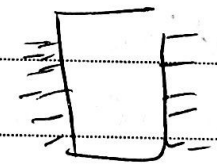
Microprocessor & Micro controllers.

P.22.

M.C.s within the same family have the same core (inst. set). However, have different peripherals & memory size.



P.23.



Dual Inline packaging (DIP)

Size is dependent on:

1- # of pins (I/Os)

2- spacing between pins

Signals & Systems.

No.

P.24

Working reg (Accumulator):
Holds the result of last executed instruction.

P.25

8-bit computers: All variables are of 8-bits length.

- Series 10, 12, 14, 16, 18, 24
- families (check the data sheet)
  - base line
  - Mid range
  - High performance.

P.27

series number → 16 F 87X ← member number in given series.

L ← low power, uses flash memory

C ← CMOS

A ← Advanced technology.

No.

- Stack: Special type of memory, you can't write or read it, it is automatically written and read.
- Volatile
- It holds the return address.

call sub1 PC=4

during PC=5

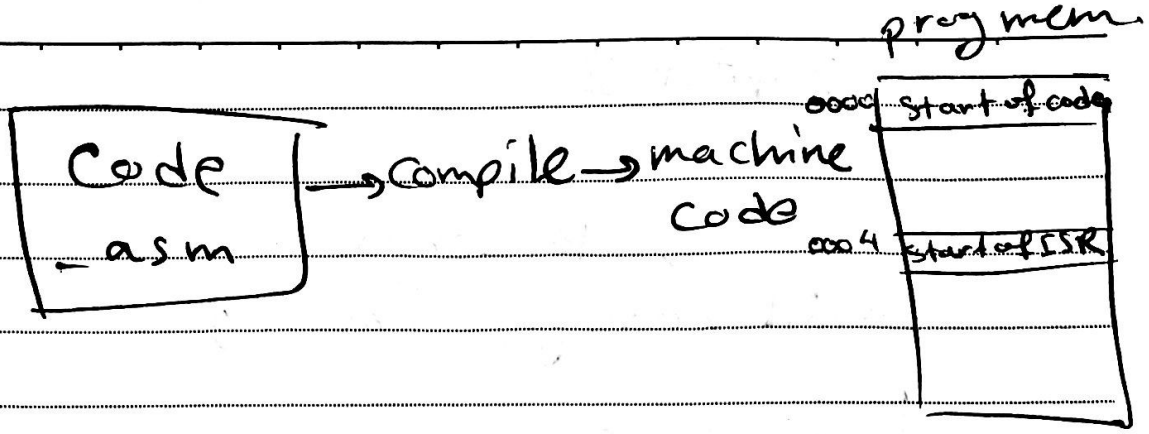
Stack

push | 5 | PC=100

return

pop PC=5

0	main
1	inst 1
2	inst 2
3	inst 3
4	call sub1
5	call sub2
	⋮
100	sub1
105	⋮
200	sub2
	⋮
209	



reset vector: default value of PC on reset 0000.

Interrupt vector: default value of PC on interrupt 0004.

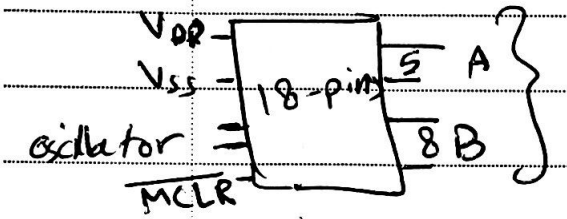
* Single interrupt: all interrupt types will make the PC = same value.
The developer has to write code to identify reason behind interrupts.

No.

Chapter 2 :

8 bit , 16 series
mid range family

↳ 16F84A → 5 pins parallel port A.
↳ 8 pins parallel port B.
↳ 8 bits timer 0.



For I/Os max 13 device.

if=0, Reset

Data memory → variables , volatile.
prog memory → instructions , permenant.
EEPROM → variables , permenant.
↑ settings.

P.6. The types of variables.

1- Value is in the instruction itself (Literal)
eg. add 5

2- The value is in data memory
add value in address 5
↳ address in instruction (direct) 7-bit
↳ address in FSR (indirect) 8-bit.
eg. add value where its address in FSR

No.

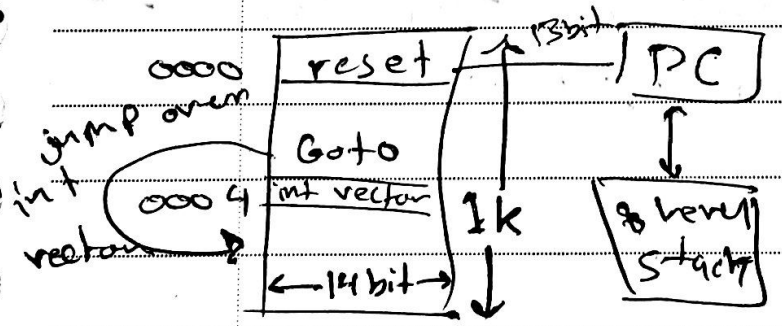
3 addressing modes :-

1- literal (value in inst)
2- Direct (value in mem., address in inst.)
3- Indirect (value in mem, address in FSR)

* Result will be stored either in w-reg or in Data memory.

* Status Register :- state of last executed instruction.

* Watch Dog timer :- It resets µ.C. on crash.



Configuration ~~code~~ word : a word stored in prog memory, it holds settings information about the prog :

① FOSC1 : FOSC0 : select type of osc. to be connected to PIC.

No.

→ RC Osc, High speed, crystal, low power its own frequency.

(2) WDTE : WDT enable : activates WDT.

(3) PWRTE : It keeps μ.c Reset mode for a while on power up.

(4) code protection :
if 0 → code protected (encrypted)

* Config word : Once written, can't be modified except by redownloading the code

No.

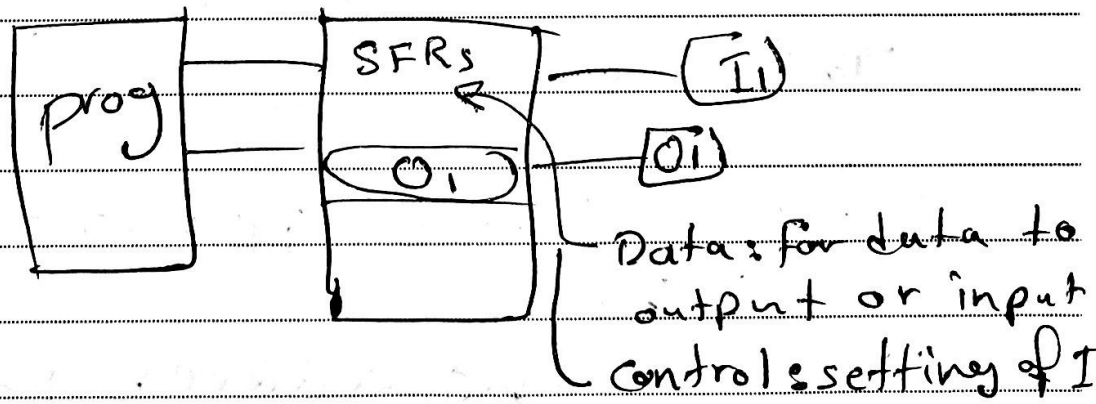
P.11:

Data memory:

↳ Vertical: Special function registers (SFRs) & General purpose registers.

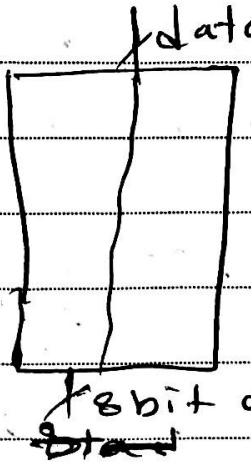
~~Spec~~ SFR: Each of these registers has its own specific function, mainly to deal with I/O.

* Memory mapped I/O: each I/O device has part of the memory mapped to it.



No.

↳ Horizontal: 2 banks (b0 & b1) ~~each~~
→ 160 words (80 word each)



$\geq \lceil \log_2 160 \rceil$
$= 8$ bits

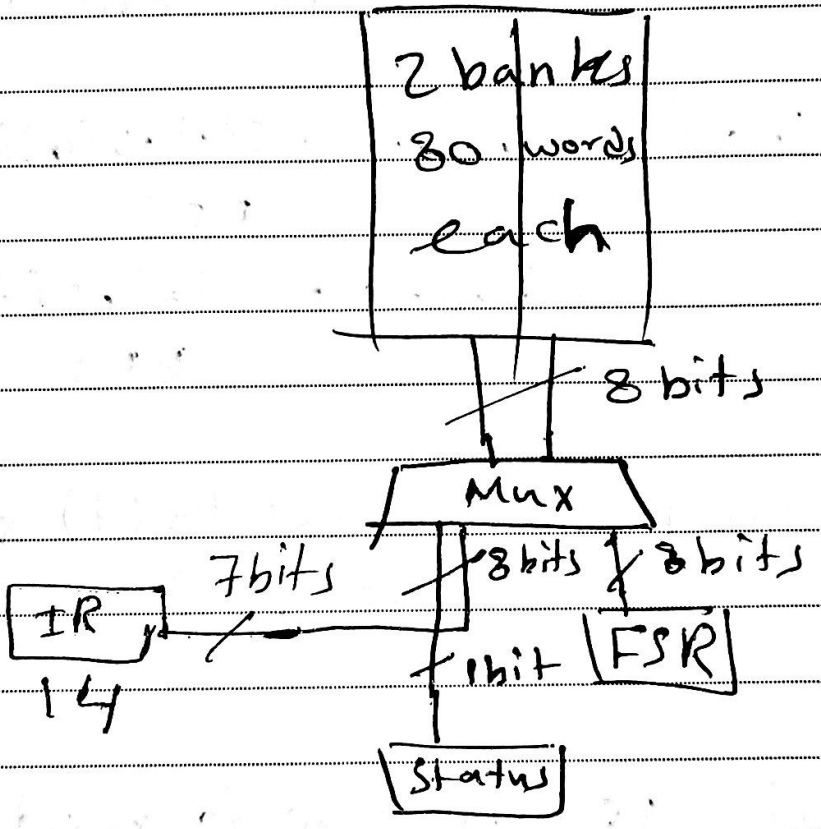
To address any word inside a given bank $\lceil \log_2 80 \rceil = 7$ bit

To access any word in the ~~data~~ memory
1) select bank (if not selected) (1 bit)
2) select the address inside the bank → 7 bit.

Dividing memory into 2 banks saves one bit out of the address
8 bits → 7 bits

No.

If the instruction contains an address of variables → the address is 7 bits & the 8th bit comes from the status register.



* You have to switch to the other bank if you want to access a register on the other bank.

No.

* Bank selection is dependent on the addressing mode.

1) Literal : No data memory access
→ no bank selection

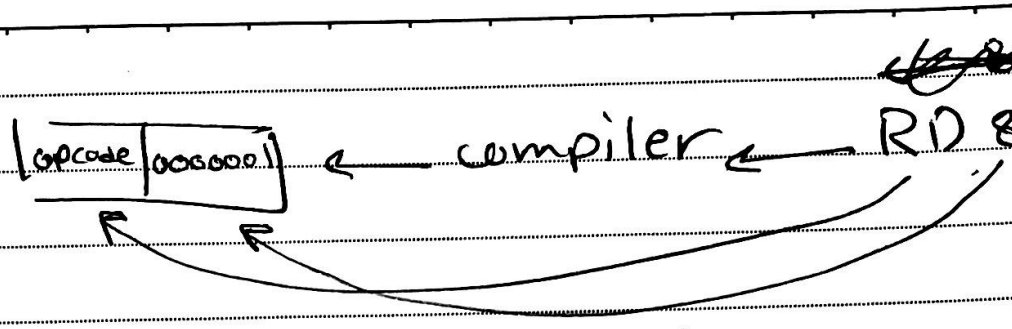
2) Direct : 4 banks : status(6) & status(5)
RP1 RP0
2 banks : status(5)
RP0

3) Indirect : 4 banks : status(7) & FSR(7)
2 banks : FSR7
↗ Fileselect register.

e.g. write pseudo code to read the addresses : 81, 5F, 7B, 92, A1
assume there is an instruction called "RD ad" that read data memory from address = ad.

No.

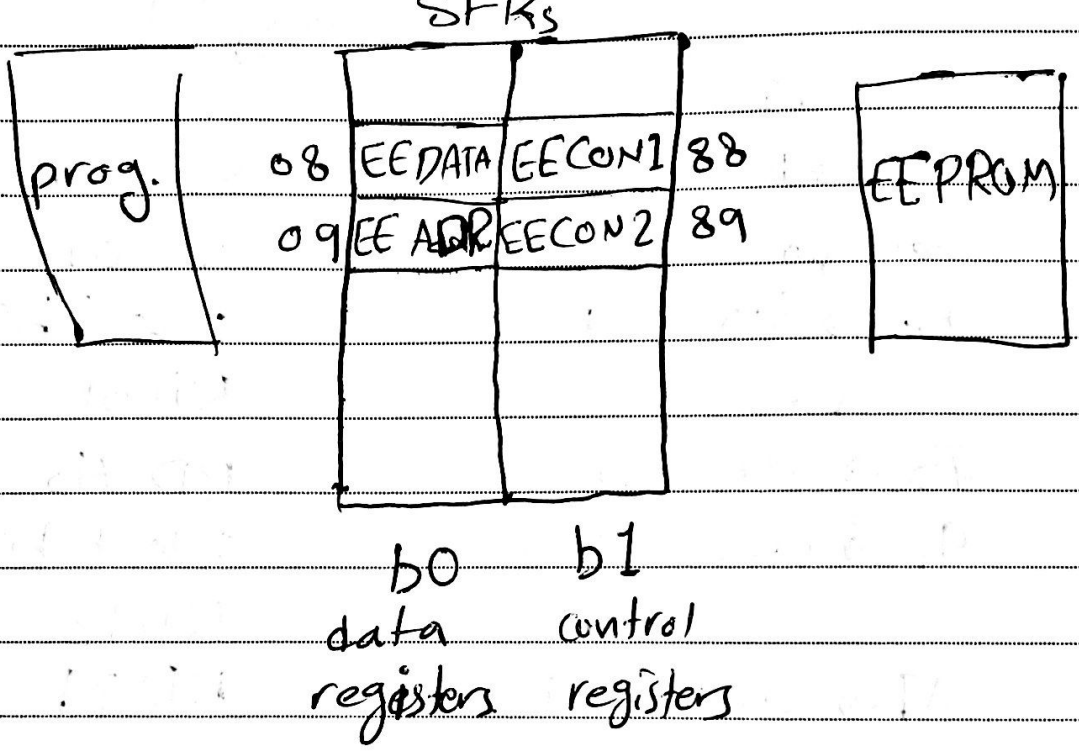
Soln →



Wrong!!

81 → 1000 0001 (b1)	Select b1 (status(5)=1)
~~5F~~ (b0)	RD 01
5F → 0101 1111 (b0)	select b0 status 5=0
	RD 5F
7B → 0111 1011 (b0)	RD 7B
92 → 1001 0010 (b1)	select b1
	RD 12
A1 → 1010 0001	RD 21

Data : Variables, Volatile.
prog : code, permenant.
EEPROM : Variables, permenant.



** To read EEPROM :
1) Write the address you want to read in ~~EEAddr~~ EE ADR. register.
2) select b1.

No.

3) Set RD bit in EECON1, when RD = 1, start reading EEPROM from address stored in EEADR to EEDATA.

4) Select b0.

5) You can read the copied word from EEDATA

Note: RD : set by sw cleared by HW. When reading is over, RD=0.

** To write to the EEPROM:

1- Write the value in EEDATA.
Bank selection 2- Write the address in EEADR.
3- Set WREN bit in EECON1. (It allows writing to EEPROM)
4- Write value 55H in EECON2.
5- write value AAH in EECON2.
6- Set WR bit in EECON1. (Start writing to EEPROM the value stored in EEDATA to the address stored in EEADR.

No.

→ Step 4 & 5 : To make sure that writing is intentional.

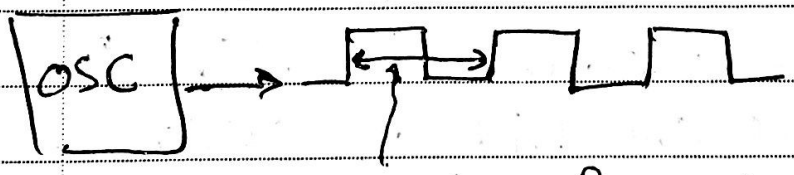
7- when writing is over EEIF = 1

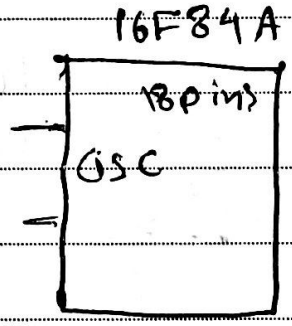
8- If writing caused an error :
WRERR = 1

9- bank selection if Repeated from 1.

Note: WR : set by sw, cleared by HW.
when writing is over ~~RD bit~~
WR bit = 0

P-20 Timing :

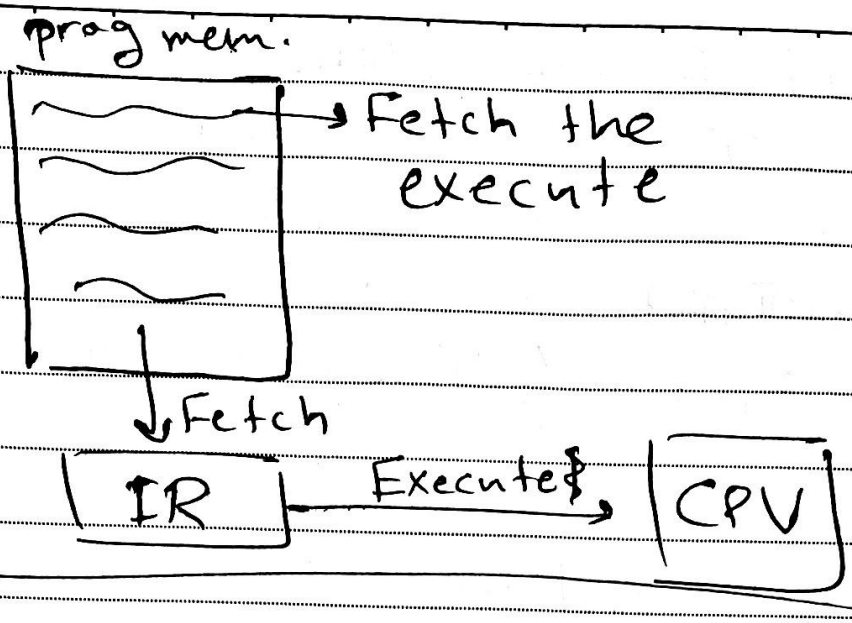




Tosc : cycle of clock that is generated by the osc.

* $F_{osc} = \frac{1}{T_{osc}}$ Frequency.

No.



E.g.

μ.c1	μ.c2
100 MHz	50 MHz
20 cycles Fetch	5 cycles Fetch
20 cycles execute	5 cycles execute.

μ.c. 2 is better although it has lower frequency.

* Instruction cycle : The time that is enough to fetch or execute single instruction.

$T_{inst} = X * T_{osc}$.

In PIC M.C : $T_{inst} = 4\ T_{osc}$.

No.

e.g. If F_{osc} = 1 MHz.

→ $T_{osc} = \frac{1}{1\,MHz} = 1\,\mu s$.

→ $T_{inst} = 4\,T_{osc} = 4 * 1\,\mu s = 4\,\mu s$.

e.g. F_{osc} = 4 MHz

→ $T_{osc} = \frac{1}{4\,MHz} = 0.25\,\mu s$.

→ $T_{inst} = 4\,T_{osc} = 1\,\mu s$.

e.g. Assume there is a prog composed of 10 instructions, how long does it need to be fully executed if F_{osc} = 4 MHz ?

Sol. F_{osc} = 4 MHz.

→ T_{osc} = 0.25 μs.

→ $T_{inst} = 1\,\mu s$.

each instruction to be fully executed it needs 2 T_{inst} (fetch & execute)

1 μs * 2 = 2 μs

No.

→ For the program to be fully executed, it needs:
10 ~~instructions~~ inst * 2 ns = 20 ns.
assuming there is no pipelining.

* ~~while~~ With pipelining, time goes to ~~approximetally~~ approximatelly half (10 ns)

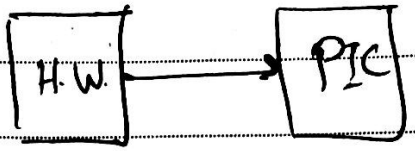
↳ some times, the pipeline fails because the fetched instruction is not the one that will be executed next. (e.g. call, return), ~~cost: 1 additio~~
cost: 1 additional Tinst.

* All instructions to be fully executed (fetch & execute) need ~~Tinst.~~ 1 Tinst. exept the instructions that make pipeline failure, they need 2 Tinst.

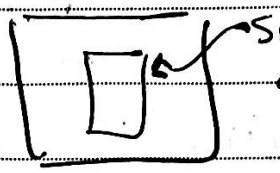
Power up and reset :

P22 To keep μC in reset mode on power up :

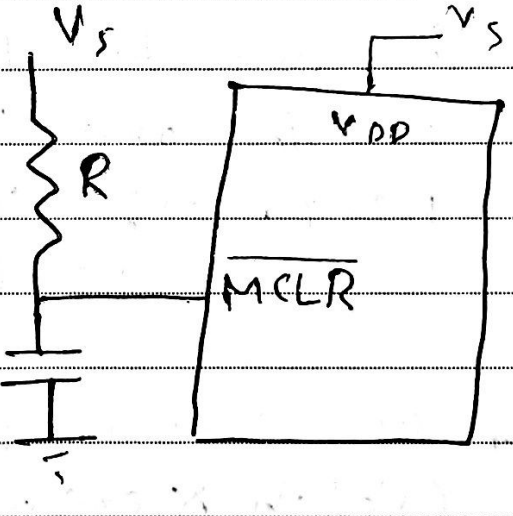
1- external



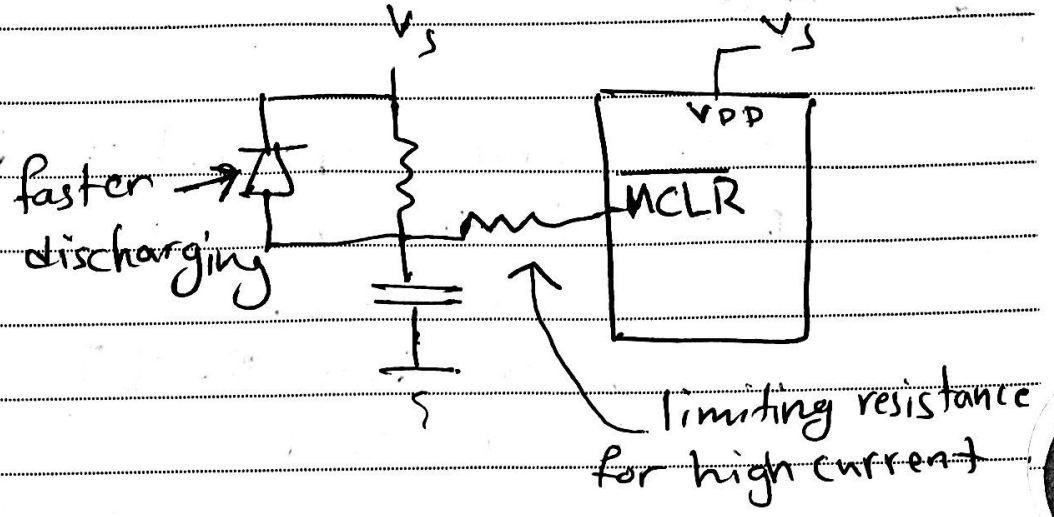
2- Internal



External :

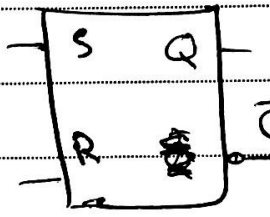


On power up, the PIC will stay in reset mode, capacitor gets charged, after that, it exits from reset mode, the time is related & to $T = R * C$



Internal:

p.23



S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	
0	1	0	1	exit from reset
1	0	1	0	reset
1	1	Unstable		

When does reset happen?

if S = 1

when does S = 1?

if a $\overline{MCLR} = 0$

b WDT detects crash

WDT won't reset PIC in sleep mode.

c power up.

No.

After power up, when does the p.c. exit from reset mode?

when R=1.

when does R=1?

if a S=0

power up timer

b if enable PWRT is enabled, we wait 1024 cycles of internal RC osc (72 ms)

c After the first counter finished counting, the second counter needs 1024 Tosc to output 1.

(if enable PWRT is enabled)

T_{abc} = [illegible] + 1024 cycles of internal RC osc (72 ms)

+1024 Tosc

if enable ost

[illegible] is enabled

No.

* Enable PWRT is in the configuration word, enable OST by default is enabled for osc types XT, HS, LP and disabled for osc type RC.

Note: If V_{DD} rise is long, and interal delay isn't enough, then use internal & external delay, and when the application is sensitive.
(internal delay < V_{DD} rise time)

*** Chapter 4

High level langunage: May be not efficient in terms of memory usage and execution time.

P.6. Any instruction with 2 var :

2 versions

→ W , Literal
→ W , Data memory.

Result is stored either in W-Reg (d=0) or in data memory (d=1)

P.7 Catigories of instructions :

1) Byte oriented file register :

* file register ≡ Data memory word.

Works on a whole byte.

needs : f (7 bits) address of file reg. / d (1 bit)

2) bit oriented file register :

works on a single bit.

needs : f (7 bits) b (3 bits) bit location

3) Literal : The value is in the instruction.

needs : k (8 bits) Add (22) 8 bits value.

↑ Literal value.

No.

→ 4) Control: Change path of execution.

needs: call K (11 bits).
goto

~~P.13 e.g. ADDWF 22, 0~~
~~ADDWF 23, 1~~
~~ADDW~~
~~ADDLW 24~~

P.13 e.g. ① ADDWF 22, 0 [W] ← [W] + [22]
② ADDWF 23, 1 [23] ← [W] + [23]
③ ADDLW, 24 [W] ← K + [W]

① 2 + 1 = 3 [W] = 2 ~~3~~ 27
② 3 + 2 = 5 [22] = 1
③ 24 + 3 = 27 [23] = ~~5~~ 2 5
[24] = 3

Subtraction: We always subtract the working register.

No.

→ SUBLW K [W] ← K - [W]

SUBWF f,d [W] ←(d=0) [f] - [W]
[f] ←(d=1)

A - B = A + 2's comp B.

2's comp & fix up to first one (1) then complement rest bits.

A = 7 = 0000 0111 —2's→ 1111 1001
B = 5 0000 0101 —2's→ 1111 1011

```
          C=1 1111 1111
A - B =   0000 0111
        + 1111 1011
        ---------------
          0000 0010
```

if C=1 ; answer is positive.

```
          C=0 0000 0001
B - A =   0000 0101
        + 1111 1001
        ---------------
          1111 1110
```

if C=0 ; aswer is negative

FIVE APPLE

No.

e.g. SUBLW 7 [W] = 2 5 0

[W] = 7 - [W] [22] = 4

= 7 - 2 [23] = 5

= 5

SUBWF 23, 0

[W] = [23] - [W]

= 5 - 5

= 0 C = 1

C is opposite of sign.

---

e.g.

INCF 15, 1 [5] = [15] + 1 [W] = 4 1 FC

DECF 16, 0 [W] = [16] - 1

COMF 17, 0 [W] = comp [17] [15] = 1 2

[16] = 2

[17] = 3

[17] = 3 = 00000011

comp = 1111 1100 = FC

No.

P.14 Logical instructions:

$A \cdot 0 = 0$	$A \cdot 1 = 1$	(AND)
$A + 0 = A$	$A + 1 = 1$	(OR)
$A \oplus 0 = A$	$A \oplus 1 = \bar{A}$	(XOR)

* We need them for bit masking: you want to set, clear or complement some bits and keep the others.

set: IOR clear: AND complement: XOR.

e.g. Write a code that complement the least significant 4 bits of wreg and keeps the others.

Sol. XORLW 0F

e.g. One instruction to set the most significant 4 bits of wreg.

Sol. IORLW F0

e.g. Write one instruction to set the ~~working~~ even bits in the working register: 01010101

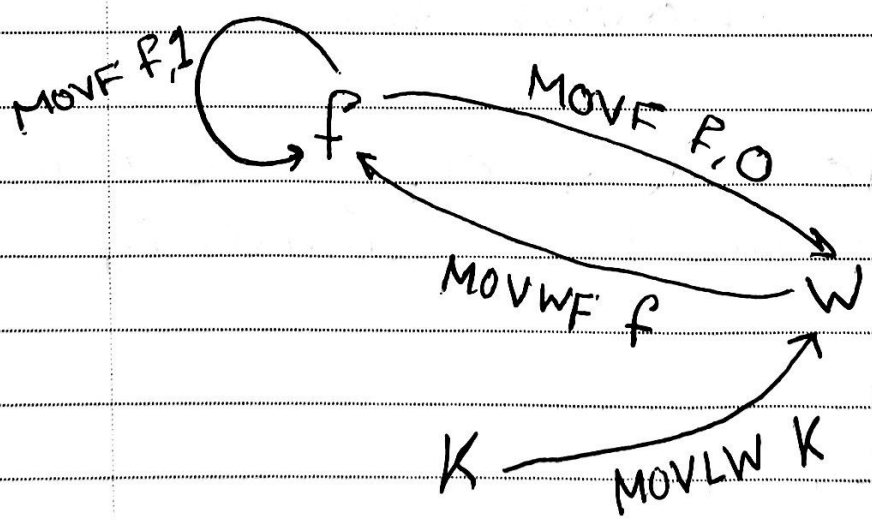
~~ANDLW 55~~ IORLW 55

e.g. Write one instruction to ~~clear~~ clear the most sig. 4 bits in the working register.

ANDLW 0F

~~e.g. set least sig. 4 bits of address 22~~

P.15 Data movement inst.:



No.

* To initialize any data memory location:

1) Put the value in W-Reg (MOVLW)

2) Move the value to data mem (MOVWF)

e.g. Write code to store the value 7 in address 17.

Sol. MOVLW 7
MOVWF 17

* To copy any value from data memory location to another:

1) Copy value to W-Reg (MOVF f_1, 0)

2) Copy W-Reg to new location (MOVWF f_2)

e.g. Write code to copy the value in address 16 to address 17.

Sol. MOVF 16, 0
MOVWF 17

No.

* MOVF f, 1

To check if the value in address f is equal to zero.

* SWAPF f,d

$a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 \xrightarrow{\text{SWAPF}} a_3 a_2 a_1 a_0 a_7 a_6 a_5 a_4$

SWAPF 11, 0

11

1F

W=F1

Ex1. Write code to initialize the data memory with value:

[16] ← 18

[17] ← 19

Ex2. Write code to do the following

[18] = [19] - [17]

No.

P.16 * Control:

	stack $\xleftarrow{\text{push}}$ pc	You
Call k	PC $\leftarrow$ k	can return
GOTO k	pc $\leftarrow$ k	You can't return
RETURN	PC $\xleftarrow{\text{pop}}$ stack	
RETLW k	(Subroutines)	
RETFIE	(Interrupt)	

P.16 — Conditional brach: GOTO if condition is true.

The next instruction will be skipped if the condition is true. Otherwise, the next Inst$ is execnted

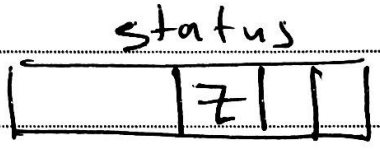
No.

Skip if condition

- BTFSS f,b if bit b in f is 1
- BTFSZ f,b if bit b in f is 0
- INCFSZ f,d if after inc. value in f, result is zero
- DECFSZ f,d if after dec. value in f, result is zero.

On skip → 2 T_{inst}
No skip → 1 T_{inst}

e.g. Write code to increment the value in address 17 if it equals zero.



Sol.
```
MOVF 17, 1
BTFSC STATUS, Z
INCF 17, 1
```

~~e.g. Wri~~

e.g. Write code to do the following

for(int i=17; i>0; i--)

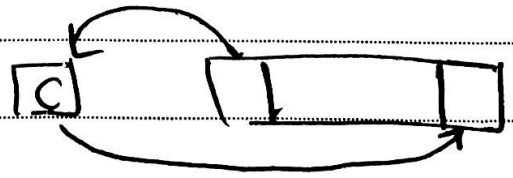
Sol.
```
    MOVLW  17
    MOVWF  22
Loop DECFSZ 22,1
    GOTO  Loop
```

P.17
```
BSF  STATUS,5   (Bank1)
BCF  STATUS,5   (Bank0)
```

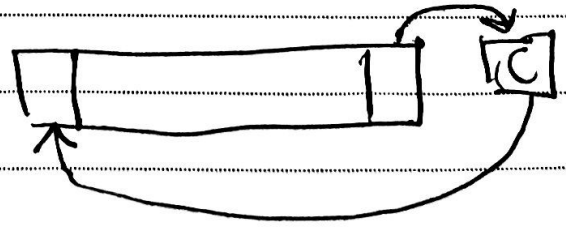
RLF → *2 if C=0
↳ *2 +1 if C=1

RRF → Integer division if C=0

RLF



RRF



No.

P.22. Label Instruction operands comment
mnemonic

- Label: *case ~~sesid~~ sensitive
  *At the begining of the line.
  *Once defined, it can be used as operand and its value = the value of first instruction that comes after it.
  *Label can be in a stand alone line.

- Comment: after the ;.

Note: If the Hex value starts with a letter (A-F) it should be ~~preceeded~~ preceded by either 0x or H.

P.23 Assembler directive: It gives the assembler some information ~~et~~ at compilation time, they they are discarded
, asm → assembler → machine code.
(They don't have any job at execution time)

FIVE APPLE

No.

1) #include

It opens a file, and uses anything in it.

e.g. #include PIC16F84A.INC

2) ORG : Tells the assembler that after converting the next inst. into binary, it should be stored at specific address.

e.g. ORG 0005

inst 1 ⟶ 05

inst 2 ⟶ 06

inst 3 ⟶ 07

3) equ : It ~~defines~~ defines a constant.

e.g. ~~equ~~ var1 EQU 5

STATUS EQU 3

RP0 EQU 5

- Var1 EQU 15

MOVLW Var1 ← Literal 8-bit
MOVF Var1, 0
↑ address 7-bit

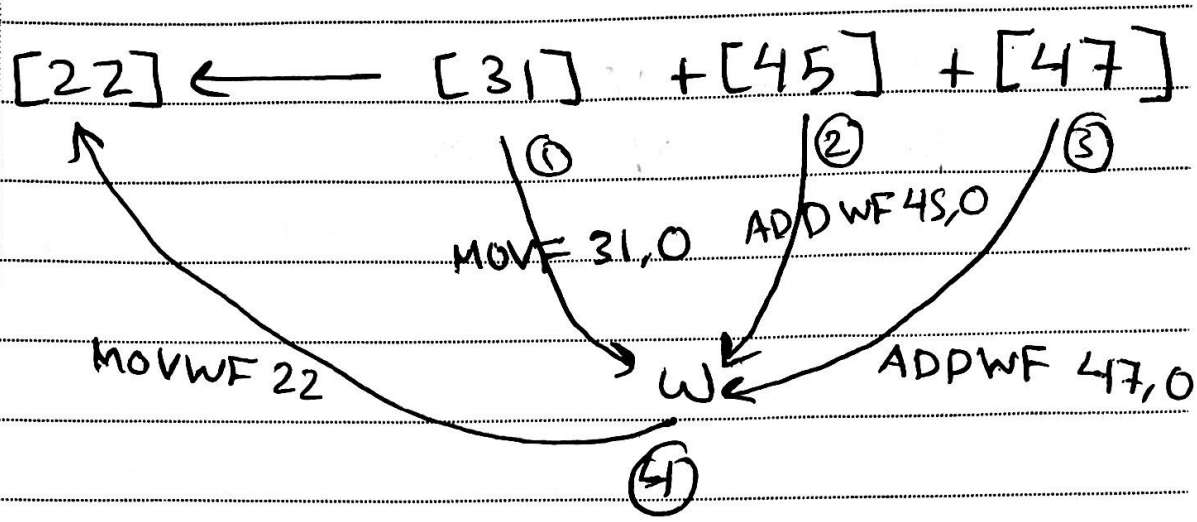
4) cblock 20

var1		Var1 EQU 20
var2	=	Var2 EQU 21
var3		Var3 EQU 22
Var4		Var4 EQU 23

endc

5) end (Tells the assembler not to compile anything beyond this point)

P.24 Sample program



Size of program in bits =

$$8 \text{ inst} * \frac{14 \text{ bit}}{\text{inst}} = 128 \text{ bits}$$

$$= 16 \text{ byte}.$$

Q. What is the binary representation of the instruction GOTO DONE?

Note: Assume that we omitted the instruction "DONE GOTO DONE", how long will the program take to be fully executed.

if $F_{OSC} = 4MHz$

→ $T_{inst} = \frac{4}{F_{osc}} = \frac{4}{4MHz} = 1\mu s$

$7\ T_{ins} = 7\ \mu s$

P.7. Conditional branch: goto if condition.

Skip: skip next inst. if condition.

e.g. if ([22] > 7)
[22] = [22] * 2
else
[22] = [22] - 9

Sol.
```
    MOVLW 7
    SUBWF 22,0 ; [22]-7
    BTFS  STATUS,C      [22]>7 → C=1
    GOTO  MUL
    GOTO  SUB
MUL BCF  STATUS,C ; *2 when C=0
    RLF  22,1
    GOTO Next
SUB MOVLW 9
    SUBWF 22,1
Next
```

No.

Sol2.
```
    MOVLW 7
    SUBWF 22,0
    BTFSC STATUS, C
    GOTO MUL
    MOVLW 9
    SUBWF 22, 1
    GOTO Next
MUL BCF STATUS, C
    RLF 22, 1
Next
```

e.g. if ([22] is even)
add 7 to addresses [21] & [22]
the values in
else
sub 7 from the values in addresses [21] & [22]

Sol.
```
    BTFSS 22, 0
    GOTO ADD
    GOTO SUB
ADD MOVLW 7
    ADDWF 21, 1
    ADDWF 22, 1
```

No.

~~SUB~~

→ GOTO NEXT

SUB MOVLW 7
SUBWF 21, 1
SUBWF 22, 1

NEXT

e.g. For (i=19 ; i>0 ; i--)
code block 1

sol. MOVLW 19
MOVWF 25

LOOP code block 1
DECFSZ 25, 1
GOTO LOOP

e.g. for (i=3 ; i<25 ; i++)
add 2 to [17]

Sol. Counter EQU 22
MOVLW 3
MOVWF Counter

Loop MOVLW 2
ADDWF 17, 1

No.

```
INCF  Counter, 1
MOVLW 0'26'
SUBWF Counter, 0
BTFSS STATUS, Z
GOTO  Loop
```

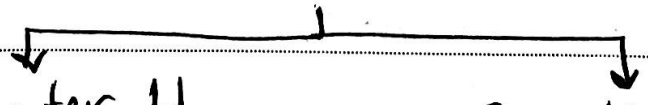
e.g.
```
for (i=7; i<29; i++)
    if ([22]>9)
        [22]=[22]*8
    else
        if([22]>4)
            [22]=[22]+7;
        else
            [22]=[22]*7
```

~~B8 F6~~

* 16 bit counters :

8 bit counter 256 iteration max.

16 bit counters



No.

Note: By default, all interrupts are disabled → No need for ORG 0004 except if you enabled interrupt.

e.g. 16 bit counter.

```
CountH   EQU   21
CountL   EQU   22

         ORG   0000
Loop     MOVF  CountL,1
         BTFSS STATUS,Z
         GOTO  DECCountL
         MOVF  CountH,1
         BTFSC STATUS,Z
         GOTO  DONE
         DECF  CountH,1
DECCountL DECF CountL,1
         GOTO  Loop
DONE     GOTO  DONE
         end
```

No.

Subroutine : code (function in c++)

starts with Label

ends with Return → PC ← pop stack

(2 Tinst) RetLW k → W ← k

↳ PC ← pop stack

and executed when called by

"call k" (2 Tinst)

↳ stack ← PC

↳ PC ← k

P.17

```
multiply  CLRW
          MOVF  31, 1
          BTFSC STATUS, Z
          RETURN
Repeat    ADDWF 30, 0
          DECFSZ 31, 1
          GOTO  Repeat
          RETURN
```

* Delay

→ SW delay : ~~None~~ None useful code, however, when this code is executed, it consumes time

→ HW delay : Timers (Asynchronous) Interrupt.

e.g. F_{osc} = 4 kHz ⟶ T_{inst} = 1 ms

```
Loop  inst 1 ; Turn on LED.
      500 NOP instructions.
      inst 2 ; Turn off LED
      GOTO Loop.
```

500 NOP.

No.

```
        MOVLW X
        MOVWF Counter
Loop    NOP                    Max 256 iter.
        DECFSZ counter, 1
        GOTO Loop
```

```
        MOVLW X
        MOVWF CounterH
        MOVLX Y
        MOVWF CounterL
Loop    NOP
        DECFSZ CounterL, 1
        GOTO Loop
        DECFSZ CounterH, 1
        GOTO Loop.
```

$$\text{Delay} = \frac{4}{F_{osc}} * \#\text{ of } T_{inst}$$

T_{inst}

Trace the code.

P.20 Single loop:

① Initialization phase.
② All iteration except the last.
③ Last iteration.

P.20 Example 1: $T_{inst} = \frac{4}{800kHz} = 5\mu s$

① 1 (MOVLW) + 1 (MOVWF)
② 199 * (1(NOP) + 1(NOP) + 1(DECFSZ) + 2 (GOTO))
③ 1(NOP) + 1(NOP) + 2(DECFSZ)

Delay = $5\mu s * [2 + 199 * 5 + 4]$
= $5\mu s * 1001$
= 5.005 ms

* If the code is subroutine, then we add 2 T_{inst} for call → delay = 5.025 ms & 2 T_{inst} for return.

No.

P.20 Modify the code to give 3ms exact delay.

$$\text{Delay} = \frac{4}{F_{osc}} * \#T_{inst}$$

$$3*10^{-3} = 5*10^{-6} * \#\text{ of } T_{inst}$$

$$\#\text{ of } T_{inst} = 600$$

$$600 = [(1+1) + (X-1)*(1+1+1+2) + (1+1+2)]$$

$$600 = 2 + 5X - 5 + 4$$

$$600 = 1 + 5X$$

$$X = \frac{549}{5} = 119.8$$

Sol.

```
      MOVLW D'119'
      MOVWF counter
      NOP
      NOP
      NOP
      NOP
del   nop
      nop
```

FIVE APPLE

No.

* Write subroutine that gives 10 ms exact delay including the call instruction.
Assume $F_{osc} = 4$ MHz.

Delay = T_{inst} * # of T_{inst}

$10 * 10^{-3} = \frac{4}{4 * 10^{-6}}$ * # of T_{inst}

of T_{inst} = 10,000

Sub

No.

10 ms, F_{osc} = 4 MHz

Nested Loop :

① Initialization
② First external ~~iteration~~ iteration
  a. all internal ~~except~~ except last
  b. Last internal
③ All external except first & last.
④ Last external.

P.21 $10 * 10^{-3} = \frac{4}{4*10^6} * \#$ of Tinst

$\#T_{inst} = 10000$

~~10000~~ ① call 2 + (1 + 1 + 1 + 1 + 1) + nop MOVLW MOVWF

② a. 249 * (1 + 2) + DECFSZ goto
  b. (2 + 1 + 2) DECFSZ DECFSZ goto

+ ③ 11 * [255 * 3 + 5] +

④ 255 * 3 + (2 + 2 + 2)
DECFSZ return.

FIVE APPLE

Embedded

No. 8-87

P23★ When do we use indirect addressing?

eg. Write code to clear the data memory from address 0X10 to address 0X4F (64 locations)

★ Use indirect addressing when we need to manipulate the address.

1) address → FSR
2) replace (f) by either 00 or INDF

```
MOVLW 17.
MOVWF FSR.
CLRF  INDF
```

eg Complement the value in address 17 using indirect addressing.

```
MOVLW 17
MOVWF FSR
COMF  00, 1
```

No.

Bank selection: direct status(5)
indirect FSR(7)

e.g. Write a code to increment the value in address 0x85 using both direct & indirect.

0x85 = 1000 0101 Bank 1
address 5

Direct:

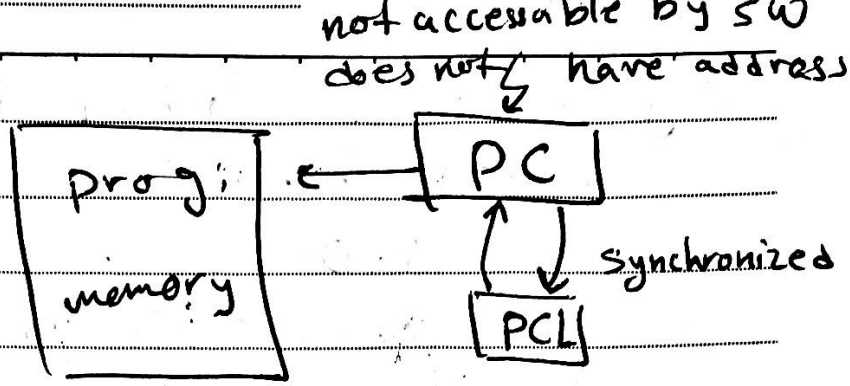
```
BSF   STATUS, 5
INCF  0x05, 1
```

In-direct:

```
~~BSF FSR, 7~~        MOVLW 0x85
~~MOVLW 0x05~~        MOVWF FSR
~~MOVWF~~             INC INDF, 1
```

* Lookup table: is a way to define any array & access any element in it.

No.




```
        MOVLW 3
        CALL SQUARES
        MOVLW 4
        CALL SQUARES
SQUARES ADDWF PCL, 1
        retlw 0
        retlw 1
        retlw 4
        retlw 9
        retlw D'16'
        retlw D'25'
```

No.

eg. Write a lookup table that when invoked, it returns x^2+5 for $X=0 \rightarrow 4$

For example, if you store 3 in w-Reg and invoked the lookup table it returns $14 = (3)^2+5$

```
Lookup   ADDWF   PCL, 1
         retlw   D'5'
         retlw   D'6'
         retlw   D'9'
         retlw   D'14'
         retlw   D'21'
```

for $X = 2 \rightarrow 6$

```
Lookup   MOVWF   Temp
         MOVLW   2
         SUBWF   Temp, 0
         ADDWF   PCL, 1
         retlw   D'9'
         retlw   D'14'
         retlw   D'21'
         retlw   D'30'
         retlw   D'41'
```

* Array of five elements using lookup table :
6 instructions and
2 instructions to access an element.

Chapter 6 :

P.3 Interrupt : code → starts at address 0004
ends with retfi
invoked by HW event.

Interrupt is high priority ~~code~~ code

Interrupt → external
↳ Internal.

P.6 maskable : can be disabled.
non-maskable : can't be disabled.

* For an interrupt to happen :
1) GIE = 1
2) Local interrupt enable = 1
3) Local interrupt flag = 1 (HW event)

5 enable: set/cleared by SW.

↓ └→

GIE 4 types of interrupt each has enable.

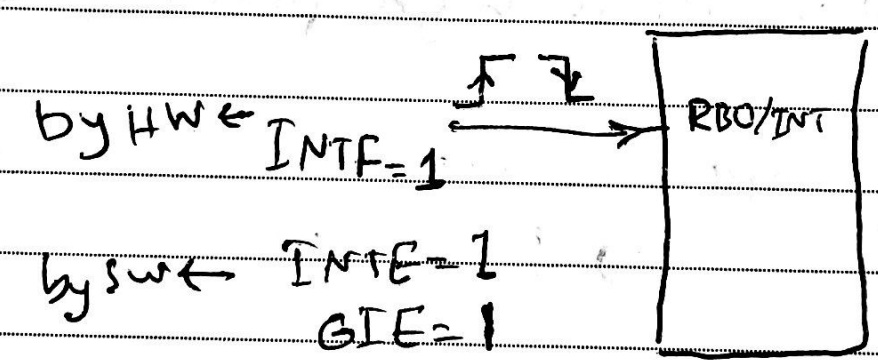
& flag

↑

set by HW.

P.7 * 4 Types of interrupts:

1) external interrupt:



2) Timer 0 overflow interrupt:
when Timer 0 finishes the time (overflow).

By HW: ~~TOIE~~ TOIF=1

By sw: TOIE=1

GIE=1

No.

3) Port B change interrupt:

By HW → RBIF=1

By SW → GIE=1

RBIE=1

4) EEPROM Write complete interrupt

Set WR bit

EEIF=1 ← By HW.

EEIE=1
GIE=1 } By SW

The PIC will wake up from sleep mode if an interrupt flag=1 while its corresponding enable=1 regardless of GIE value.

To deal with interrupt (10 bits)

↳ GIE bit
↳ 4 flag bits
↳ 4 local enable bit } INTCON except EEIF is in EECON1
↳ 1 bit ~~in option_reg~~

↑

in option_Reg (6)

FIVE APPLE

No.

P.11 Clear GIE: To prevent any interrupt from interrupting the current interrupt.

Q. Why we have to end the interrupt code by retfie?

A. Includes implicit set GIE.

P.12 2. If you deal with port B change interrupt, you should clear the flag (RBIF) at the beginning of the code.

6. Befor RETFIE, you should clear the flag in order not to get back into ISR after returning from interrupt.

Note: Interrupt flags are set by HW cleared by SW.

P.15 Context saving:

```
ISR     MOVF 10,0
        MOVWF Temp      ] → 10 → Temp

        MOVWF 10   changes [10] value.

        MOVF Temp,0
        MOVWF 10
        RETFIE
```

P.16 eg. write complete program such that :-

* when EEPROM write complete interrupt happen, increment the address is EEADDR ~~code~~ and set WR_bits.
* when external interrupt 7 happen, subtract 7 from address 16.
* When timer 0 overflow interrupt happen, multiply address 16 by 2.
* Do context saving for W-Reg.

~~#include~~

No.

Sol.

```
        #include   P16F84A.INC
        ORG        0000
        GOTO       START
        ORG        0004
        GOTO       ISR
START   BSF        INTCON, GIE
        BSF        INTCON , INTE
        BSF        INTCON , TOIE
        BSF        INTCON , EEIE
Loop    GOTO       Loop

ISR     MOVWF      Temp
        BTFSC      INTCON , INTF
        GOTO       external Code
        BTFSC      INTCON, TOIF
        GOTO       Timer0 Code
        BTFSC      INTCON , EEIF
        GOTO       EEcode
```

$$Delay = \frac{4}{F_{osc}} * prescale * (256 * IN)$$

e.g. Write code to complement the value in address 22, every 5 ms use timer0 & interrupt, assume F_{osc} = 800 kHz

$$5 * 10^{-3} = \frac{4}{800 * 10^3} * prescale * (256 - IN)$$

$$pre * (256 - IN) = 1000$$

pre = 2 → (256 - IN) = 500 → IN = -244 ✗

pre = 8 → (256 - IN) = 125 → IN = 131

```
        ORG   0000
        GOTO  START
        ORG   0004
        GOTO  ISR
START   BSF   INTCON, GIE
        BSF   INTCON, T0IE
        MOVLW D'131'
        MOVWF TMR0
```

No.

```
        Select Bank1
        MOVLW  B'XXOX 0010'
        MOVWF  OPTION_REG
Loop    GOTO   Loop

ISR     COMF   22,1
        MOVLW  D'131'
        MOVWF  TMR0
        BCF    INTCON,TOIF
        RETFIE
```

e.g. Write code to increment a counter every one second using Timer0. Assume $F_{osc} = 4\ MHz$

$$Delay = \frac{4}{F_{osc}} * pre * (256 - IN)$$

$$1 = \frac{4}{4*10^6} * pre * X$$

$$pre * X = 10^6$$

$$pre = 256 \longrightarrow X = \frac{10^6}{256} > 255$$

$$Max\ delay = \frac{4}{4*10^6} * 256 * 256 = 65.XX\ ms$$

No.

→ 1 = Counter * 10 ms

counter = 100

$10 * 10^{-3} = \frac{4}{4 * 10^6} * pre * X$

$pre * X = 10000$

$pre = 128$

$X = 78$

IN = 256 - 78 = 178

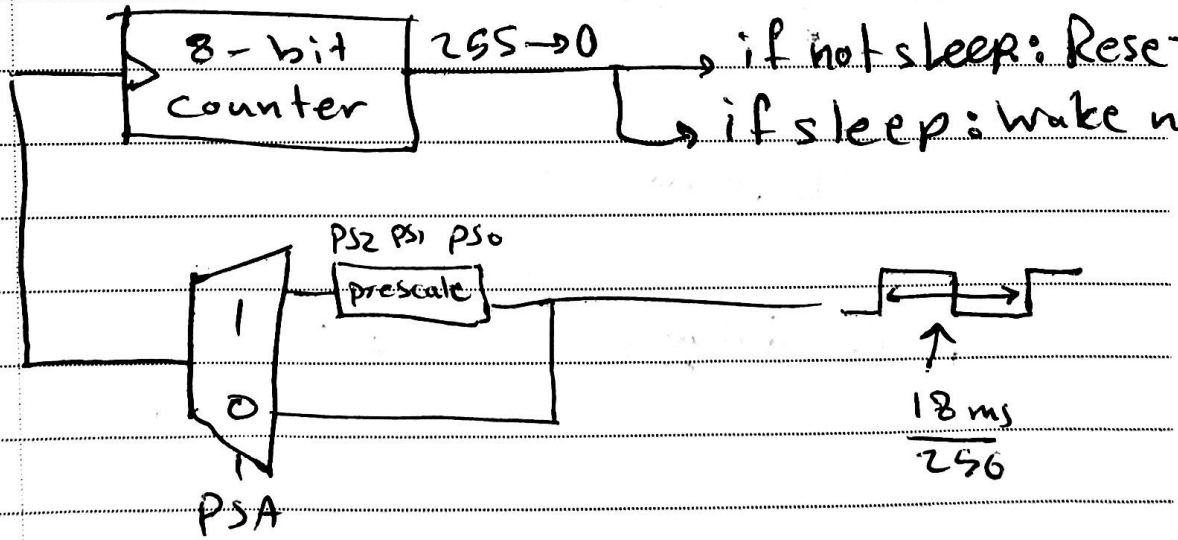
```
        ORG 0000
        GOTO START
        ORG 0004
        GOTO ISR
START   BSF INTCON, GIE
        BSF INTCON, TOIE
        MOVLW D'100'
        MOVWF 22
        MOVLW D'178'
        MOVWF TMR0
        BSF STATUS, RP0
        MOVLW B'XX0X 0110'
        MOVWF OPTION_REG
        BCF STATUS, RP0
```

No.

Loop ← GOTO Loop

```
ISR   BCF  INTCON, TOIF
      MOVLW  D'178'
      MOVWF  TMR0
      ~~DECFSZ Counter, 1~~
      DECFSZ  22, 1
      RETFIE
      INCF    17, 1
      MOVLW   D'100'
      MOVWF   22
```

P. 24 Watch Dog Timer:



No.

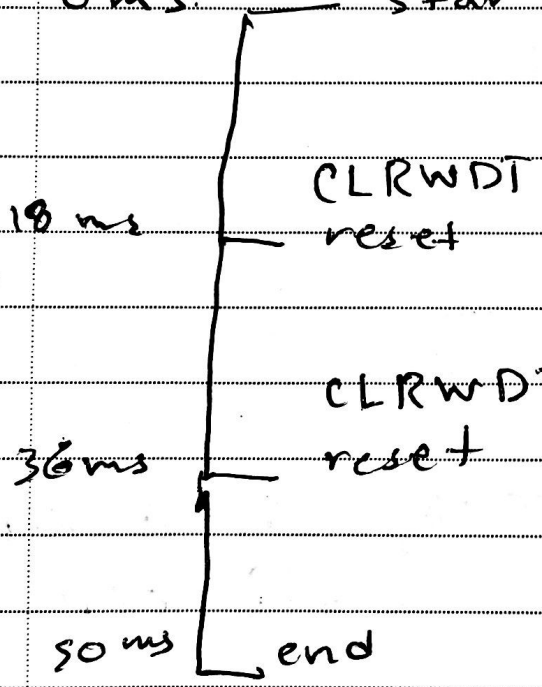
* After how long does WDT do reset

~~If all time = 18 ms~~

$\frac{18\text{ ms}}{256}$

$256 * \frac{18\text{ ms}}{256} * \text{prescale}$

delay = 18 ms * prescale.



Max delay = 18 ms * 128

= 2.3 second.

P.25 SLEEP:- It makes the CPU to enter sleep mode.

Saves power:- Turns osc off.
- Stops executing the code.
- all memory values are kept as they are.
- Clears WDT (Starts counting from 0)
- ~~Keeps~~ WDT keeps running in sleep mode (It has own RC Osc)

* To wake up from sleep:
1- WDT if enabled when overflows.
2- $\overline{MCLR} = 0$
3- Interrupt flag while its corresponding local enable = 1, Regardless of ~~GIE~~ GIE.

SLEEP → It makes the CPU enter the sleep mode, Max sleep time if WDT is enabled = 1.8 ms * ^{max}prescale
= 2.3 second

No.

Sleep wake up:

1- Interrupt $\begin{cases} GIE=0, & PC \\ GIE=1, & 0004 \end{cases}$

2- WDT ⟶ PC

3- $\overline{MCLR}$ ⟶ 0000

Note: All interrupts may wakeup PIC from sleep mode except timer 0 interrupt because the osc is off.

e.g. Write code to read address 07 (Port B) every 1.15 seconds.

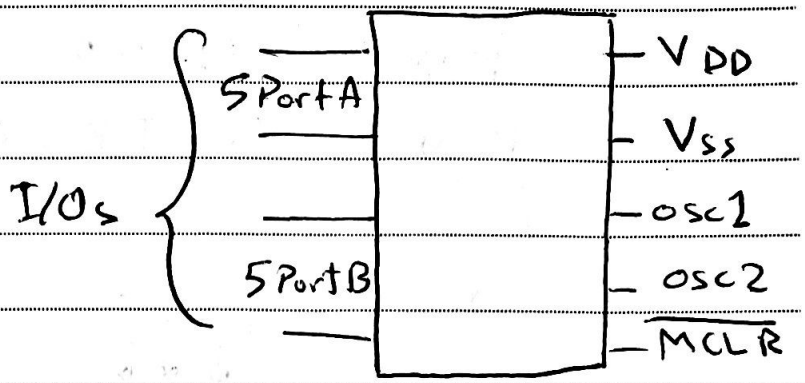
Sol. Enable WDTE in config word.

```
      ORG 0000
START BSF STATUS, RP0
      MOVLW B'XXXX1110'   ; PSA
      MOVWF OPTION_REG
      BCF   STATUS, RP0
Loop  Sleep
      MOVF  07, 0
      GOTO  Loop
      END
```

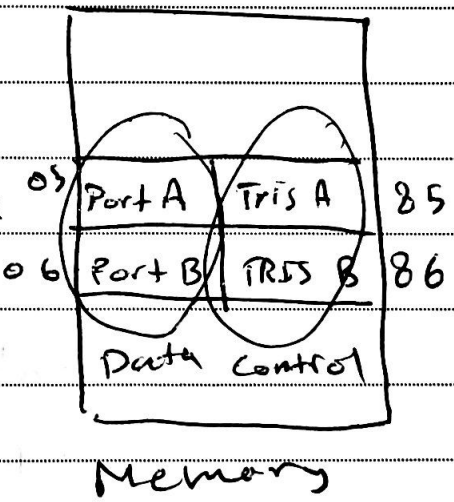
No.

→ WDT Delay: $\frac{1.15 \text{ sec}}{18 \text{ ms}} = 64$

Chapter 3: Parallel ports



TRISA, TRISB:
Identifies each pin whether its input or output.



* Every pin is independent from all other pins.
* All pins are half duplex
* TRIS bits = 0 → corresponding pins are O/P
  = 1 → Input.

No.

~~Example:~~ ~~BSF TRISB~~

~~MOVLW~~

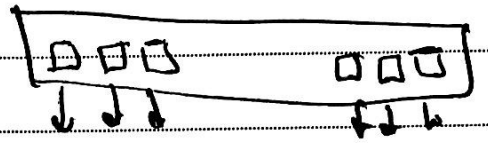
~~BSF TRISB,~~

Example: BSF STATUS, RP0

MOVLW B'11100111'

MOVWF TRISB

MOVF PORTB, 0

W = 

Depends on connected I/P devices.

Any instruction that modifies Port A or Port B registers effects only output ~~ports~~ pins. Because input pins take their value from outside.

External interrupt:

BCF PortB, 0 } No ~~external~~ external interrupt
BSF PortB, 0 }

positive edge must be external.

P.9 Example:

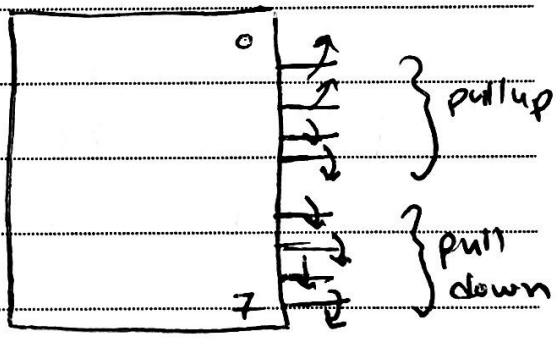
```
flash EQU  15
      ORG  0000
      GOTO START
      ORG  0004
      GOTO ISR
START BSF  INTCON , GIE
      BSF  INTCON, INTE
      BSF  STATUS, RPO
      MOVLW B' XXXX XX01'
      MOVWF TRISB
      BCF  STATUS, RPO
      CLRF PORTB ; LED is off
      CLRF flash ; Not flashing.
Loop  BTFSS flash.0
      Goto Loop
      MOVLW B'0000 0010'
      XORWF PORTB, 1
```

No.

→ delay 1 second.
Goto Loop
ISR MOVLW B'0000 0001'
XORWF flash, 1
BCF INTCON, INTF
retfie
end.

* Slides 12 – 17 باقي

P-19 e.g.



Bank1
MOVLW 0XFF
MOVWF TRISB
Bank0
MOVF PORTB, 0
MOVWF 0X1A

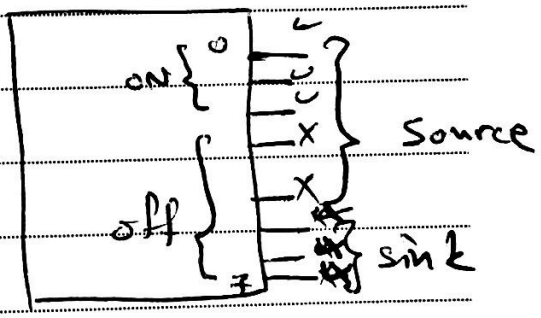
[0X1A] = 1111 0011 = F3

No.

OPTION-REG(7): RBPU = 0
: There is internal pull-up resistor connected to port B.

e.g. Write code to connect 8 LEDs to port B such that,
RB0 - RB4 as current source.
RB5 - RB7 as current sink.
RB0 - RB2 ON.
RB3 - RB7 off.

Sol.
B1 select
CLRF TRISB
B0 select
MOVLW B'1110 0111'
MOVWF PortB



P.26 Schmitt trigger: $V_{LH} > V_{HL}$
V_{HL}: If initially o/p = 1, to convert to 0, the o/p voltage must go less than V_{HL}.
V_{LH}: If initially o/p = 0, to convert to 1, o/p voltage must go above V_{LH}.

FIVE APPLE

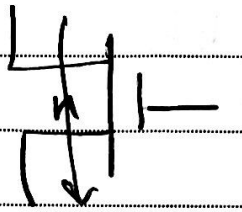
No.

→ Advantages of schmitt trigger:
① cancels the noise
② Fast switching.

slides 27 – 29 Ch6

P.30 The oscillator:

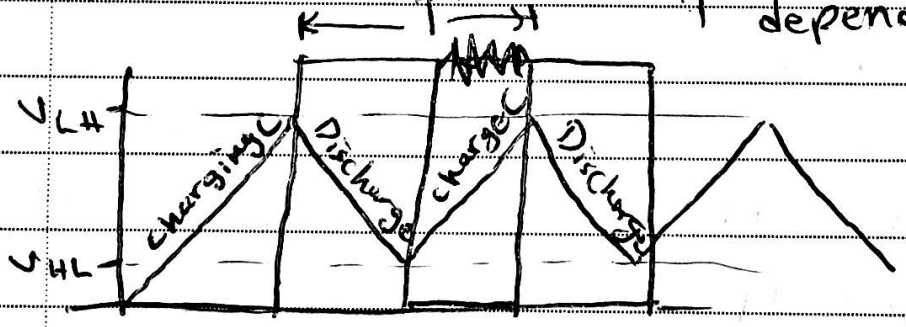
NMOSFET:



$V_G = 1$, S.C.

$V_G = 0$, O.C.

RC OSC:



T depend on C.

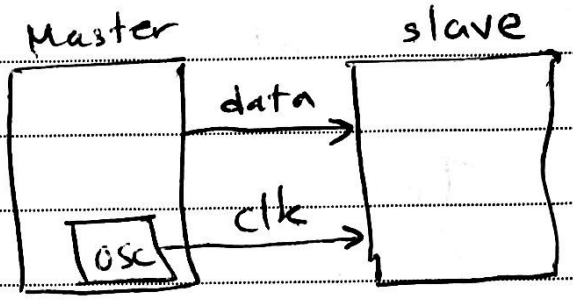
Crystal, Ceramic: stabelize faster.

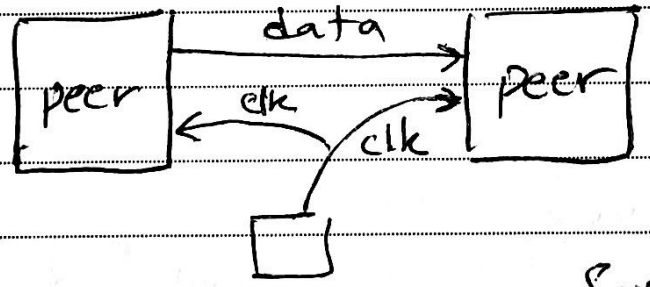
~~P.32 Capacitor in osc: stabalize faster~~

P.35 Push buttons with pull up resistor.

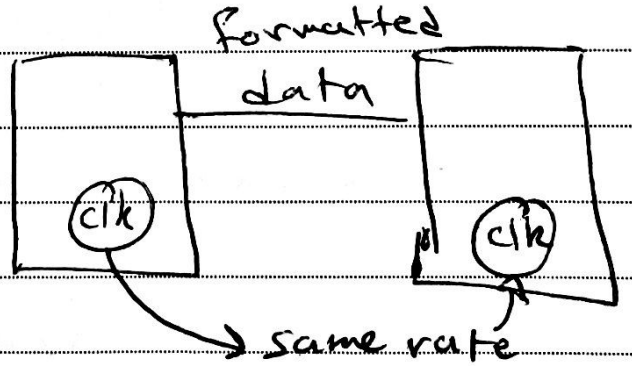
Chapter 10:

P.5 Synchronous:





Asynchronous:



* No 2 clock of exact same freq, to solve that, add stop & start bits to resynchronize after each word.

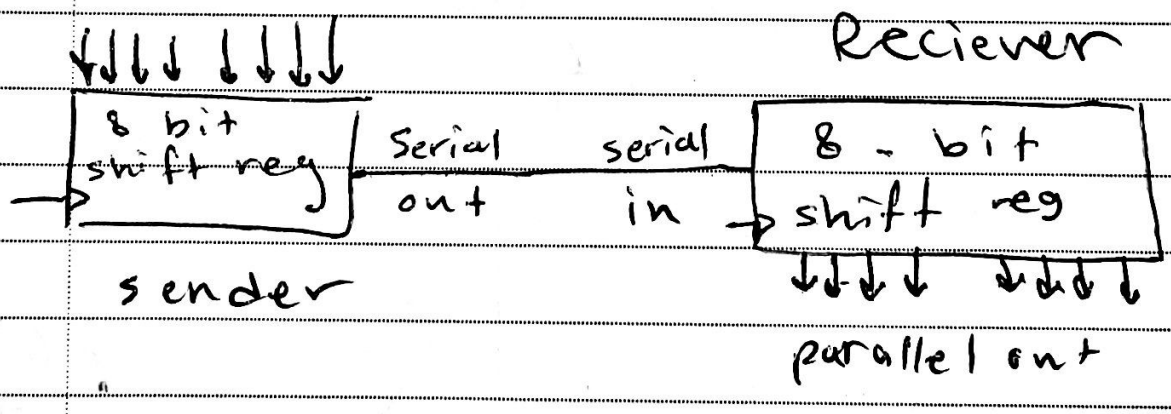
No.

* formatted data according to a protocol:

1) start bit(s)
2) stop bit(s)
3) parity
4) Data rate.

P.7 Serial communication:

Transmission: Parallel in, Serial out.
Reception: Serial in, Parallel out.



~~Acknowlege tha~~

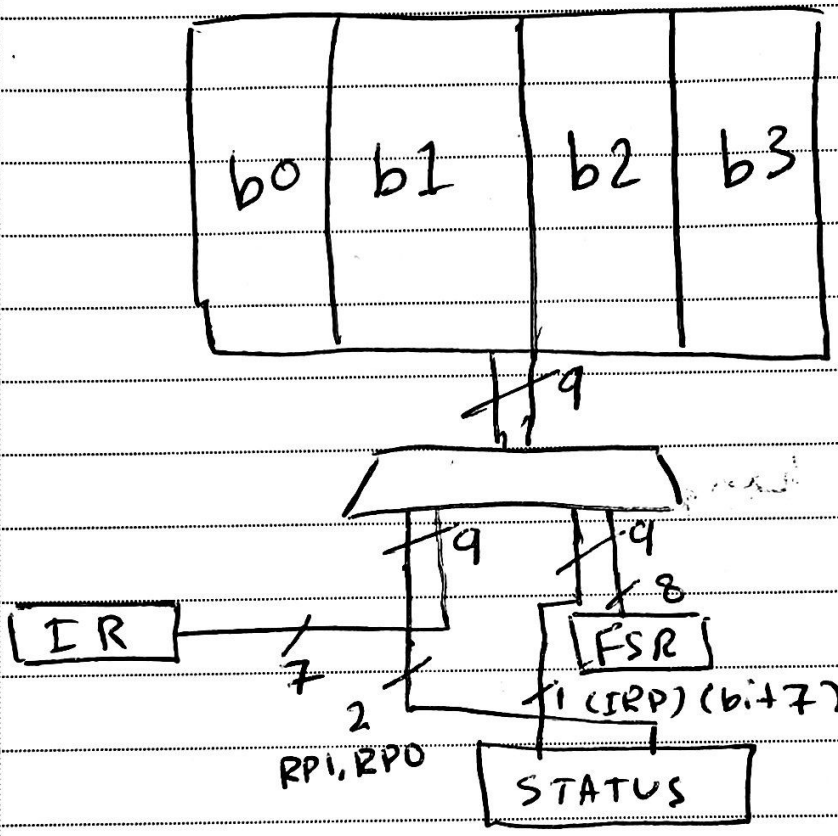
Acknowlege that Tx is over: TXIF=1

Acknowlege that a byte is recived:-
RCIF=1

No.

* من slide 16 في serial com برد

P.18 16F87XA



e.g. Write code to read the address 0x19B to w-reg use Direct & indirect : Bank select → 1 1001 1011

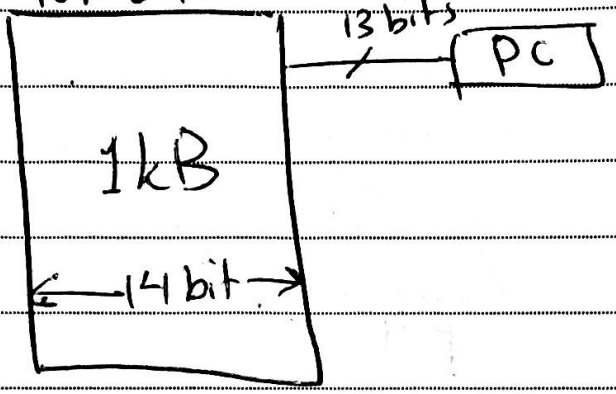
Solutions Direct: BSF STATUS, 6
BSF STATUS, 5
MOVF 0x15, 0

No.

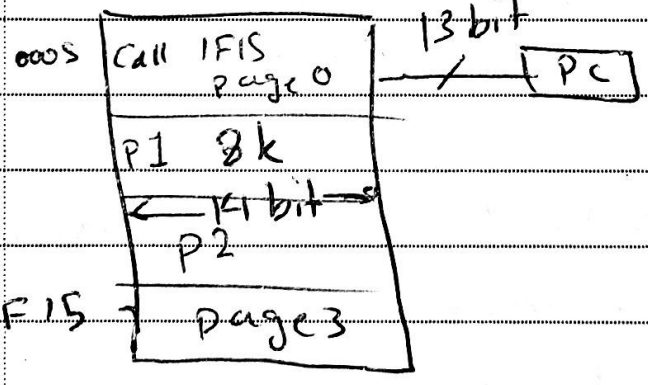
→ Indirect: BSF STATUS, IRP
MOVLW 0x9B
MOVWF FSR
MOVF INDF, 0

* Program Memory:

16F84A



16F87XA



IFIS = 1 1111 0001 0101

Call k k = 11 bits.
GOTO k Address = 13 bits.

No.

→ Select page first, then the address inside the page

page selection : Modify ~~PE~~ PCLATH<4:3>

→ CALL IF15 → compiler k=11bits.

Δ Can ~~IF15~~

110 111 0001 0101

opcode

call 715

BSF PCLATH,3
BSF PCLATH,4
CALL 715

P.16 Interrupts:

ALL new interrupts + EEPROM write complete interrupt, need:

① GIE ~~(2)~~ } By sw.
② PEIE
③ Local enable
④ Local flag } By HW

P.21 ~~TXEN, 1~~

~~TXSTA, TXREG~~

P.22 TSR Reg: Not accessable by sw (has no address).

A TXREG: Accessable, put & tranmitted byte in it.

* 2 enables: TXEN=1 for TX only.
SPEN=1 for Tx/Rx Serial port

* Pin Buffer and Control:
Transmit start & end bit.

* TX9=1: send extra bit with each byte. The 9th bit must be stored in TX9D.

* Once the byte in TXREG moves to TSR → TXIF=1

* Once TSR is empty → TRMT=1

Ihsham ahmad ali

No.

→ TXIF = 1 if GIE=1 TXIE=1 PEIE=1
Enable by sw.

* TXIF : Read only flag
set / Reset by HW.
=1 if TXREG is empty.
=0 if TXREG is full.

* Band rate = F (Fosc, BRGH bit, SPBRG reg.)

P.27 Asynchronous Receiver :

* RSR not accessable by sw.
* RCREG is accessable by sw.
  ↳ queue of two levels (FIFO)

* If the stop bit of the third word was received befor reading the previous 2 words → ① OERR = 1
  ② Third byte is lost.
  ③ reception stops.
  ↑ Read only.

~~* IOC~~ * To prevent it: read the byte as soon as it is received (RCIF = 1)
- RCIF = 1 : if at least there is one byte in RCREG.
- RCIF = 0 : if RCREG is empty.

↑ Read only.

GIE = 1, PEIE = 1, RCIE = 1

* To resolve OERR problems:
1- reset reception. CREN = 0 → OERR = 0
2- Read RCREG.
3- CREN = 1

No.

* For reception to be enabled:
1) SPEN=1 2) CREN=1

* Band rate = F(Fosc, BRGH, SPBRG)
bit

* If received stop bit=0 → framing error.
stop bit must be=1.

```
MOVF   RCREG,0
MOVF   RCREG,0
```

RCIF=0

P.33 Examples Transmit 0X40 - 0X4F

① $9.6 \times 10^3 = \frac{Fosc}{\frac{16}{64} * (1+SPBRG)}$

$9.6 \times 10^3 = \frac{20 * 10^6}{16 * (1+SPBRG)}$

$9.6 \times 10^3 * 16 * (1+SPBRG) = 20 * 10^6$

$SPBRG = \frac{20 * 10^6}{9.6 \times 10^3 * 16} - 1 = 129$

No.

② TRISC(6) = 1

TXSTA = X010 X 1XX

RCSTA = 1XXX XXXX

↑ SPEN

SPBRG = 129

③

```
        ORG  0000
START   BSF  TRISC,6
        BSF  TXSTA,2
        BSF  TXSTA,5
        BSF  RCSTA,7
        MOVLW D'129'
        MOVWF SPBRG
        MOVLW 0X40
        MOVWF FSR
LOOP    MOVF  INDF,0
        MOVWF TXREG
        INCF  FSR
LOOP2   BTFSC PIR1,TXIF
        GOTO  LOOP2
        MOVLW 0X50
        SUBWF FSR,0
        BTFSS STATUS,Z
        GOTO  LOOP
```

** ~~SLII~~ Chapter 11:

P.4 Analog: ⟶ ADC ⟶ Digital:

* Continuous in amplitude and time
* Unlimited: More accurate ⟶ Max acceptable error
* Discrete in amplitude & time
* Limited: Less accurate.

P.5 A ⟶ ADC ⟶ D

$F_s = 1/t_s$

↳ Sampling: Discrete in time
Read input analog signal every sampling time. (t_s)

↳ Quantizations: Discrete in amplitude.
Round the read sample to the closest acceptable digital value.

r (# of bits) ⟶ 2^r digital levels.

f_s increases ⟶ More accurate.
⟶ Need more storage.
⟶ Bandwidth.
⟶ Consume more power.
⟶ Higher cost.

No.

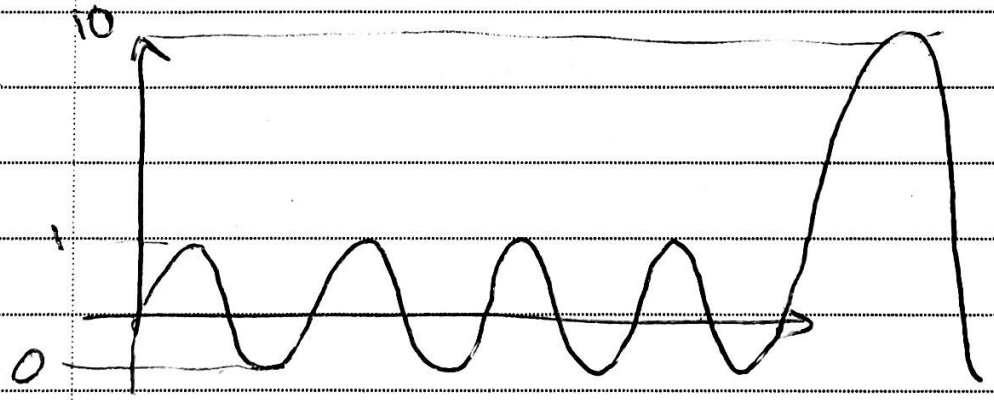
→ $f_s > 2f_{max}$ → aliasing.

ADC max sampling frcq = 16 kHz

f_{max} = 8 kHz.

r↑ → + accuracy.
- storage
- power
- Bandwidth
- cost
- slower.

Eg.



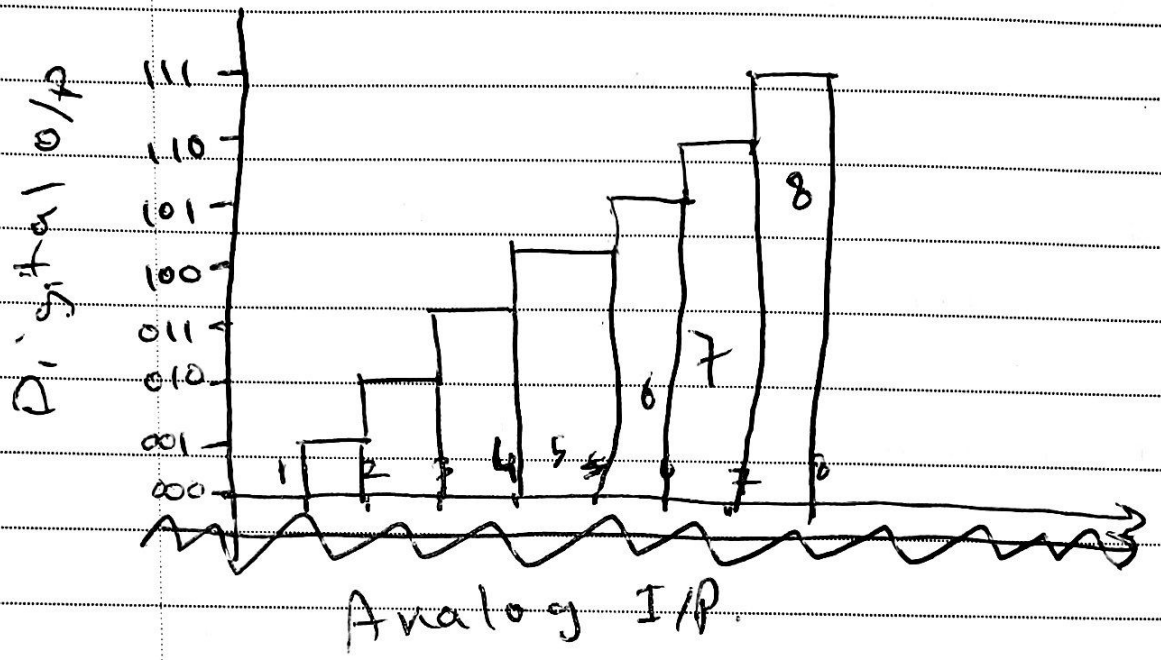
(0 → 10) 2^5 levels, low accuracy

(0 → 1) 2^5 levels, Higher accuracy.

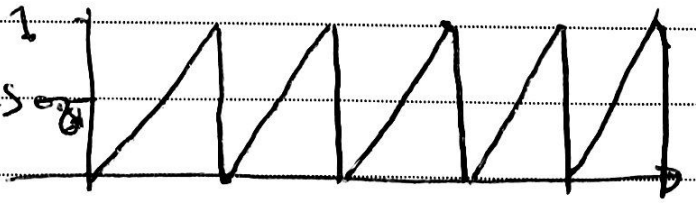
No.

P.6 sampling rate: according to the Input quantization characterestics graph.

Quantization:



error = input − output



Max error = $\frac{1}{2}V$

step width = $\frac{V_r^+}{2^n}$ = $\frac{V_{REF+} - V_{REF-}}{2^n}$

least significant bit voltage

n° number of bits.

FIVE APPLE

No.

→ Max error $\xrightarrow[\text{to}]{\text{goes}}$ 1 LSB = $\frac{V_r}{2^n}$

* Step size is smaller → less error.

* Step size is a measure of resolution.

* Smaller step size is more accurate.

P.7 Max error = $\frac{1}{2}$ LSB = $\frac{1}{2}\frac{V_r}{2^r}$

$= \frac{V_r}{2^{r+1}}$

Max acceptable error = $\frac{V_r}{2^{r+1}}$

LSB = Minimum change in analog input that difinetly will change the digital output.

$\frac{1}{2} f_{max}$

ADC → have tradeoff between accuracy & speed.

P.12 Sample & hold (Sampling): You can't read a sample while the input is changing.

* Before taking the sample fix the input.

P.14 → * Before taking a sample, fix (hold) the input, take sample, convert it, then release the input.

* When the ~~switch~~ switch is closed, the capacitor starts charging until it is close to V_s.

* When the switch gets opened, V_c almost fixed, you can take a sample.

* To take a sample, close the switch until $V_c \approx V_s$, then open the switch, V almost constant, read sample → convert.

No.

$$V_c = V_s * (1 - e^{-t/\tau}), \quad \tau = RC$$

* How long the switch should be kept closed.
It depends on "≈" (error)
"≈" ⟶ 10% error.
$V_c = 0.9 V_s$.

$$0.9 V_s = V_s * (1 - e^{-t/\tau})$$

$$t = 2.3 * \tau.$$

* max error $= \frac{V_r}{2^{n+1}}$

$$V_c = V_s - \frac{V_s}{2^{n+1}}$$

$$V_s - \frac{V_s}{2^{n+1}} = V_s * (1 - e^{-t/\tau})$$

$$t = -\ln\left(\frac{2^{n+1}-1}{2^{n+1}}\right) * \tau$$

* as n increases ⟶ sampling time ↑
accuracy ↑

No.

* If ADC is 8 bits:

$$t = -\ln \frac{1}{2^{8+1}} * \tau = -\ln \frac{1}{2^9} * \tau$$

$$= 6.2 * \tau$$

n = 10 bits → $t = 7.6 * \tau$.

P.19 — The 8 analog inputs are on:
RA0 – RA3, RA5, RE0 – RE2

To connect an analog input to RE0, what is the value of the selection line? Ans: 101

* 8 pins can still work as digital in addition to analog.

* You have to specify each of the pins whether it is digital or analog. PCFG(3:0)
They indentify whether voltage refrences internal (V_{DD}, V_{SS}) or external (RA3 & RA2).

No.

P.20 PIR1 : ADIF = 1
PIE1 : ADIE = 1
INTCON : GIE, PEIE
} interrupt.

P.21 ADC:
sampling $-\ln\frac{1}{2^{n+1}} * \tau$

Quantization: $(n+2) * T_{AD}$

T_{AD}: should be $\geq$ 1.6 μs for correct quantization if $T_{AD} <$ 1.6 μs, error quantization.

e.g. If F_{osc} = 10 MHz, what is the fastest quantization time for 10 bits ADC.

$T_{osc} = \frac{1}{10M} = 0.1$ μs,

choose $T_{AD} = 16\,T_{osc} = 1.6$ μs

quantization time = 12 * 1.6 μs = 19.2 μs

FIVE APPLE

eg. F_{OSC} = 100 kHz, T_{OSC} = 10 μs

$T_{AD} = 2\,T_{OSC}$ → T_{AD} = 20 μs (slow!!)
choose internal with T_{AD} = 2 μs.

Sample:

Sampling Time	Quantization Time	Intersampling Time
$-\ln\frac{1}{2^{n+1}} * T$	$(n+2) * T_{AD}$	$2\,T_{AD}$

next sample

← One sample time assuming repetitive sampling. →

$$\text{Sampling rate} = \frac{1}{T_s}$$

Max freq. that can be acquired $= \frac{1}{2} * \frac{1}{T_s}$

P.24 Step 5: start Quantization.

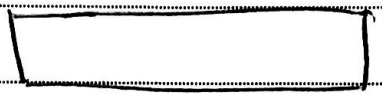
GO/$\overline{\text{DONE}}$ = 1 → start quantization, but you have to gurantee that a good sample was taken.

waited $-\ln\frac{1}{2^{n+1}} * T = 7.6 * T$ for 10 bit ADC.

No.

P.25 Step6: read result from ADRESH, ADRESL.

 Right justified

 left justified.

according to ADCON1<7>

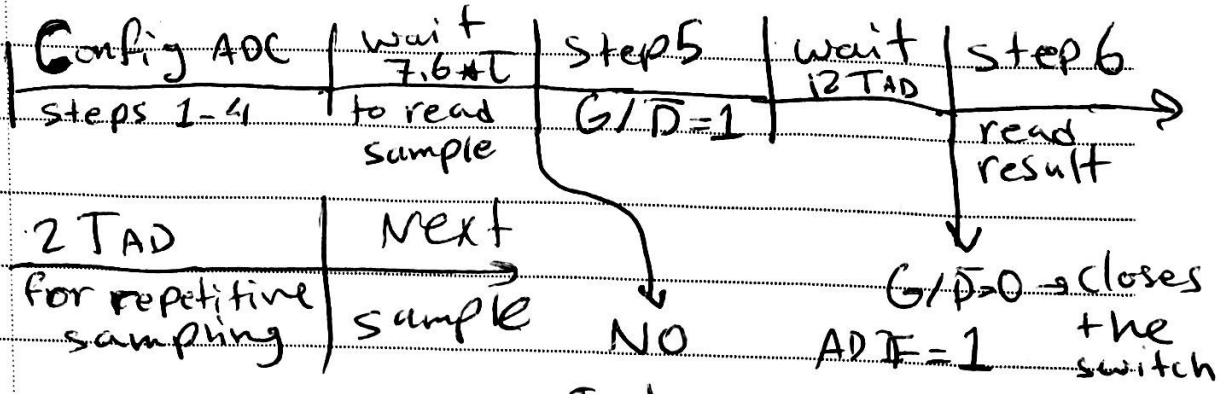
* Always choose right justified.
** Choose left justified only if you want to ~~convert~~ read the most sig. bits.

	10 0000 0001	513
1	0000 0001	Right just.
512 = 4 * 128	1000 0000	Left just

GO/$\overline{DONE}$ = 0 when quantization is over.
ADIF = 1 interrupt if GIE, ADIE, PIE = 1

No.

Converting :



Indecation
(You have to write code)

P.29 More accurate model for sampling ckt.

Sampling time = $7.6 * (R_S + R_{IC} + R_{SS}) * C_{HOLD}$
+ Op_Amp Settling time (2 μs)
+ Temp_coef
↳ = 0.05 μs for each C° > 25°

P.31 One sample assume = 29 μs + 2 * 1.6 μs
repetitive sampling = 32.2 μs

Maximum sampling rate (including repetitive) = $\frac{1}{32.2\,\mu s}$ = 31 KHz

Max freq = $\frac{1}{2} * 31\,KHz = 15.5\,kHz$

No.

Q. What the value of PCFG should be to connect an analog input to AN7 & voltage references to be external?

Ans. From P. 27 : 1000

Q. To connect only one analog input & external voltage references.

Ans. From P.27 : 1111

P.27 C/R column :

C: # analog inputs.

R: # external voltage references.

Step 1-4	wait sampling time	G/$\overline{D}$=1	wait	read result

G/$\overline{D}$ = 0

ADIF = 1

e.g. Write code that keeps reading 2 analog inputs from RA0& RA1, and output the 8 bit result to port B.

Chapter 8:

P.7 To read which button was pressed, we use port B change interrupt.

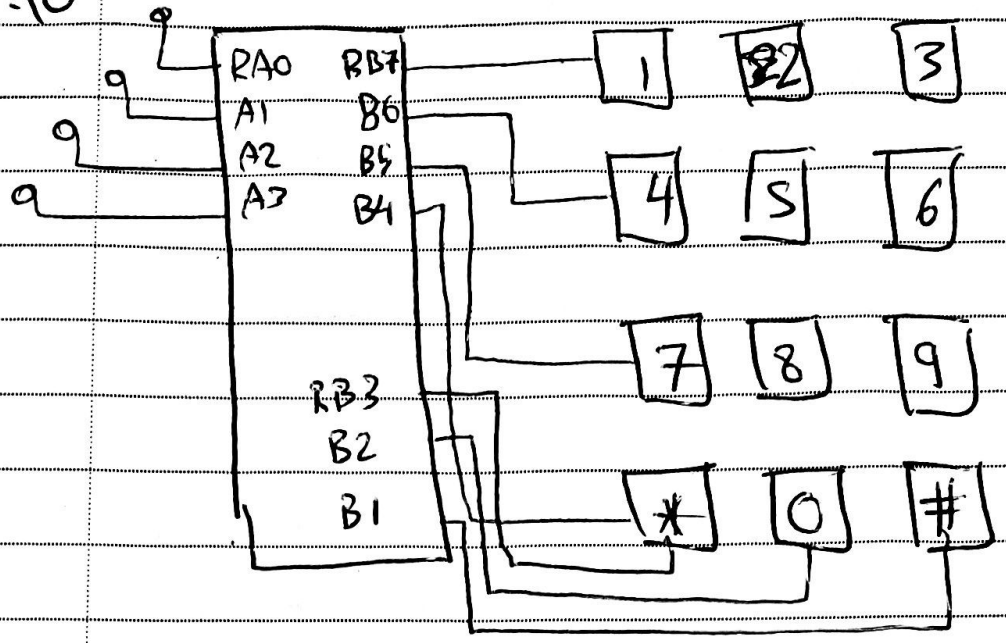
1) Read row pins:- Make row pins input, make column pins output, output zero on column pins.

2) Read column pins: Make column pins input, make row pins output, output zero on row pins.

* Press 9 : row pattern = 1101 XXXX
  column pattern = XXXX & 1101
  row # : 2
  column # : 2

No.

P.10



P.11 Port B change interrupt:

To clear RBIF:

1) Read port B, MOVF PORTB, 0

2) Clear flag, BCF INTCON, RBIF

P.12 Press 7:

Row Index = 1101 0000

column index = 0000 0110

W_Reg = ~~1101 0000~~

0000 0110

Convert, Find row:

Row_Index = 2

column_Index = 0

W_Reg = ~~0~~ 2

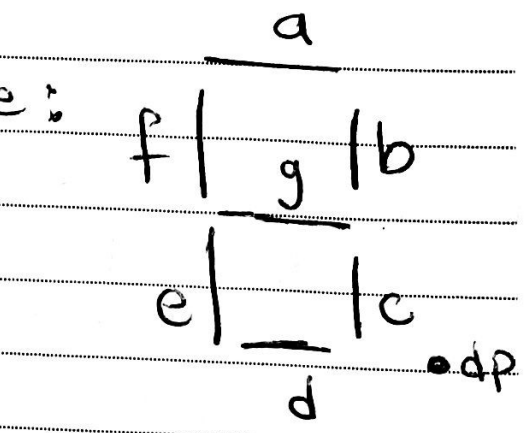
No.

P.14 Compute value:

W = 3 * Row_Index + Column_Index.

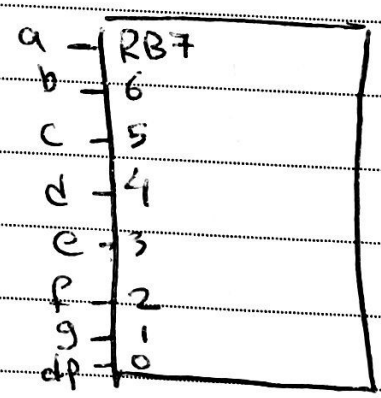
LED Displays:

Common Cathode:

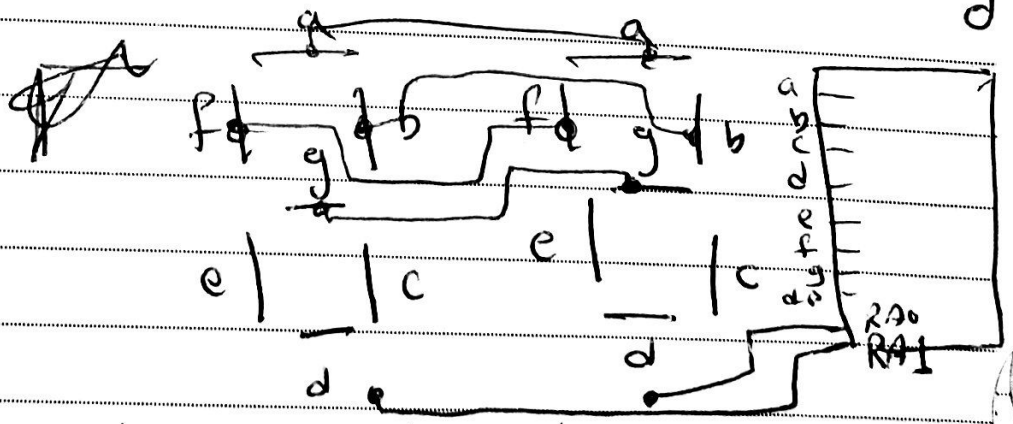


Display 6:

```
Bank1
MOVLW 0x00
Bank0
MOVLW B'1011 1110'
MOVWF PortB.
```



Connecting ~~Multiple~~ Multiple 7-seg.



connect Leds together

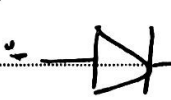
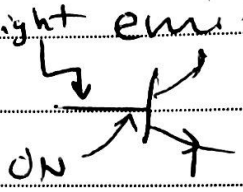
No.

Display 26:

```
        Bank 1
        CLRF TRISB
        CLRF TRISA
        ~~bsf~~ Bank 0
Loop    BSF PORTA, 0
        BCF PORTA, 1
        MOVLW B'1011 1110'
        MOVWF PORTB
        Delay 10 ms
        BCF PORTA, 0
        BSF PORTA, 1
        MOVLW B'1101 1010'
        MOVWF PORTB
        Delay 10 ms
        GOTO Loop
```

No.

P.29 Optical object sensor:

- IR diode:  , when On, it emits IR.
- Light transistor: 

- Plastic housing: prevents seen ligh from passing, only allows unseen lights to pass.

P.36 Servo motor: signal freq = 50Hz T = 20ms

to rotate 35°:

$$35 * \frac{1.5ms - 1.25ms}{90} + 1.25$$

To write code to rotate 90°:

```
Loop  BSF PortB, 1
      delay 1.5ms
      BCF PORTB, 1
      delay 18.5ms
      goto Loop.
```

P.43 L293D: It drives motor by external power supply.

- 4 half bridges: motors rotate in one direction, OR:
- 2 full H-bridges: motors rotate in two directions.

16 pins:

* 2 voltage sources V_{LS}, V_{OS} **
* 4 ground & Heat sink.
* 2 enable pins.
* 4 inputs (A)
* 4 outputs (Y)

** V_{LS} : Logic source voltage.
- Logic=1 → V_{LS}

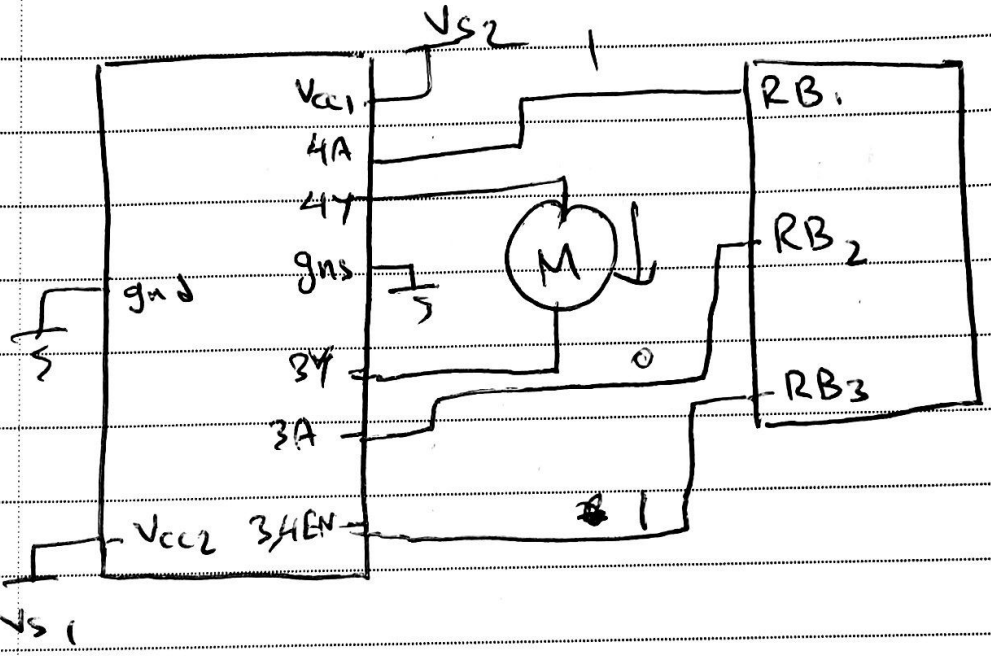
$V_{OS} > V_{LS}$

No.

Do all required connections on the figure given to:

* Make the current ↓ passes in the motor when RB1=1, RB2=0.
* current ↑ when RB1=0, RB2=1
* motor off when RB3=0.

And write code to make the current ↓.



Code :

```
BSF PORTB, 3
BSF PORTB, 1
BCF PORTB 2
```

No.

P.50 switch debouncing.

↳ HW Filter with schmitt trigger.

↳ SW Delay.

$

SW :

```
ISR     delay 20 ms
        BCF  INTCON, INTF
        INCF Counter, 1
        retfie
```

Delay > Bouncing time.