

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
```

```

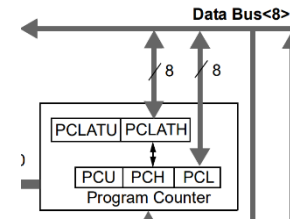
; Llegir el valor del PC
movff PCL, PCL_tmp ; Movem el contingut del registre PCL a algun registre
; Llegir el registre PCL actualitza els valors de PCLATH i PCLATU
movff PCLATH, PCH_tmp ; Movem el contingut del registre PCH a algun registre
movff PCLATU, PCU_tmp ; Movem el contingut del registre PCU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

```

```

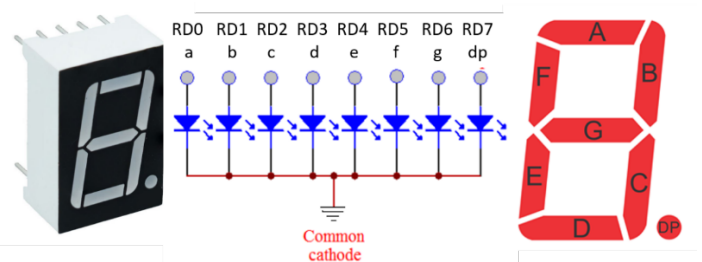
loop bra loop; Bucle infinit
end

```



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.



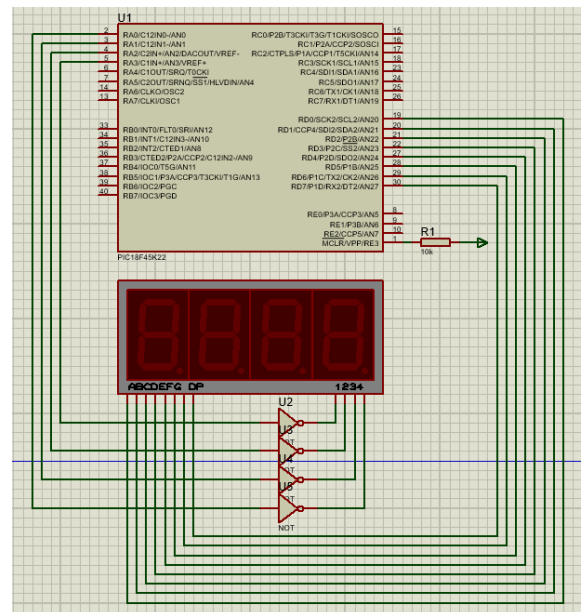
3.1 (0,5p) Quin valors tindran els registres ANSELA, ANSELD, TRISA i TRISD?

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

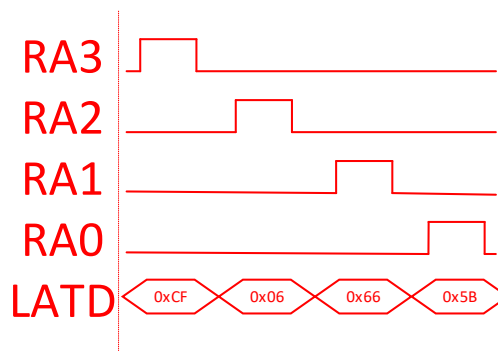
3.2 (1,5p) Dibuixa un cronograma en el que indiquis els valors que has d'escriure als registres LATA i LATD (el PORTA i el PORTD) per tal de veure el número '3.142' als displays (atenció al punt)

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:

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

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

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
```