

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !**

**P1. (1,5 punts)** Indica si les següents afirmacions sobre el PIC18F són certes (C) o falses (F). (Encert +0.3, Error -0.2)

- ☐ El byte menys significatiu d'una instrucció, s'emmagatzema sempre en una adreça parell de la memòria de programa
- ☐ Treballar usant 'Access bank' a la RAM és més ràpid que treballar amb la RAM en mode banked i disposem de més memòria
- ☐ Usant una Fosc de 4MHz, el temps d'execució d'una instrucció (no de salt) 'single-word' serà d'1µs
- ☐ La instrucció 'ADDWF INDF1, W, A' accedeix al banc de memòria indicat a FSR1H
- ☐ El bus d'adreces de la memòria de dades és de 12 bits, això permet tenir fins a 4096Bytes de dades sense haver de modificar l'arquitectura del PIC18F

**P2. (1,5 punts)** Una persona d'una altra facultat ha decidit fer ús de chatGPT per escriure una funció en ASM per llegir el valor del Program Counter d'un PIC18F45k22. Aquesta és la conversa amb les respostes reals de la IA. Comenta els errors i proposa una solució millor:

**Usuari:** Hola!

**ChatGPT:** Hola! Com et puc ajudar avui?

**Usuari:** Pots donar-me un exemple de com llegir el PC d'un PIC18F?

**ChatGPT:** Sí, es possible llegir el valor del Program Counter (PC) d'un PIC18F a través del llenguatge d'assemblea de baix nivell.

Aquí tens un exemple de codi en assemblea per llegir el valor del PC d'un PIC18F:

```
; Inici del programa
org 0x000 ; Adreça d'inici del programa

main:
    ; Llegir el valor del PC
    movf PCL, W ; Mova el contingut del registre PCL a la memòria temporal W
    ; Bucle infinit
    goto main
```

**Usuari:** Moltes gràcies!

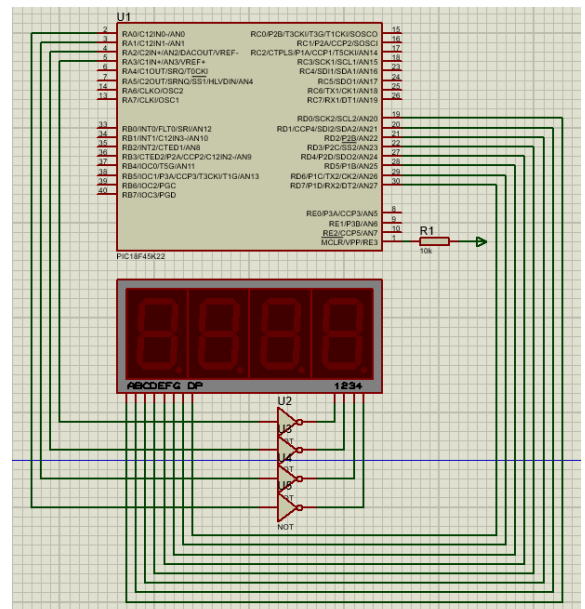
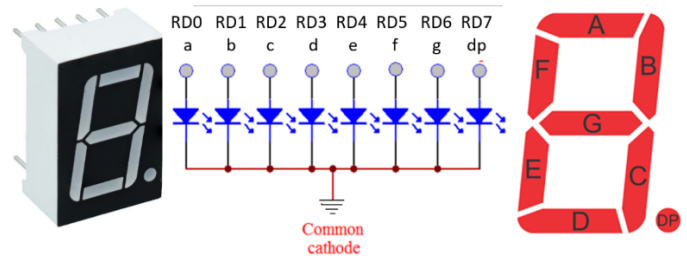
**ChatGPT:** De res, sempre estic aquí per ajudar-te. Si tens alguna altra pregunta o necessites ajuda, no dubtis en demanar-m'ho!

**P3. (2 punts)** Tenim 4 displays 7-segments connectats a un PIC18F45K22 seguint el mateix esquema que a la placa de desenvolupament feta servir al laboratori:

La gestió de la selecció dels displays és: al treure un "1" al pin corresponent del PORTA, es selecciona el càtode comú d'un display i aquest s'encén. RA0 selecciona el display de més a la dreta, RA3 el de més a l'esquerra.

3.1 (0,5p) Quin valor tindran els registres ANSELA, ANSELD, TRISA i TRISD?

3.2 (1,5p) Dibuixa un cronograma en el que indiquis els valors que has d'escriure als registres LATA i LATD (el PORTA i el PORTD) per tal de veure el número '3.142' als displays (atenció al punt)



Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

Tota resposta sense justificar es considerarà nul·la !

**P4. (3 punts)** El proper diumenge 7 de maig Barcelona acull la cursa popular de 10Km d’El Tall Anglès. Els organitzadors de la carrera ens demanen que implementem un cronòmetre més precís que el que usaven fins ara. Abans mesuraven el temps de pas dels corredors en segons. Ens demanen que ara mesurem el temps en centèsimes de segon. Farem servir el micro PIC18F45K22, amb una Fosc=12MHz. Cal utilitzar el **Timer2**, configurat per tal que cridem **la seva RSI exactament cada centèsima**, dins de la qual s’incrementarà la variable global *csCount*.

4.1 (1,5 punts) Configura els SFR necessaris per tal que el Timer2 generi un Interrupt Flag cada centèsima.

4.2 (1 punt) Configura els SFR necessaris del sistema d’interrupcions per tal que es cridi a la **RSI d’alta prioritat** amb cada Flag de Timer2.

4.3 (0,5 punts) El reglament de la cursa diu que tancarà l’arribada a meta al cap d’1 hora i 45 minuts de la sortida. Amb quin tipus de dades en C declararies la variable *csCount* per tal que pugui emmagatzemar valors fins el temps màxim de la prova, usant el mínim espai de memòria possible? (en la taula de la dreta pots veure quins tipus de dades admet XC8)

XC8 INTEGER DATA TYPES	
Type	Size (bits)
unsigned char	8
unsigned int	16
unsigned short long	24
unsigned long	32

**P5. (2 punts)** La pregunta anterior proporciona un comptador de centèsimes des de l'inici d'una prova esportiva. Considereu que cada cop que un corredor/a creua la meta, es genera un flanc ascendent en el pin INT1, i llegint el PORT C podem obtenir un codi de 8bits que identifica el corredor que acaba de passar. La RSI també fa una còpia del valor actual de centèsimes per guardar el temps d'aquest corredor/a.

Suposeu que ja està tot configurat per tal que es cridi la RSI d'alta prioritat amb cada flanc ascendent al pin INT1 i amb cada Flag del Timer2.

A continuació teniu el codi font assembler amb la RSI:

```
csCount    equ 0x000

            org 0x08

checkTMR2:  BTFSS PIR1,1,A
            BRA checkINT1
            BTFSS PIE1,1,A
            BRA checkINT1
            BCF PIR1,1,A
            CALL 0x200 ; increments csCount
checkINT1:  BTFSS INTCON3,0,A
            BRA finishRSI
            BTFSS INTCON3,3,A
            BRA finishRSI
            BCF INTCON3,0,A
copyId:     MOVF PORTC,W,A
            MOVWF POSTINC0,A
            LFSR FSR1,csCount
            MOVLW 3
copyTime:   MOVFF POSTINC1,POSTINC0
            CPFSEQ FSR1L,A
            BRA copyTime
finishRSI:  RETFIE FAST
```

5.1 (1 punt) Suposeu que hem entrat a la RSI perquè **ha vingut un Flag de INT1 però no de TMR2 (INT1IF=1, TMR2IF=0)**. Sabem que  $F_{osc}=12\text{MHz}$ . Calculeu el temps d'execució de la RSI.

5.2 (1 punt) Si a continuació del codi de dalt afegíssim una RSI de baixa prioritat, escriuríem una directiva `org 0x18`, i tot seguit el codi de la RSI\_low. ¿Per a què afegim les directives `org 0x08` i `org 0x18`?

Amb el codi de l'enunciat, tindríem algun problema en relació amb les directives *org*? Si fos el cas, com ho solucionaríeu?

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P1. (1,5 punts)** Indica si les següents afirmacions sobre el PIC18F són certes (C) o falses (F). (Encert +0.3, Error -0.2)

- ☒ **C** El byte menys significatiu d'una instrucció, s'emmagatzema sempre en una adreça parell de la memòria de programa
- ☐ **F** Treballar usant 'Access bank' a la RAM és més ràpid que treballar amb la RAM en mode banked i disposem de més memòria
- ☒ **C** Usant una Fosc de 4MHz, el temps d'execució d'una instrucció (no de salt) 'single-word' serà d'1µs
- ☒ **C** La instrucció 'ADDWF INDF1, W, A' accedeix al banc de memòria indicat a FSR1H
- ☒ **C** El bus d'adreces de la memòria de dades és de 12 bits, això permet tenir fins a 4096Bytes de dades sense haver de modificar l'arquitectura del PIC18F

**P2. (1,5 punts)** Una persona d'una altra facultat ha decidit fer ús de chatGPT per escriure una funció en ASM per llegir el valor del Program Counter d'un PIC18F45k22. Aquesta és la conversa amb les respostes reals de la IA. Comenta els errors i proposa una solució millor:**Usuari:** Hola!**ChatGPT:** Hola! Com et puc ajudar avui?**Usuari:** Pots donar-me un exemple de com llegir el PC d'un PIC18F?**ChatGPT:** Sí, es possible llegir el valor del Program Counter (PC) d'un PIC18F a través del llenguatge d'assemblea de baix nivell.

Aquí tens un exemple de codi en assemblea per llegir el valor del PC d'un PIC18F:

```
; Inici del programa
org 0x000 ; Adreça d'inici del programa

main:
    ; Llegir el valor del PC
    movf PCL, W ; Mova el contingut del registre PCL a la memòria temporal W
    ; Bucle infinit
    goto main
```

**Usuari:** Moltes gràcies!**ChatGPT:** De res, sempre estic aquí per ajudar-te. Si tens alguna altra pregunta o necessites ajuda, no dubtis en demanar-m'ho!**ChatGPT ha comès diversos errors i omissions:**

No fa 'include' del processador que fem servir, no especifica les adreces de memòria de programa on va cada secció del codi, té errors de traducció (Llenguatge d'assemblea) i, el més important, en el seu codi només llegeix la part baixa del registre PC (cal llegir també els altres dos registres que el componen i fer-ho en l'ordre adequat). Un codi que sí que llegeix el PC pot ser:

```
#include p18XXXXX.inc ; Include register definition file
; Inici del programa
org 0x000 ; Adreça d'inici del programa
    goto main ; Saltem a l'inici del programa
org 0x008 ; Aquí anirà l'RSI d'alta prioritat (ara no ens cal)
org 0x018 ; Aquí anirà l'RSI de baixa prioritat (ara no ens cal)

main
```

```

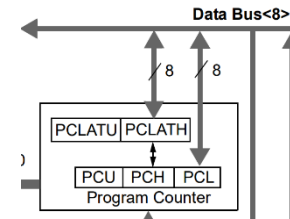
; Llegir el valor del PC
movff PCL, PCL_tmp ; Movem el contingut del registre PCL a algun registre
; Llegir el registre PCL actualitza els valors de PCLATH i PCLATU
movff PCLATH, PCH_tmp ; Movem el contingut del registre PCH a algun registre
movff PCLATU, PCU_tmp ; Movem el contingut del registre PCU a algun registre

```

```

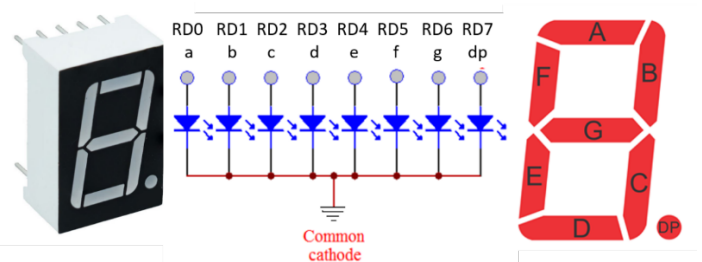
loop bra loop; Bucle infinit
end

```



**P3. (2 punts)** Tenim 4 displays 7-segments connectats a un PIC18F45K22 seguint el mateix esquema que a la placa de desenvolupament feta servir al laboratori:

La gestió de la selecció dels displays és: al treure un “1” al pin corresponent del PORTA, es selecciona el càtode comú d’un display i aquest s’encén. RA0 selecciona el display de més a la dreta, RA3 el de més a l’esquerra.



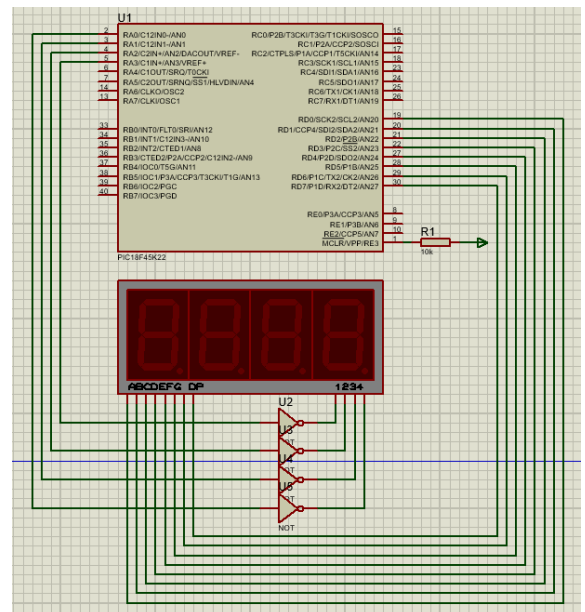
3.1 (0,5p) Quin valors tindran els registres ANSELA, ANSELD, TRISA i TRISD?

Hem de configurar els pins com a outputs digitals:  
 ANSELA=0xX0  
 ANSELD=0x00  
 TRISA=0x00  
 TRISD=0x00

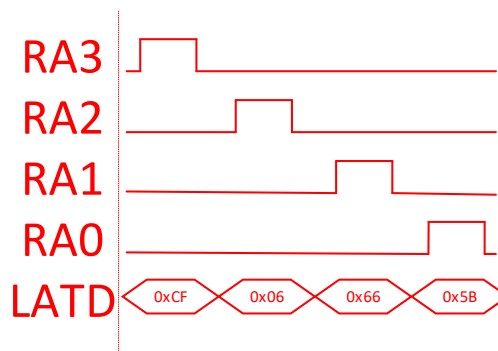
3.2 (1,5p) Dibuixa un cronograma en el que indiquis els valors que has d’escriure als registres LATA i LATD (el PORTA i el PORTD) per tal de veure el número ‘3.142’ als displays (atenció al punt)

La codificació del PORTD per encendre el 7-segments amb els dígets que volem és:

‘3.’ -> PORTD=0xCF  
 ‘1’ -> PORTD=0x06  
 ‘4’ -> PORTD=0x66  
 ‘2’ -> PORTD=0x5B



Un cronograma d’activació dels PORTS que mostra els dígets que volem i que no produeix ghosting pot ser:



Aquest cronograma mostra una iteració d’un bucle que s’ha de repetir mentre volem que es mostrin els dígets als 7-segments.

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !**

**P4. (3 punts)** El proper diumenge 7 de maig Barcelona acull la cursa popular de 10Km d'El Tall Anglès. Els organitzadors de la carrera ens demanen que implementem un cronòmetre més precís que el que usaven fins ara. Abans mesuraven el temps de pas dels corredors en segons. Ens demanen que ara mesurem el temps en centèsimes de segon. Farem servir el micro PIC18F45K22, amb una Fosc=12MHz. Cal utilitzar el **Timer2**, configurat per tal que cridem **la seva RSI exactament cada centèsima**, dins de la qual s'incrementarà la variable global **csCount**.

4.1 (1,5 punts) Configura els SFR necessaris per tal que el Timer2 generi un Interrupt Flag cada centèsima.

El Timer2 va comparant tota l'estona el valor del comptador amb el PR2. Quan són iguals, diem que hi ha un "match". Cada N matches (configurat amb el Postscaler) llavors el mòdul posa TMR2IF=1.

$$T_{INT} = \frac{4}{F_{OSC}} \cdot PRE \cdot (PR2 + 1) \cdot POST = \frac{4}{12MHz} \cdot PRE \cdot (PR2 + 1) \cdot POST$$

L'enunciat diu que volem un Flag d'Interrupció exactament cada centèsima de segon, per tant:

$$T_{INT} = 0.01s = \frac{4}{12MHz} \cdot PRE \cdot (PR2 + 1) \cdot POST \Rightarrow PRE \cdot (PR2 + 1) \cdot POST = 30000$$

El prescaler pot valer 1, 4 o 16.

Veiem el cas PRE=1:  $(PR2 + 1) \cdot POST = 30000$ . El POST més gran possible és 16, el PR2 màxim és 255. La multiplicació dona  $(255 + 1) \cdot 16 = 4096$ . No arribem a 30000, per tant PRE=1 descartat!

Veiem el cas PRE=4:  $(PR2 + 1) \cdot POST = 7500$ . Tampoc podem arribar a 7500, per tant PRE=4 descartat!

Veiem el cas PRE=16:  $(PR2 + 1) \cdot POST = 1875$ . Ara sí que podríem arribar a 1875, per tant ens quedem PRE=16 i avaluem diferents POST:

¿POST=16? Això donaria  $(PR2 + 1) \cdot 16 = 1875 \Rightarrow (PR2 + 1) = 117.19$ . Dóna decimals, descartat! (volem interrupció cada centèsima **exacta**!)

¿POST=15? Això donaria  $(PR2 + 1) \cdot 15 = 1875 \Rightarrow (PR2 + 1) = 125$ . Sense decimals, perfecte! Per tant:

A **T2CON** hem de configurar:

A **PR2** hem de gravar el valor  $125 - 1 = 124$ .

- **POST=15 (T2OUTPS = 0b1110)**
- **PRE=16 (T2CKPS = 0b1x)**
- **TMR2ON=1 (per activar el Timer)**

4.2 (1 punt) Configura els SFR necessaris del sistema d'interrupcions per tal que es cridi a la **RSI d'alta prioritat** amb cada Flag de Timer2.

Una possible opció és:

IPEN = 1 // habilitem gestió de nivells de prioritat.

GIEH = 1 // habilitem el Global Interrupt Enable de les High Priority

GIEL = x // m'és igual si habilitem o no les Low. L'enunciat no especifica res d'aquest cas.

TMR2IP = 1 // configurem que la interrupció de Timer2 sigui d'alta prioritat

TMR2IF = 0 // opcionalment podríem netejar el Flag de Timer2 abans d'habilitar-la

TMR2IE = 1 // per últim, cal habilitar la font d'interrupció de Timer2 amb el seu Interrupt Enable.

4.3 (0,5 punts) El reglament de la cursa diu que tancarà l'arribada a meta al cap d'1 hora i 45 minuts de la sortida. Amb quin tipus de dades en C declararies la variable **csCount** per tal que pugui emmagatzemar valors fins el temps màxim de la prova, usant el mínim espai de memòria possible? (en la taula de la dreta pots veure quins tipus de dades admet XC8)

**XC8 INTEGER DATA TYPES**

Type	Size (bits)
unsigned char	8
unsigned int	16
unsigned short long	24
unsigned long	32

Primer hem de saber quin valor màxim pot arribar a valer la variable **csCount** (son centèsimes).

Temps màxim = 1h + 45min = 60min + 45min = 105min. Expressat en centèsimes seria:

$$105min \cdot \frac{60seg}{1min} \cdot \frac{100cs}{1seg} = 630000 \text{ cs}$$

El valor 630000 no cap en un char (màxim 255), ni en un int (màxim 65535).

Un short long admetria fins  $2^{24} - 1 = 16777215$ . Aquí sí cap, per tant usariem **unsigned short long**.

**P5. (2 punts)** La pregunta anterior proporciona un comptador de centèsimes des de l'inici d'una prova esportiva. Considereu que cada cop que un corredor/a creua la meta, es genera un flanc ascendent en el pin INT1, i llegint el PORT C podem obtenir un codi de 8bits que identifica el corredor que acaba de passar. La RSI també fa una còpia del valor actual de centèsimes per guardar el temps d'aquest corredor/a.

Suposeu que ja està tot configurat per tal que es cridi la RSI d'alta prioritat amb cada flanc ascendent al pin INT1 i amb cada Flag del Timer2.

A continuació teniu el codi font assembler amb la RSI:

```
csCount    equ 0x000

org 0x08

checkTMR2: BTFSS PIR1,1,A    +1 cicle (no salta)
            BRA checkINT1    +2 cicles
            BTFSS PIE1,1,A
            BRA checkINT1
            BCF PIR1,1,A
            CALL 0x200 ; increments csCount
checkINT1:  BTFSS INTCON3,0,A +2 cicles (salta)
            BRA finishRSI
            BTFSS INTCON3,3,A +2 cicles (salta)
            BRA finishRSI
            BCF INTCON3,0,A    +1 cicle
copyId:     MOVF PORTC,W,A      +1 cicle
            MOVWF POSTINC0,A    +1 cicle
            LFSR FSR1,csCount  +2 cicles
            MOVLW 3             +1 cicle
copyTime:   MOVFF POSTINC1,POSTINC0
            CPFSEQ FSR1L,A
            BRA copyTime
            } veure 5.1
finishRSI:  RETFIE FAST        +2 cicles
```

**5.1 (1 punt)** Suposeu que hem entrat a la RSI perquè **ha vingut un Flag de INT1 però no de TMR2 (INT1IF=1, TMR2IF=0)**. Sabem que  $F_{osc}=12\text{MHz}$ . Calculeu el temps d'execució de la RSI.

Al costat de cada instrucció hem escrit la duració en cicles d'execució  $T_{cy}$ . A la secció "checkTMR2" no fem salt doncs sabem que  $TMR2IF=0$  per l'enunciat. A la secció "checkINT1" sí saltem als dos BTFSS doncs sabem que  $INT1IF=1$ , i evidentment  $INT1IE$  també ha de ser 1 (si no, no estariem executant la RSI).

Veiem ara el bucle "copyTime" (fa una còpia del unsigned short long csCount a un posició de memòria on apunta FSR0, que no cal saber on és). Primer copiem l'adreça de csCount (0x000) al FSR1. Per tant, el bucle comença amb  $FSR1L=0x00$ .

A les dues primeres iteracions, el CPFSEQ compara el valor 3 amb 0x01 i 0x02 respectivament. Cap dels dos cops fem skip. Per tant els cicles de cada iteració son: +2cicles (MOVFF), +1cicle (CPFSEQ sense skip), +2cicles (BRA). A la tercera iteració, el CPFSEQ sí fa skip (doncs  $FSR1L$  serà 0x03). Per tant els cicles de la 3a iteració son: +2cicles (MOVFF), +2cicles (CPFSEQ amb skip sobre una SingleWord).

Si sumem cicles, tenim:  $1+2+2+2+1+1+1+2+1+2+(2+1+2)+(2+2)+2 = 29$  cicles.

Cada cicle d'execució és  $1 T_{cy} = \frac{4}{12\text{MHz}}$ . Per tant,  $t_{exec} = 29 \cdot \frac{4}{12\text{MHz}} = 9.66 \mu\text{s}$

**5.2 (1 punt)** Si a continuació del codi de dalt afegíssim una RSI de baixa prioritat, escriuríem una directiva `org 0x18`, i tot seguit el codi de la RSI\_low. ¿Per a què afegim les directives `org 0x08` i `org 0x18`?

Són necessàries per indicar que el codi que ve a continuació s'ha de col·locar a partir d'una adreça concreta de memòria de programa (Flash). Quan el PIC intenti executar una RSI High anirà a l'@ 0x08 de la Flash i executarà el que hi hagi a partir d'allà. Quan es cridi a una RSI Low, anirà a l'@ 0x18 de Flash i executarà el que hi hagi.

Amb el codi de l'enunciat, tindríem algun problema en relació amb les directives `org`? Si fos el cas, com ho solucionaríeu?

Sí que tindríem un problema, doncs el codi de la RSI d'alta prioritat és massa llarg. Si comptem quants bytes ocupa, son 22words (44bytes). Si comença des de l'@ 0x08, la primera adreça lliure seria  $8+44=52$  (0x34). Per tant, hauríem escrit codi sobre les posicions reservades per a la RSI de Baixa Prioritat.

La solució seria que a la posició 0x08 hi hagués un salt (e.g. GOTO) a una altra part de memòria de programa on hi hagués espai suficient per posar el codi de la RSI.

```
org 0x000
    GOTO main

org 0x08
    GOTO rsiHigh

org 0x18
    GOTO rsiLow

main:
    ; main code here...

rsiHigh:
    ; RSI High Priority code here

rsiLow:
    ; RSI Low Priority code here
```



Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P1. (2 punts)**1.1 Quants cicles triga a executar-se el següent codi si  $F_{osc}=4\text{MHz}$ :

```
LFSR FSR2, 000 ; 2 cicles
loop
  CLRF INDF2, A ; 1 cicle
  INCF FSR2L, F, A ; 1 cicle
  BTFSC FSR2L, 6, A ; 1 cicle si no salta, 3 si salta una instrucció 2-word
  GOTO final; 2 cicles
  GOTO loop ; 2 cicles
final
```

El codi itera sobre el banc 0 de la memòria de dades fins que el bit 6 de l'FSR2L és 1. En total realitza 64 iteracions.

$LFSR (2c) + 63 * (CLRF(1c) + INCF(1c) + BTFSC \text{ fa skip } (3c) + GOTO (2 \text{ cicles})) + CLRF(1c) + INCF(1c) + BTFSC \text{ no fa skip } (1c) + GOTO (2 \text{ cicles}) = 448 \text{ cicles}$

1.2 Proposa una alternativa que tingui la mateixa funcionalitat però que sigui més eficient en el número de cicles que triga en executar-se. Quant triga la teva proposta?

Es pot millorar l'eficiència del codi canviant GOTOs per BRA (estalvia un cicle al BTFSS quan fa skip), També es pot reorganitzar la seqüència de salts per no fer skip a cada iteració i fer servir POSTINC per incrementar l'FSR2 sense necessitar l'INCF

Una proposta pot ser:

```
LFSR FSR2, 000 ; 2 cicles
loop
  CLRF POSTINC2, A ; 1 cicle
  BTFSS FSR2L, 6, A ; 1 cicle si no salta, 2 cicles si salta
  BRA loop; 2 cicles
final
```

$LFSR (2c) + 63 * (CLRF(1c) + BTFSS \text{ no fa skip } (1c) + BRA (2 \text{ cicles})) + CLRF(1c) + BTFSS \text{ fa skip } (2c) = 257 \text{ cicles}$

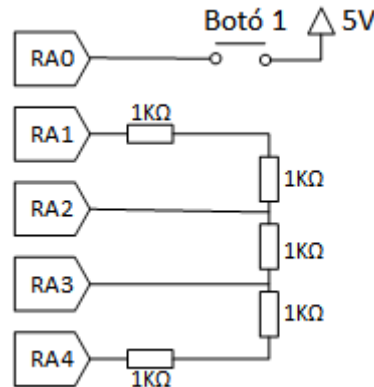
## P2. (2 punts)

Dos alumnes han decidit fer una part d'una pràctica de laboratori per separat i just abans de la classe unir el que han fet. Aquest és el resultat:

Codi

```
MOVLB 0x0F
CLRF ANSELA, B
BSF TRISA, 0, B
BCF TRISA, 1, B
BCF TRISA, 4, B
BSF TRISA, 2, B
BSF TRISA, 3, B
loop
BSF LATA, 1, B
BCF LATA, 4, B
BRA loop
end
```

Esquemàtic



2.1 Si suposem que VDD=5V, quin serà l'estat dels següents bits:

Tenim una caiguda de tensió entre RA1 i RA4 de 5V. Com que totes les resistències tenen el mateix valor, cau el mateix voltatge a cada resistència: 1V

Bit	Estat (i justificació)
PORTAbits.RA0 amb el botó 1 premut	RA0=1. Els 5V arribaran al pin.
PORTAbits.RA0 amb el botó 1 sense polsar	RA0= indeterminat. El pin queda a l'aire al no disposar d'una R de pull-down
PORTAbits.RA1	RA1=1. Pin configurat com a output amb LAT configurat a 1.
PORTAbits.RA2	RA2=1. Al pin arriben 3V que és més gran que VIH
PORTAbits.RA3	RA3=1. Al pin arriben 2V que és més gran que VIH. Si suposem un VOH menor a 5V pot ser indeterminat.
PORTAbits.RA4	RA4=0. Pin configurat com a output amb LAT configurat a 0.

2.2 Circula corrent pel pin RA1? Quants amperes? Això suposa algun problema?

Sí que circula corrent entre RA1 i RA4.

Si suposem que RA1=5V i RA4=0V:

$I = V/R = 5V/5000\Omega = 1mA$

Com que el límit dels pins és de 20mA no suposa cap problema.

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P3. (3 punts)**

Mireu aquest codi d'una funció pel 18F45K22

```
Calcula:    MOVLB 3                ; Supposem Bank 3 disponible
            CLRF 00, B
            MOVWF 01, B
            MOVLW 08
            MOVWF 02, B
            MOVF 01, W, B

loop:       ANDLW 01
            BZ  no
            INCF 00, F, B
no:         RRNCF 01, F, B        ;RR Rotate Right, com shiftar un bit
            MOVF 01, W, B
            DECFSZ 02, F, B
            BRA loop

            MOVF 00, W, B
            RETURN
```

Si cridem la funció d'aquesta manera

```
MOVLW 36h
CALL Calcula
```

3.1 Quin valor tindrem al WREG quan retorni? (0.5p)

El WREG valdrà 4, que és el nombre de 1's que tenia el paràmetre i que s'ha anat calculant a l'adreça 300h.

3.2 Què tindrem a les adreces 300h, 301h i 302h després de la seva execució? (0.75p)

300h tindrà també 4, 301h després de la rotació completa tindrà el valor original del paràmetre, 36h i 302h és el comptador que s'ha anat decrementant fins quedar a 0.

3.3 Quants Bytes ocupa a memòria de programa la funció "Calcula" ? (0.5p)

La funció va des de l'etiqueta calcula fins al return (sinó no seria una funció) i consta de 15 instruccions codificades amb 1 word, per tant 30 Bytes en total.

El codi de cridar a una funció no es compta com a codi de funció. Voldria dir que una funció que es crida més vegades ocupa més que una que es crida poques vegades?

3.4 Quina funcionalitat té el codi englobat a “Calcula” ? (0.5p)

El codi compta el nombre de bits a 1 que té el valor rebut a WREG. Ho fa rotant-lo cap a la dreta 8 vegades i mirant si el bit que queda a la posició 0 és o no un 1. Si ho és incrementa el comptador de la posició 300h.

3.5 El temps d'execució és independent del contingut de WREG a la crida? En cas de que no ho sigui, calcula el cas millor i el cas pitjor. (0.75p)

El temps és independent del valor.

El nombre d'iteracions és constant, 8, perquè fem un recorregut sobre 8 elements.

Dins de cada iteració tenim una estructura alternativa, amb la instrucció BZ després de la AND.

Tant si salta (BZ branch farà dos cicles) com si no salta (BZ farà un cicle i INCF un cicle) el codi triga el mateix, per tant no hi ha cap dependència del valor inicial a WREG.

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P4. (3 punts)**

4.1 (1,5 punts) Disposeu del següent codi que usa el Timer0 per a generar una interrupció periòdica cada cert temps. Dins de la RSI fa que un pin canviï periòdicament de nivell lògic (*toggle*). Completa els 2 espais emmarcats amb el codi necessari per a que funcioni bé el programa.

```
void interrupt low_priority lowRSI()
{
    if (TMR0IF==1 && TMR0IE==1)
    {
        TMR0L=131;

        TMR0IF=0; // cal netejar flag, per tal que no es
                  // torni a cridar RSI quan sortim

        // a cada RSI faig un canvi de nivell
        // ... al pin de Output:
        LATAbits.LATA0=!LATAbits.LATA0;
    }
}

void main(void)
{
    initOutputPinRA0(); // doneu per fet que aquesta
                        // subrutina esta ben programada

    // inicialitzacio Timer0
    T0CONbits.TMR0ON=0;
    T0CONbits.T08BIT=1;
    T0CONbits.T0CS=0;
    T0CONbits.PSA=0;
    T0CONbits.T0PS=4;
    TMR0L=131; // ajustarem periodicament aquest valor inicial de comptatge

    //inicialitzacio interrupcions

    TMR0IP=0; // establim prioritat a Low (veure capçalera RSI)
    TMR0IF=0; // netegem possible Flag que hi hagués previ (poc probable,
              // doncs s'havien executat molt poques línies de codi i
              // segurament no havia donat temps a fer cap overflow encara)
    TMR0IE=1; // habilitem font d'interrupció TMR0

    IPEN=1; // habilitem sistema de nivells de prioritat a les interrupcions
    GIEL=1; // habilitem el Global Interrupt de les Low
              // (la nostra interrupció és Low)
    GIEH=1; // habilitem el Global Interrupt de les High. Cal habilitar-lo
              // també, si no no es cridaria a cap RSI (ni High ni Low)

    //start Timer0
    T0CONbits.TMR0ON=1;

    while (1);
}
```

Ens diu que el comptatge s'inicia periòdicament des d'un cert valor: aquesta és la estratègia per tal que la interrupció periòdica del Timer 0 arribi cada cert número desitjat de ticks.

Per tant, després de cada overflow, cal restablir dins la RSI el valor inicial de 131.

4.2 (1 punt) Executem el codi de la pregunta P4 en un micro que funciona amb  $F_{osc}=16\text{ MHz}$ . ¿Cada quant temps s'executa la interrupció de Timer0?

La fórmula per calcular el període entre RSI és:

$$T_{RSI} = T_{CS} \cdot PRE \cdot nTicks$$

$T_{CS}$  és el període del Clock Source escollit. En el cas del codi anterior es posa el bit T0CS a 0, per tant selecciona usar el clock d'instruccions Fcy. Sabem que  $F_{osc}=16\text{MHz}$ , per tant  $F_{cy}=16\text{MHz}/4 = 4\text{MHz}$ .

El bit PSA a 0 indica que s'utilitza Prescaler. El valor del Prescaler surt del número gravat al camp T0PS=4. El valor 4 (100 en binari), segons la descripció del SFR, correspon al Prescaler 1:32 (és a dir: divideix freqüència d'entrada entre 32, o bé multiplica període per 32).

Per tant:

$$T_{RSI} = T_{CY} \cdot PRE \cdot nTicks = \frac{1}{4\text{MHz}} \cdot 32 \cdot nTicks$$

Per saber el número de Ticks que transcorre entre interrupcions, cal saber el valor inicial al qual restablim el Timer a cada overflow: 131 (el codi ens dona aquest valor).

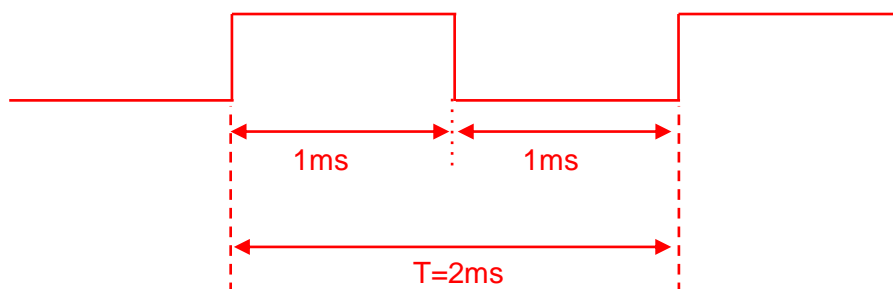
Com que el timer està configurat a 8 bits (T08BIT és 1), vol dir que es comptarà sempre de 131 fins al overflow (256). Per tant, comptem un número de ticks =  $256-131 = 125$ .

$$T_{RSI} = \frac{1}{4\text{MHz}} \cdot 32 \cdot 125 = 1\text{ms}$$

4.3 (0,5 punts) ¿Quina és la freqüència de la senyal que surt pel pin RA0?

Sabem que la RSI s'executa cada 1ms. Dins del codi hi ha una instrucció per fer un Toggle del pin RA0. Per tant, cada 1ms la senyal de RA0 canvia de valor.

Veiem-ho gràficament:



Si la senyal té un període de  $T=2\text{ms}$ , llavors la freqüència és de  $\frac{1}{2\text{ms}} = 500\text{Hz}$

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

Tota resposta sense justificar es considerarà nul·la !

**P1. (1,5 punts)**

1.1 (1 punt) Calcula quant triga el següent codi a executar-se si es fa servir una Fosc=8MHz

```

mystery_routine
    valor1 equ 000
    valor2 equ 0x1FF

    LFSR FSR0, valor1 ; 2 cicles
    LFSR FSR1, valor2 ; 2 cicles
acaba
    MOVFF POSTINC0, POSTDEC1 ; 2 cicles
    BTFSS FSR0H, 0, A ; 1 cicle si no salta, 2 cicles si salta (3 si la següent
instrucció fos de double word)
    BRA acaba ; 2 cicles

```

Un anàlisi de l'execució es pot trobar en la següent taula

Inst/Cicle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...	1280	1281	1282	1283
LFSR FSR0	F	E	E																
LFSR FSR1			F	E	E														
MOVFF					F	E	E												
BTFSS							F	E	No salta										
BRA								F	E	E									
(INST)									F	-									
MOVFF										F	E	E							
BTFSS												F	E	No salta					
BRA													F	E	E				
[...]														F	-	253 iteracions més fins que FSR0H[bit0]=1			
MOVFF															F	E	E		
BTFSS																	F	E	E SALTA
BRA																		F	-

De forma alternativa podem comptar:

FSR(2)+FSR(2)+255\*(MOVFF(2)+BTFSS(1)+BRA(2))+MOVFF(2)+BTFSS(2)=1283 cicles

1283 cicles \*(4/8MHz)=641,5us

1.2 (0,5 punts) Indica, de forma breu, què fa aquest codi i quant ocupa a memòria de programa  
Aquest codi el que fa és copiar el Bank0 de memòria de dades al Bank1, però de forma inversa:  
[000] -> [1FF]  
[001] -> [1FE]  
[...]  
[0FF] -> [100]

En total ocupa a memòria de programa 8 words:  
LFSR(x2) 2w, MOVFF 2w, BTFSS 1w, BRA 1w

## P2. (2 punts)

Indica si són certes (C) o falses (F) les afirmacions següents (cada encert suma 0.2 / cada error resta 0.2)

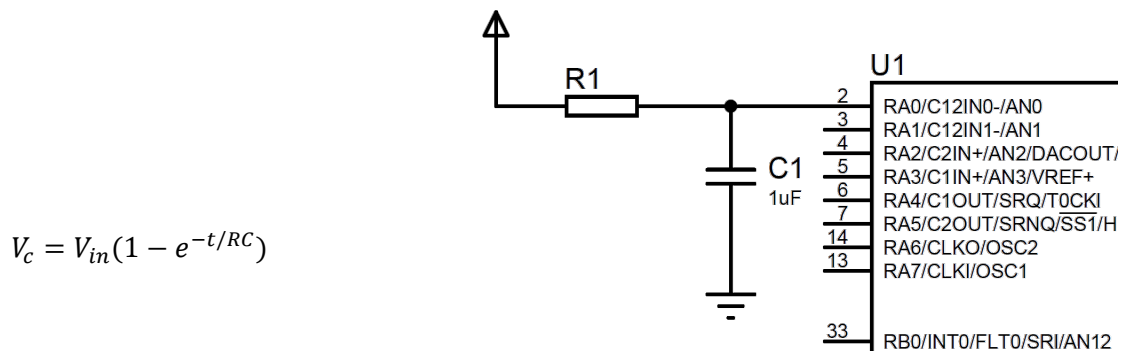
- ☐ F Utilitzem la instrucció RETFIE FAST per a sortir de les RSI de baixa prioritat. Les d'alta prioritat han d'afegir un codi en assembler per a guardar/restaurar el WREG, STATUS i BSR en variables de memòria RAM, i sortir usant RETFIE.
- ☐ F El flag de petició d'interrupció IF no s'activa si aquesta està deshabilitada mitjançant el bit d'enable IE.
- ☐ C L'avantatge d'una arquitectura Harvard enfront d'una Von Neumann és que permet la concurrència en l'accés a dades i instruccions.
- ☐ C Totes les instruccions del PIC18F45k22 ocupen 16 bits, excepte GOTO, LFSR, MOVFF i CALL.
- ☐ C Accedint a la memòria de dades, si ho fem en mode banked podem accedir a més posicions de memòria que si ho fem en mode Access Bank.
- ☐ C "CLRF 00, A" i "CLRF 00, B" accedeixen a la mateixa posició de memòria si BSR=0.
- ☐ F El registre ANSELA no es troba a l'Access Bank. Per accedir-hi ho fem en mode banked i BSR=0.
- ☐ C El byte menys significatiu d'una instrucció, s'emmagatzema sempre en una adreça parell de la memòria de programa.
- ☐ C Usant un clock de CPU de 4MHz, el temps d'execució (en microsegons) d'una instrucció que provoca un salt de tipus skip i precedeix a una instrucció 'double word' serà de 3µs.
- ☐ F Podem triar la prioritat de la interrupció externa INT0 fent servir el bit INT0IP.



Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P3. (1,5 punts)**

Tenim connectat al PIN RA0 un circuit resistència-condensador que es carregarà segons l'equació de càrrega del condensador vista al laboratori



Suposant que  $V_{in}$  són 5V i que el condensador estigui descarregat a temps 0, quina  $R1$  triaries si volguessis assegurar que l'entrada es troba a 0 lògic com a mínim 100 milisegons.

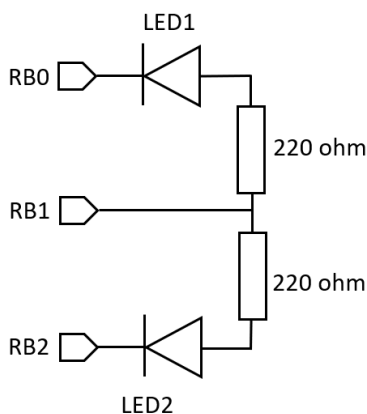
Sabem que el pin es llegirà com a 0 sempre que el valor a l'entrada sigui més petit que  $V_{IL\ MAX}$ . A les condicions d' $V_{IN}=5V$ ,  $V_{IL\ MAX}=0,8V$ .

$$0,8 = 5 (1 - e^{-0,1/R*1u})$$

Aïllant  $R$  trobem que la resistència ha de ser més gran que 573,5KΩ

**P4. (1,5 punts)**

Escriu el codi necessari en ensamblador del PIC18 per aconseguir encendre el LED1 i mantenir el LED2 apagat.



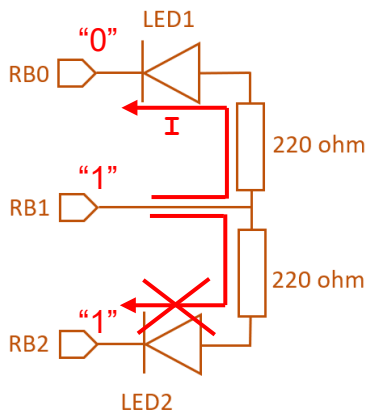
Per tal d'encendre el LED1 haurem de posar més tensió a RB1 que a RB0: per tant a RB1 posarem "1 lògic" i a RB0 posarem "0 lògic". Per tal de posar un valor digital a un pin, l'hem de configurar com a Output!

Per mantenir el LED2 apagat, tenim dues possibilitats:

- 1) podem fixar el valor del pin RB2 a "1 lògic", configurant el pin a Output. D'aquesta manera no hi ha diferència de potencial entre RB1 i RB2.
- 2) Podem configurar el pin RB2 com a Input. D'aquesta manera el micro no hi estableix cap voltatge, i el pin no interacciona amb el circuit connectat (pin en alta impedància)

El següent codi mostra una possible solució fent que el RB2 sigui un Output que tregui "1 lògic". Hem de recordar que per treballar amb els pins com a digitals, cal ajustar els bits corresponents del registre ANSELx (bits a 0 per a que els pins siguin Digitals). Els registres ANSELx no estan al Access Bank.

La solució mostra diferents formes d'establir el valor als bits d'un registre.



```

MOVLB 0x0F           ; BSR = 0x0F
CLRF ANSELB, B       ; posem a 0 tots els bits de ANSELB (pins digitals)
                        ; B= banked access (posem BSR=0x0F abans)

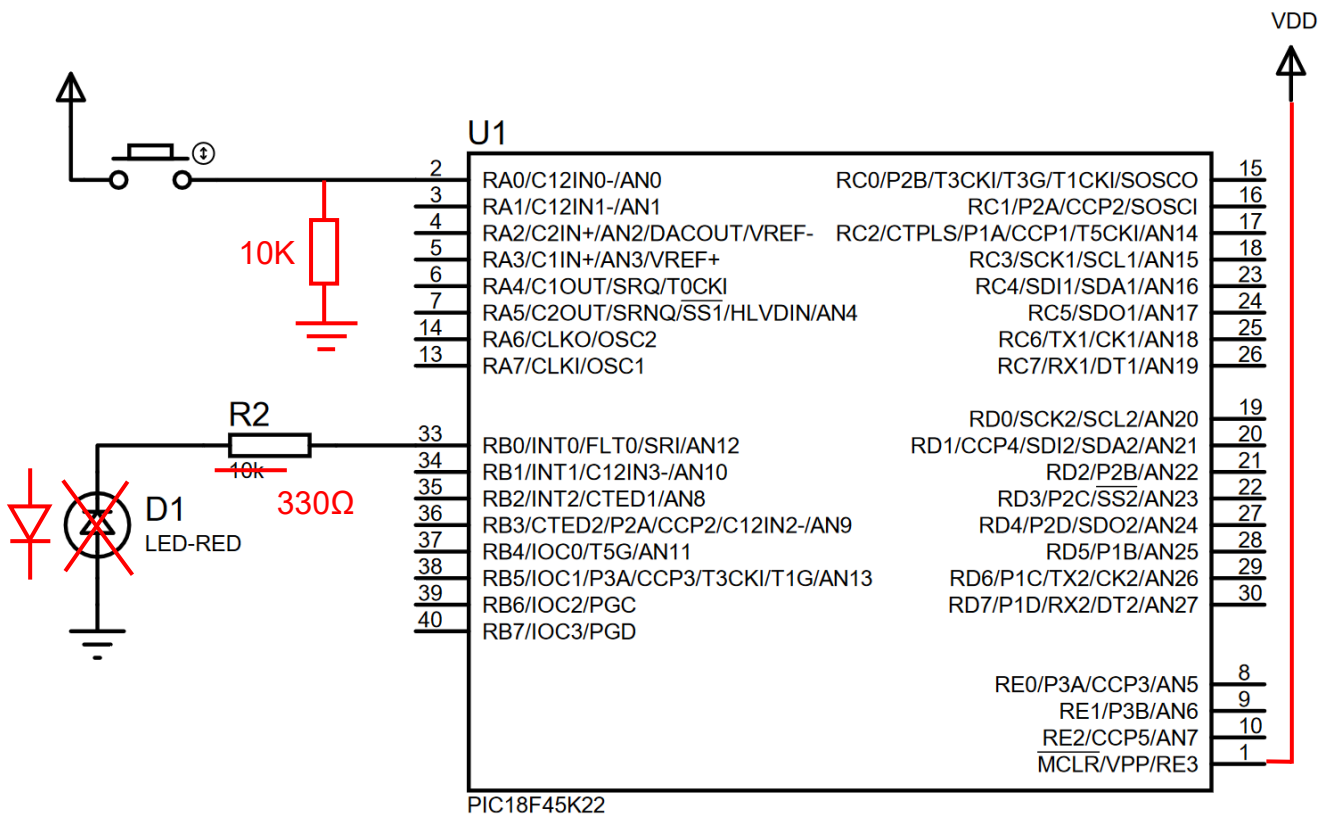
MOVLW 0x00           ; WREG = 0
MOVWF TRISB, A       ; posem a 0 tots els bits de TRISB (pins a Output)

BCF LATB, 0, A       ; LATB0 = 0
BSF LATB, 1, A       ; LATB1 = 1
BSF LATB, 2, A       ; LATB2 = 1

```

### P5. (1,5 punts)

Un estudiant ha entregat un esquemàtic d'un projecte Proteus on vol encendre un led en funció de l'estat d'un botó. Si el botó està premut el led s'ha d'encendre, si el botó no està premut el led ha d'estar apagat. Tot i copiar el codi de github, el projecte no funciona. Indica tots els errors que trobes en l'esquemàtic



1) Cal connectar el símbol de VDD al pin 1 (/MCLR) per tal d'evitar que el micro es resetegi aleatòriament. La funció de reset (MCLR= Master Clear) funciona amb lògica negada, per tant hem de posar un "1 lògic" per a que no es reiniciï.

2) Al circuit del polsador cal posar una resistència de pull-down, per tal que es llegeixi un 0 quan el botó no està premut. Altrament, el pin quedaria a l'aire.

3) Per tal que s'encengui el led, l'ànode ha d'estar a una tensió més gran que el càtode. Al diagrama, l'ànode està a Ground (0V), per tant serà impossible col·locar al càtode una tensió més petita. Una possible solució és girar el Led (tal com es veu a la solució proposada al dibuix).

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !**

Una altra solució seria connectar l'ànode a VDD en comptes de a 0V: això requeriria encendre el led amb lògica negada, és a dir el pin RB0 hauria de treure un "0" per encendre'l, i un "1" per apagar-lo.

4) El valor de 10K (que proposa Proteus per defecte) és massa gran i provoca que la corrent que circula sigui massa petita com per a encendre el led de la simulació. Valors adequats poden ser 470Ω, 330Ω, o propers.

**P6. (2 punts)**

Volem programar una sèrie d'interrupcions i hem trobat el següent codi a una coneguda web d'ajuda per a estudiants. Al copiar-ho han quedat espais en blanc. Completa els espais emmarcats amb el codi necessari per a que funcioni bé el programa sense modificar el codi que sí que s'ha copiat bé.

```
void interrupt low_priority lowRSI()
{
    if (TMR0IF==1 && TMR0IE==1)
    {
        tractament_T0(); // doneu per fet que aquesta
                        // subrutina esta ben programada
        TMR0IF=0;
    }
}
void interrupt highRSI()
{
    if (ADIF==1 && ADIE==1)
    {
        tractament_AD(); // doneu per fet que aquesta
                        // subrutina esta ben programada
        ADIF=0;
    }
}

void main(void)
{
    iniTimer(); // doneu per fet que aquestes
    iniAD();    // subrutines estan ben programades
```

```
//inicialitzacio interrupcions
```

```
// Sempre cal fer el següent: habilitar cada font d'interrupció involucrada, configurar prioritats...
// ... i activar els enables globals necessaris.
// A la solució següent veureu diferents formes d'accedir als bits dels registres en C.
```

```
INTCONbits.TMR0IF=0;    // opcional: podem netejar el Flag per si hagués succeït interrupció prèvia
INTCON2bits.TMR0IP=0;    // cal posar la prioritat a 0 (low priority), doncs el TMR0 es tracta a lowRSI
INTCONbits.TMR0IE=1;    // habilitem font d'interrupció TMR0
```

```
ADIF=0;    // opcional: podem netejar el Flag per si hagués succeït interrupció prèvia
ADIP=1;    // cal posar la prioritat a 1 (high priority), doncs el AD es tracta a highRSI
ADIE=1;    // habilitem font d'interrupció AD
```

```
RCON = RCON | 0x80;    // IPEN=1. Habilitem nivells de prioritat, doncs tenim les 2 RSI programades
INTCON = INTCON | 0x80; // GIEH=1. Global Enable de les interrupcions d'alta prioritat
INTCON = INTCON | 0x40; // GIEL=1. Global Enable de les interrupcions de baixa prioritat
```

```
while (1);
```

```
}
```

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P1. (2 punts)**1.1 Quants cicles triga a executar-se el següent codi si  $F_{osc}=4\text{MHz}$ :

```
LFSR FSR2, 000
loop
  CLRF INDF2, A
  INCF FSR2L, F, A
  BTFSC FSR2L, 6, A
  GOTO final
  GOTO loop
final
```

1.2 Proposa una alternativa que tingui la mateixa funcionalitat però que sigui més eficient en el número de cicles que triga en executar-se. Quant triga la teva proposta?

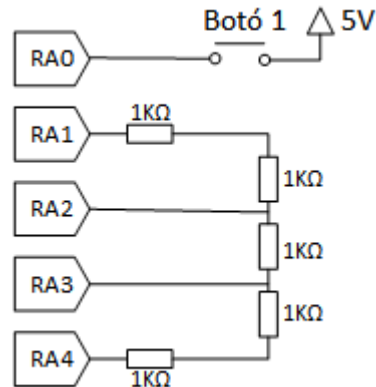
## P2. (2 punts)

Dos alumnes han decidit fer una part d'una pràctica de laboratori per separat i just abans de la classe unir el que han fet. Aquest és el resultat:

Codi

```
MOVLB 0x0F
CLRF ANSELA, B
BSF TRISA, 0, B
BCF TRISA, 1, B
BCF TRISA, 4, B
BSF TRISA, 2, B
BSF TRISA, 3, B
loop
BSF LATA, 1, B
BCF LATA, 4, B
BRA loop
end
```

Esquemàtic



2.1 Si suposem que VDD=5V, quin serà l'estat dels següents bits:

Bit	Estat (i justificació)
PORTAbits.RA0 amb el botó 1 premut	
PORTAbits.RA0 amb el botó 1 sense pulsar	
PORTAbits.RA1	
PORTAbits.RA2	
PORTAbits.RA3	
PORTAbits.RA4	

2.2 Circula corrent pel pin RA1? Quants amperes? Això suposa algun problema?

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P3. (3 punts)**

Mireu aquest codi d'una funció pel 18F45K22

```
Calcula:    MOVLB 3                ; Supposem Bank 3 disponible
            CLRF 00, B
            MOVWF 01, B
            MOVLW 08
            MOVWF 02, B
            MOVF 01, W, B

loop:       ANDLW 01
            BZ  no
no:          INCF 00, F, B
            RRNCF 01, F, B        ;RR Rotate Right, com shiftar un bit
            MOVF 01, W, B
            DECFSZ 02, F, B
            BRA loop

            MOVF 00, W, B
            RETURN
```

Si cridem la funció d'aquesta manera

```
MOVLW 36h
CALL Calcula
```

3.1 Quin valor tindrem al WREG quan retorni? (0.5p)

3.2 Què tindrem a les adreces 300h, 301h i 302h després de la seva execució? (0.75p)

3.3 Quants Bytes ocupa a memòria de programa la funció "Calcula" ? (0.5p)

3.4 Quina funcionalitat té el codi englobat a “Calcula” ? (0.5p)

3.5 El temps d'execució és independent del contingut de WREG a la crida? En cas de que no ho sigui, calcula el cas millor i el cas pitjor. (0.75p)

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P4. (3 punts)**

4.1 (1,5 punts) Disposeu del següent codi que usa el Timer0 per a generar una interrupció periòdica cada cert temps. Dins de la RSI fa que un pin canviï periòdicament de nivell lògic (*toggle*). Completa els 2 espais emmarcats amb el codi necessari per a que funcioni bé el programa.

```
void interrupt low_priority lowRSI()
{
    if (TMR0IF==1 && TMR0IE==1)
    {
        

        // a cada RSI faig un canvi de nivell
        // ... al pin de Output:
        LATAbits.LATA0=!LATAbits.LATA0;
    }
}

void main(void)
{
    initOutputPinRA0(); // doneu per fet que aquesta
                        // subrutina esta ben programada

    // inicialitzacio Timer0
    T0CONbits.TMR0ON=0;
    T0CONbits.T08BIT=1;
    T0CONbits.T0CS=0;
    T0CONbits.PSA=0;
    T0CONbits.T0PS=4;
    TMR0L=131; // ajustarem periodicament aquest valor inicial de comptatge

    //inicialitzacio interrupcions
    

    //start Timer0
    T0CONbits.TMR0ON=1;

    while (1);
}
```



4.2 (1 punt) Executem el codi de la pregunta P4 en un micro que funciona amb  $F_{osc}=16\text{ MHz}$ . ¿Cada quant temps s'executa la interrupció de Timer0?

4.3 (0,5 punts) ¿Quina és la freqüència de la senyal que surt pel pin RA0?

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !**

**P1. (2,5 punts)**

Tenim la següent rutina que calcula el mínim d'un vector de 256 elements guardat al banc 2 de memòria del PIC deixant el resultat al WREG.

```
r_min:    vector_ad    EQU 200h
          SETF        WREG,A
          LFSR        FSR0, vector_ad
next:     CPFSGT      INDF0,A      ; els modes indirectes van amb Access Bank o es pot
          MOVF        INDF0,W,A   ; deixar en blanc perquè és l'opció per defecte.
          INCFSZ      FSR0L,F,A
          BRA         next
          RETURN
```

1.1 Calcula el temps d'execució de la rutina desde que se li fa la crida (CALL r\_min, fins que torna al principal). Fosc=12Mhz.

A la taula veiem l'anàlisi de l'execució des del CALL fins que es torna de la funció rmin.

Instr / Cicle		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...	1281	1282	1283	1284	1285	1286
(pre CALL)	E																..						
CALL rmin	F	E															..						
(post CALL)		F	-														..						
SETF			F	E													..						
LFSR				F	F	E											..						
CPFSGT						F	E										..						
MOVF o SKIP							F	?									..						
INCF SZ								F	E	FSR0L 0 a 1							..						
BRA									F	E							..						
(return)										F	-						..						
CPFSGT											F	E					..						
MOVF o SKIP												F	?				..						
INCF SZ													F	E	FSR0L 1 a 2								
BRA														F	E		..						
(return)															F	-	..						
.. iteracions..																	..						
CPFSGT																		E					
MOVF o SKIP																		F	E				
INCF SZ																			F	E	FSR0L FF a 0		
BRA																				F	-		
RETURN																					F	E	
(post RETURN)																						F	-
(post CALL)																							F

Hem comptat 1286 cicles d'instrucció des del CALL fins que es retorna (s'accepten interpretacions + - 1 o 2).

$$t = 1286 \text{ cicles} \times (4 \text{ tics rellotge} / 1 \text{ cicle}) \times (1 \text{ tic rellotge} / 12 \times 10^6 \text{ Hz}) = 428.6 \text{ us (microsegons)}$$

Alternativament podem comptar:

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !**

$\text{sef}(1) + \text{lfsr}(2) + 255 (\text{cpfsgt}(1 \text{ o } 2) + \text{movf}(1 \text{ o } 0) + \text{incfsz}(1) + \text{bra}(2)) + 1 (\text{cpfsgt}(1 \text{ o } 2) + \text{movf}(1 \text{ o } 0) + \text{incfsz}(2) + \text{return}(2)) = 3 + 255(5) + 1(6) = 1284$ , amb els 2 d'entrar a la funció, 1286.

1.2 El temps d'execució és independent del contingut del vector de 256 elements? Si no ho és, indica el temps d'execució en el cas millor i pitjor.

Si, és independent.

Suposem que el nombre de la posició apuntada per INDF0 és menor o igual que WREG.

Instrucció / cicle	x	x+1	x+2	x+3
CPFSGT INDF0,A (compara W amb on apunta INDF0, no fa skip)	F	E		
MOVF INDF0,W,A (actualitza W amb el nou mínim)		F	E	
INCFSZ FSR0L,F,A (incrementa el punter 0 al següent)			F	E
...				F

Suposem ara que el nombre de la posició apuntada per INDF0 és major que WREG.

Instrucció / cicle	x	x+1	x+2	x+3
CPFSGT INDF0,A (compara W amb on apunta INDF0, farà SKIP)	F	E		
MOVF INDF0,W,A (se li havia fet FETCH, no fa EXE per l'SKIP)		F	-	
INCFSZ FSR0L,F,A (incrementa el punter 0 al següent)			F	E
...				F

1.3 Quants Bytes ocuparà la rutina a la memòria de programa?

r_min:	vector_ad	EQU 200h	no ocupa codi, defineix una etiqueta
	SETF	WREG,A	1 instrucció single word (2B)
	LFSR	FSR0, vector_ad	1 instrucció double word (4B)
next:	CPFSGT	INDF0,A	1 instrucció single word (2B)
	MOVF	INDF0,W,A	1 instrucció single word (2B)
	INCFSZ	FSR0L,F,A	1 instrucció single word (2B)
	BRA	next	1 instrucció single word (2B)
	RETURN		1 instrucció single word (2B)

En total la rutina ocupa 16 Bytes a memòria de programa.

**P2. (1 punt)**

Indica maneres de, amb un màxim de dues instruccions i dos cicles, posar a zero el contingut d'un registre REG (omple la taula, fins 8). Exemple:

Suposarem que REG està accessible al ACCESS BANK (,A)

MOVLW 0 MOVWF REG	CLRF REG,A	COMF REG,W,A ANDWF REG,F,A	SETF REG,A INCF REG,F,A
CLRF WREG MOVWF REG,A (qualsevol altre que posi WREG a 0)	MOVLW 0 ANDWF REG,F,A (qualsevol altre que posi WREG a 0)	MOVF REG,W,A SUBWF REG,F,A (també XORWF, compte amb els borrow!)	SETF REG,A COMF REG,F,A (COMF nega bit a bit)

Entre d'altres solucions.

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P3. (1 punt)**

Indica quin serà el valor del registre STATUS després d'executar les següents instruccions. Si un bit del registre no es veu afectat indica-ho amb una 'x'

Instruccions	STATUS				
	N	OV	Z	DC	C
MOVLW 0 MOVWF 0x00, A MULWF 0x00, F, A Cap instrucció modifica l'STATUS	X	X	X	X	X
CLRF 0x00, B [0xB00]=0 INCF 0x00, B [0xB00]=1 INCF modifica tots els bits	0	0	0	0	0
MOVLB 3 CLRF 0x00, B [0x300]=0 DECF 0x00, B [0x300]=FFh DECF modifica tots els bits	1	0	0	0	0
MOVLW 0xE0 WREG=E0h MOVWF 0x00, A [0x000]=E0h RLNCF 0x00, F, A [0x000]=C1h RLNCF modifica N i Z	1	X	0	X	X

**REGISTER 5-2: STATUS REGISTER**

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC <sup>(1)</sup>	C <sup>(2)</sup>
bit 7							
							bit 0

**Legend:**

R = Readable bit  
-n = Value at POR

W = Writable bit  
'1' = Bit is set

U = Unimplemented bit, read as '0'  
'0' = Bit is cleared

x = Bit is unknown

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **N:** Negative bit  
This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative (ALU MSB = 1).  
1 = Result was negative  
0 = Result was positive

bit 3 **OV:** Overflow bit  
This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude which causes the sign bit (bit 7 of the result) to change state.  
1 = Overflow occurred for signed arithmetic (in this arithmetic operation)  
0 = No overflow occurred

bit 2 **Z:** Zero bit  
1 = The result of an arithmetic or logic operation is zero  
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit Carry/Borrow bit<sup>(1)</sup>  
For ADDWF, ADDLW, SUBLW and SUBWF instructions:  
1 = A carry-out from the 4th low-order bit of the result occurred  
0 = No carry-out from the 4th low-order bit of the result

bit 0 **C:** Carry/Borrow bit<sup>(2)</sup>  
For ADDWF, ADDLW, SUBLW and SUBWF instructions:  
1 = A carry-out from the Most Significant bit of the result occurred  
0 = No carry-out from the Most Significant bit of the result occurred

#### P4. (1 punts)

En un fòrum d'internet, un usuari assegura que no hi ha problemes en connectar els pins I/O d'un PIC18F45K22 (VDD=5V) amb els pins I/O d'una Raspberry PI (VDD=3V3). Estàs d'acord amb aquesta afirmació?

Raspebrry PI GPIO input/output pin electrical characteristics	
Output low voltage $V_{OL}$	<0,4V
Output high voltage $V_{OH}$	>2,4V
Input low voltage $V_{IL}$	<0,8V
Input high voltage $V_{IH}$	>2V

Hem de comprovar els 4 casos possibles:

1 - Enviar un 0. Raspberry com a output i PIC com a input:

La senyal de la raspberry estarà entre 0V i 0,4V. El PIC llegirà un 0 entre 0V i 0,8V. És correcte i tenim un marge de soroll de 0,4V

2 - Enviar un 0. PIC com a output i raspberry com a input:

La senyal del PIC estarà entre 0V i 0,6V. La raspberry llegirà un 0 entre 0V i 0,8V. És correcte i tenim un marge de soroll de 0,2V

3 - Enviar un 1. Raspberry com a output i PIC com a input:

La senyal de la raspberry estarà entre 2,4V i 3,3V. El PIC llegirà un 1 a partir dels 2V. És correcte i tenim un marge de soroll de 0,4V

4 - Enviar un 1. PIC com a output i raspberry com a input:

La senyal del PIC estarà entre 4,3V i 5V. La raspberry llegirà un 1 a partir de 2V. Però el pin de la raspberry pi és tolerant a un input més gran que el seu VDD? L'enunciat no ens dóna informació sobre aquest punt. En cas afirmatiu, no hi hauria problema i l'usuari tindria raó. Si no fos així, podem malmetre els GPIO de la raspberry.

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P5. (2 punts)**

Tenim la següent rutina que gestiona les interrupcions d'un PIC18F45K22 amb una Fosc=4MHZ

```
org 0x0008
    BTFSS INTCON3, INT1IE, A
    BRA exit
    BTFSS INTCON3, INT1IF, A
    BRA exit
    INCF 0x00, A
    BCF INTCON3, INT1IF, A
exit: RETFIE FAST
```

2.1 Quina és la freqüència màxima a la que podem rebre interrupcions sense perdre cap al pin RB1 (INT1)? Supposeu que tot està ben inicialitzat i que el micro no té configurada cap altra font d'interrupció. (Fosc=4MHz)

Perdrem interrupcions si aquestes arriben més ràpid del que les podem tractar:

3 o 4 cicles de latència + 8 cicles que triga l'execució de l'RSI (sabem que no hi ha cap altra font d'interrupció i si s'executa és degut a la INT1). En total 11 o 12 cicles a 4MHz.

Cas millor:  $11 \text{ cicles} * 4/4\text{M} = 0,000011\text{s} \rightarrow 90909\text{Hz}$

Cas pitjor:  $12 \text{ cicles} * 4/4\text{M} = 0,000012\text{s} \rightarrow 83333\text{Hz}$

2.2 Quines modificacions trobaríem al codi de l'RSI anterior si aquesta fos de baixa prioritat?

Si la interrupció és de baixa prioritat la trobarem l'RSI a l'espai reservat per a la rutina de servei a la interrupció de baixa prioritat: 0x0018

Al ser de baixa prioritat no es faran servir els shadows registres llavors el codi ha de guardar i restaurar l'estat, per exemple ho pot fer amb

```
MOVFF WREG, WREG_TMP
MOVFF BSR, BSR_TMP
MOVFF STATUS, STATUS_TMP
...
MOVFF WREG_TMP, WREG
MOVFF BSR_TMP, BSR
MOVFF STATUS_TMP, STATUS
```

I al no fer servir els shadows register, retornarem amb un RETFIE normal, no FAST



### 2.3 Quins bits configuraries, i amb quin valor, per a activar la INT1 sense prioritats?

És una interrupció de core. Necessitem configurar:

ANSELB[1] = 0 (Digital)

TRISB[1] = 1 (input)

INT1EDG = x (configurem la detecció de flanc ascendent o descendent)

INT1IF = 0

INT1IE = 1

GIE = 1

### 2.4 Quins bits configuraries, i amb quin valor, per a activar la INT1 com a baixa prioritat?

Necessitem configurar:

ANSELB[1] = 0 (Digital)

TRISB[1] = 1 (input)

INT1EDG = x (configurem la detecció de flanc ascendent o descendent)

INT1IP = 0 (low priority)

IPEN = 1 (activem les prioritats)

INT1IF = 0

INT1IE = 1

GIEL = 1 (habilitem les interrupts de baixa prioritat)

GIE = 1

Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !****P6. (1 punt)**

Indica si són certes (C) o falses (F) les afirmacions següents (cada encert suma 0.2 / cada error resta 0.2)

☐ F Utilitzem la instrucció RETFIE FAST per a sortir de les RSI de baixa prioritat. Les d'alta prioritat han d'afegir un codi en assembler per a guardar/restaurar el WREG, STATUS i BSR en variables de memòria RAM, i sortir usant RETFIE.

És exactament el contrari. Sortim de les RSI d'alta prioritat amb un RETFIE FAST (així es recupera el context des dels Shadow Registers). I són les de baixa prioritat les que han d'afegir codi extra i sortir amb RETFIE.

☐ C El PIC té una pila de 31 posicions a on es guarden adreces de memòria de programa, per a que es pugui retornar de crides a subrutines del programa o d'atenció a interrupcions.

Cada cop que fem un CALL o RCALL, o bé es crida a una RSI, es guarda en aquesta pila l'adreça de retorn (valor actual del PC). Veure capítol 5.1.2 Return Address Stack al datasheet.

☐ F En un pin configurat com a Output, el sentit de la corrent elèctrica mai pot ser de l'exterior cap a l'interior del pin

El fet de configurar un pin com a Output indica que controlem el valor lògic (i per tant el voltatge) que hi ha en el pin. Una cosa diferent és el flux de corrent elèctrica que s'hi pugui generar en funció del circuit que connectem al pin. Fixeu-vos, com a exemple, en un circuit R-LED que activem per "0". La corrent entrarà de l'exterior cap a l'interior del pin quan encenguem el LED.

☐ C Per a executar una instrucció "MOVFF dada1, dada2" no cal haver especificat abans amb el BSR el Banc de memòria on estan les dades.

El BSR és necessari per especificar el codi de Bank (4 bits), quan l'adreça que es codifica a les instruccions és de 8 bits. Però el MOVFF té codificades a la pròpia instrucció les adreces completes de 12 bits tant de dada1, com de dada2. Aquest és el motiu pel qual necessita més espai que l'habitual, essent una de les instruccions double-word que tenim al PIC.

☐ F En l'algoritme d'escaneig d'un Keypad "row-by-row", els díodes de protecció que posem a cada fila són per evitar curtcircuits en el cas que es premin més d'1 tecla dins la mateixa fila.

Quan escanegem per files, no hi ha cap problema en prémer més d'1 tecla dins la mateixa fila. El problema podria venir si la pulsació de varis botons dins la mateixa columna unís el valor 0 de la fila escanejada, amb el valor 1 d'alguna de les inactives. Els díodes prohibeixen aquesta corrent de curtcircuit.

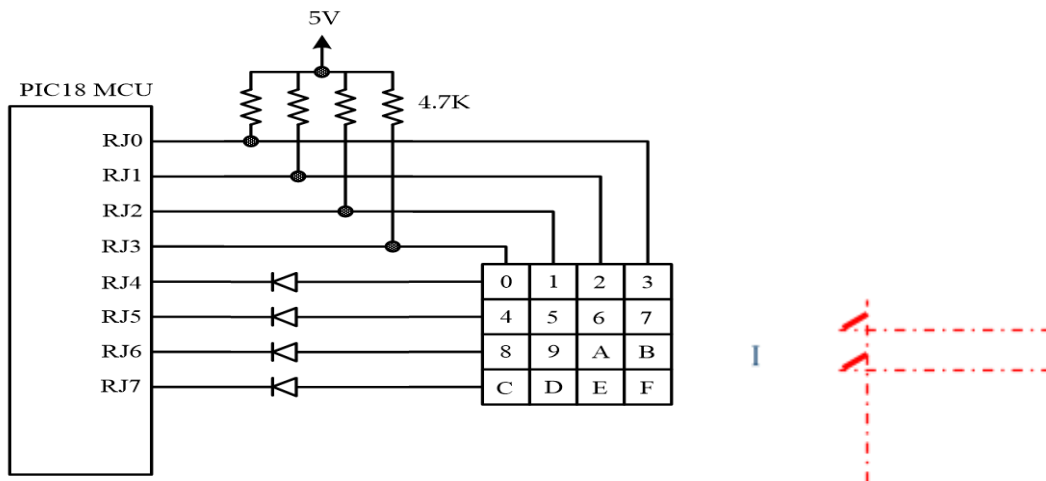


Figure 7.26b Sixteen-key keypad connected to PIC18 (used in all SSE demo boards)

Situació amb botons “0” i “4” premuts alhora, que podria haver causat un curtcircuit si no estiguessin els díodes

### P7. (1,5 punts)

Un alumne ha programat el següent codi per a mostrar si un botó està premut o no, usant un display de 7segments amb 4 caràcters (com el de la placa EasyPIC del laboratori). Sempre que el botó estigui apretat, ha d'aparèixer “PULS”, si no ho està ha de sortir “FREE”.

```
void main() {
    configPorts(); // ports digitals, i ajusta pins I/O adequadament

    while(1) {
        if (PORTBbits.RB0 == 1) {
            pintaDisp(3,CODI_P);
            pintaDisp(2,CODI_U);
            pintaDisp(1,CODI_L);
            pintaDisp(0,CODI_S);
            while (PORTBbits.RB0 == 1);
        }
        else {
            pintaDisp(3,CODI_F);
            pintaDisp(2,CODI_R);
            pintaDisp(1,CODI_E);
            pintaDisp(0,CODI_E);
            while (PORTBbits.RB0 == 0);
        }
    }
}
```

Però el resultat no és l'esperat, i en comptes d'aparèixer les paraules senceres, només s'encén el display de més a la dreta, com es veu a continuació:

El que es veu amb botó premut:



El que es veu amb botó lliure:



Cognoms, Nom \_\_\_\_\_ DNI \_\_\_\_\_

**Tota resposta sense justificar es considerarà nul·la !**

Expliqueu per què passa això, i si creieu que el problema ve del software, proposeu una modificació del programa per tal que es vegin les paraules “PULS” i “FREE” senceres.

Com ens diuen que el display de l'exercici és com el de la placa EasyPIC, sabem que amb un Port del PIC especifiquem els segments encesos o apagats dels dígit (7segments). I que tenim 4 pins del PIC addicionals per a escollir quin dígit dels quatre volem encendre. El codi crida a una rutina anomenada pintaDisp, que selecciona un dígit concret d'entre els 4 possibles i col·loca un valor a un port per a encendre/apagar els segments.

Però recordem que aquest sistema de tenir un únic Port compartit pels 4 dígit, requereix que fem multiplexació en el temps, i contínuament anem activant tots els dígit, en un cicle sense fi. Si en algun moment s'atura aquest cicle, ens quedarem mostrant només el darrer dígit demanat.

Si ens fixem al codi, després de pintar les 4 lletres de “PULS”, fem una **espera activa** fins que deixem de polsar. Tanmateix, després de pintar “FREE”, fem una **espera activa** fins que tornem a polsar.

Aquestes esperes actives innecessàries, bloquegen l'execució del bucle infinit del main, i per tant s'atura l'algoritme d'escriptura al display. La solució és tan simple com eliminar aquestes esperes actives.

```
while(1) {
    if (PORTBbits.RB0 == 1) {
        pintaDisp(3,CODI_P);
        pintaDisp(2,CODI_U);
        pintaDisp(1,CODI_L);
        pintaDisp(0,CODI_S);
        // ELIMINEM AIXO: while (PORTBbits.RB0 == 1);
    }
    else {
        pintaDisp(3,CODI_F);
        pintaDisp(2,CODI_R);
        pintaDisp(1,CODI_E);
        pintaDisp(0,CODI_E);
        // ELIMINEM AIXO: while (PORTBbits.RB0 == 0);
    }
}
```

Evidentment hi ha altres alternatives de codi que solucionen el problema. L'important és que no parem mai de refrescar l'escriptura dels 4 dígit. P.ex: podríem mantenir els dos “while problemàtics”, movent les 4 instruccions pintaDisp a dins de cada while, com veiem aquí:

```
while(1) {
    if (PORTBbits.RB0 == 1) {
        while (PORTBbits.RB0 == 1) {
            pintaDisp(3,CODI_P);
            pintaDisp(2,CODI_U);
            pintaDisp(1,CODI_L);
            pintaDisp(0,CODI_S);
        }
    }
    else {
        while (PORTBbits.RB0 == 0) {
            pintaDisp(3,CODI_F);
            pintaDisp(2,CODI_R);
            pintaDisp(1,CODI_E);
            pintaDisp(0,CODI_E);
        }
    }
}
```