Pràctiques de Sistemes Digitals i Microprocessadors Curs 2023-2024

Pràctica 1

LSBattleShip

Alumnes	Login	Nom
	oriol.aparicio	Oriol Aparicio Casanovas
	hicham.naf	Hicham Naf

Entrega	Placa	Memòria	Nota

Data	31/12/23

Index

1. Síntesi de l'enunciat

Per a realitzar la pràctica de LSBattleShip implementarem digitalment el joc d'enfonsar vaixells BattleShip entre dues persones.
El funcionament consistirà a fer que, els dos usuaris introdueixin les coordenades dels seus 5 vaixells per torns. Un cop distribuïts comença el joc, llavors per torns els jugadors hauran d'atacar les caselles del taulell i el sistema comprovarà si hi havia un vaixell contrincant. Finalment, el joc finalitzarà quan un dels jugadors hagi encertat la posició de tots els vaixells contrincants.

Per a realitzar la pràctica la dividirem en dues fases, i en aquesta memòria explicarem la fase 2. En aquesta segona fase ens centrarem a fer tota la lògica del taulell de joc, en el que mostrem en temps real les coordenades al taulell que es van introduint, per a després, en començar una partida, fer que puguem jugar.

Primer de tot, el que farem és guardar les coordenades introduïdes a la fase 1, per mostrar-les al taulell de cada jugador respectivament. Un cop estiguin totes introduïdes i guardades als taulells, podrem començar la partida. Un cop comencem, mostrarem en blau tots els taulells i mostrarem el cursor per anar seleccionant caselles per atacar-les. Aquestes es mostraran en verd (si l'hem encertat) o en vermell (si hem tocat aigua). Les accions d'atacar s'aniran intercalant entre els jugadors fins que un d'ells guanyi, llavors finalitzarà el joc.
En paral·lel, anirem ajustant els servos de la vida de la flota de cada jugador, que s'anirà actualitzant a mesura que es vagin encertant coordenades. També, si es toca un vaixell, sonarà un so agut per l'speaker, i si fallem un so greu, i en finalitzar la partida, sonarà una melodia de 5 notes durant 5 segons

Altrament, per comunicar la fase 2 amb la fase 1, ens ajudarem de 5 senyals per traspassar les dades d'una fase a l'altre. Els senyals seran: *BoatCoords[7..0], NewCoord, NewBoat, StartGame i CurrentPlayer*. Cal remarcar que el senyal de *NewBoat* no l'hem hagut de fer servir per a la nostra implementació.

2. Disseny de software

Main

Només iniciar el programa, el primer apartat de codi on anirem és el Main. Primer de tot ens dedicarem a inicialitzar ports, perifèrics i variables, per a poder utilitzar-los durant tota l'execució del programa.
Tot seguit, inicialitzarem la matriu de leds, mostrant en color blau tot el taulell, ja que no tindrem cap vaixell introduït de moment. Per a mostrar-la correctament, ho fem dues vegades, ja que si no no ho mostrava correctament.
Un cop fetes les inicialitzacions correctament, ens quedarem en bucle a la Fase 1, esperant el senyal de *StartGame*, però durant aquest temps ens dedicarem a cridar la interrupció de *NewCoord*, per anar introduint les coordenades (ho explicarem en el següent apartat en més detall).

```
MAIN
    CALL    CONFIG_PORTS                    ;inicialitzar ports
    CALL    INIT_VARS                       ;inicialitzar perifèrics i variables
    CALL    CONFIG_INT

    ;CARREGAR JUGADOR 0
    MOVLW   .255                ;AQUI ANIREM LLEGINT LES COORDENADES
    MOVWF   VALUEESPERA,0       ;DE LA FASE 1 I LES ANIREM GUARDANT A LA RAM
    CALL    ESPERA
    CALL    LECTURA_VALORS
    MOVLW   .255
    MOVWF   VALUEESPERA,0
    CALL    ESPERA
    CALL    LECTURA_VALORS

    MOVLW   .255
    MOVWF   VALUEESPERA,0
    CALL    ESPERA
```

Quan ens hagin premut *StartGame*, ja podrem començar una partida, però abans de tot caldrà inicialitzar el cursor, el timer 0 i invertir els taulells de manera que el jugador 0 ataqui el jugador 1 i viceversa. Abans d'entrar a comprovar els polsadors, però, mostrarem de nou el taulell un altre cop.

```
FASE1
    BTFSS   PORTD,RD2,0     ;ESPEREM EL SENYAL START GAME PER SABER SI
    GOTO FASE1              ;HEM DE SORTIR DEL BUCLE DE LA FASE 1 I
    CALL INIT_CURSOR        ;JA PODEM COMENÇAR A JUGAR
    CALL INIT_TIMER0
    BTG  CURRENT_GRID,0,0
    CALL    LECTURA_VALORS
    MOVLW   .255
    MOVWF   VALUEESPERA,0
    CALL        ESPERA
```

Un cop fet tot això, entrarem en un bucle on anirem comprovant els 5 polsadors d'acció que tenim, revisant periòdicament si s'ha premut algun i això ho anirem fent fins que s'acabi el joc.

```
LOOP                                    ;fem polling dels polsadors

    BTFSC   PORTD,RD1,0                 ;Esperem a que es premi el polsador.
    CALL    CONTROL_RIGHT
    BTFSS   PORTB,RB5,0                 ;Esperem a que es premi el polsador.
    CALL    CONTROL_DOWN
    BTFSC   PORTD,RD3,0                 ;Esperem a que es premi el polsador.
    CALL    CONTROL_ATACK
    BTFSS   PORTB,RB4,0                 ;Esperem a que es premi el polsador.
    CALL    CONTROL_LEFT
    BTFSS   PORTB,RB3,0                 ;Esperem a que es premi el polsador.
    CALL    CONTROL_UP
    GOTO LOOP

    END
```

Interrupció NewCoord

Aquesta interrupció ens saltarà quan estiguem en el bucle d'espera de la fase 1 i ens arribi el senyal de *NewCoord* que és per flanc de baixada en l'INT0.
Tal com està explicat a l'apartat de configuracions de la interrupció.

Quan salta la interrupció el primer que fem és mirar quin flag s'ha activat.

```
RSI_GEN
    BTFSC   INTCON,TMR0IF,0
    GOTO    RSI_T0
    BTFSC   INTCON,INT0IF,0
    GOTO    RSI_INT0
    BTFSC   INTCON3,INT1IF,0
    GOTO    RSI_INT1
    RETFIE FAST
```

Si en aquest cas ha estat el d'INT0IF, anem a la funció RSI_INT0.

```
RSI_INT0                                ;Interrupcio NewCoord
    MOVLW   .255
    MOVWF   VALUEESPERA,0
    CALL        ESPERA
    CALL ESCRIPTURA_VALORS_F1

    BTFSS   CURRENT_GRID,0,0
    INCF MAX_PWM,1,0

    BSF     LATA,RA3,0
    CALL    ESPERA_1MS
    BCF     LATA,RA3

END_RSI_INT0
    BCF     INTCON,INT0IF,0             ;Netejem el flac de la interrupcio i fem retfie fast
    RETFIE FAST
```

Aquí el que fem és esperar un temps d'espera de més de 50us per assegurar la inicialització de la matriu, no caldria perquè aquesta interrupció saltarà cada molt de temps, però ho hem deixat per si de cas. Llavors el que fem és cridar la funció ESCRIPTURA_VALORS_F1, que el que fa és escriure a la RAM al taulell del jugador que indica la variable CURRENT_GRID i l'adreça de la coordenada de BoatCoords.

En aquesta adreça hi posem un 01, d'aquesta manera indiquem que tenim una casella d'un vaixell en aquesta coordenada i mostrem el taulell. La funció ESCRIPTURA_VALORS_F1 està explicada amb més detall més endavant.

Després incrementem el valor de MAX_PWM si CURRENT_GRID val 0 per contar les caselles totals només d'un jugador.

Finalment, enviem el senyal d'ACK pel RA3 durant 1 ms perquè tingui durada suficient per a interceptar-la en la fase 1.

Interrupcio New Boat

La interrupció del *NewBoat* salta quan un jugador ha introduït la última casella del vaixell i ha passat 1 segon després, rebem el senyal *NewBoat*.
El que fem és fer Togle a la variable CURRENT_GRID per canviar de jugador i a la sortida RA4 que és el led del CURRENT GRID.

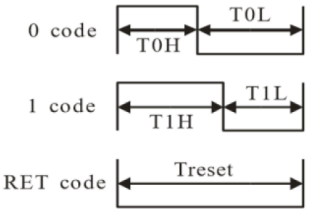
Llavors cridem la funció LECTURA_VALORS que el que fa és mostrar el taulell d'aquest jugador.
Finalment esperem un temps i baixem el flag de l'INT1 abans de sortir de la interrupció.

```
RSI_INT1                                ;Interrupció del NewBoat per canviar de jugador

    BTG CURRENT_GRID,0,0
    BTG LATA,RA4,0
    CALL    LECTURA_VALORS
    MOVLW   .255
    MOVWF   VALUEESPERA,0
    CALL        ESPERA

END_RSI_INT1
    BCF     INTCON3,INT1IF,0            ;Netejem el flac de la interrupcio i fem retfie fast
    RETFIE FAST
```

Assignar Color LED

Hem pensat a fer una funció genèrica que puguem fer servir per assignar qualsevol color a un led, en lloc de crear una funció per cada color és més òptim tenir una sola que ens faci qualsevol color. Primer de tot hem de saber que necessitem per printar un color a un LED, cada LED l'hi hem de passar la informació dels 3 colors RGB en 8 bits cadascun.

Cada bit que pot ser 1 o 0, se li ha d'indicar d'una manera específica com indica la següent taula:



Data transfer time(TH+TL=1.25μs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50μs	

Si volem un 1, haurem d'enviar un '1' durant 400 ns +-150 ns d'error i després un '0' durant 800 ns +-150 ns d'error.
Això traduït a temps d'instrucció serien 4 instruccions a '1' i en acabat 8 instruccions a '0'.
Per a enviar un 0, si aprofitem els marges d'error els temps seran oposats, primer enviar '1' durant 800 ns i en acabat un '0' durant 400 ns.
Fent-ho d'aquesta manera veiem que té una simetria que hem aprofitat de la següent manera:

```
BSF     LATD,RD0,0      ;VALOR= 1       0
NOP                     ;       (1)     (1)
NOP                     ;       (2)     (2)
BTFSS   VALOR,0,0       ;       (3)     (3)
BCF     LATD,RD0,0      ;       (4)     (4)
NOP                     ;       (1)     (5)
NOP                     ;       (2)     (6)
NOP                     ;       (3)     (7)
BCF     LATD,RD0,0      ;       (4)     (8)
NOP                     ;       (5)     (1)
NOP                     ;       (6)     (2)
NOP                     ;       (7)     (3)
NOP                     ;       (8)     (4)
```

Amb aquest codi el que fem és posar a 1 la sortida RD0 que és on tenim connectada la matriu durant 4 instruccions si VALOR val 1 o 8 instruccions si val 0.

Sabent el següent:
Un NOP només serveix per gastar 1 temps d'instrucció
BTFSS en el nostre cas gasta 2 temps d'instrucció si salta la línia sinó 1.

El següent pas és en lloc de comprovar el valor d'un sol bit, que la funció recorri tot una variable de 8 bits i vagi generant els polsos que tocant.
Per fer-ho hem fet uns d'una instrucció que és el Rotate, RLNCF, el que fa és la variable que té rota tots els seus valors cap a l'esquerra.
Amb això ens quedaria la següent funció:

```
1- LED_GREEN
2-      BSF     LATD,RD0,0
3-      rlncf LED_Green_Mirall,1,0
4-      NOP
5-      BTFSS LED_Green_Mirall,7,0
6-      BCF     LATD,RD0,0
7-      NOP
8-      NOP
9-      NOP
10-     BCF     LATD,RD0,0
11-     NOP
12-     NOP
13-     GOTO LED_GREEN
```

A LED_Green_Mirall hi tenim els 8 bits del color verd, posem com a exemple que hi tenim '11100000', fem un rotate cap a l'esquerra en la línia 3 (els rotates només gasten 1 temps d'instrucció) i comprovem el bit 7 en la línia 5, com que hem fet el rotate abans de comprovar el contingut ara tenim '11000001' és a dir que el bit que volem està sempre a la posició 7.

Es podria haver fet el rotate després de la comprovació perfectament.

Afegint una etiqueta al principi i un GOTO al final (tenint en compte que gasta 2 temps d'instrucció) aquest bucle es va repetint i printant els 1 i 0 de la variable de 8 bits.

Ara només és limitar aquest bucle perquè es faci només 8 cops, per a fer-ho utilitzem una variable anomenada COMPTADOR.

La idea principal del COMPTADOR és senzilla, aprofitem la instrucció rotate, si carregem un '10000000' en la variable COMPTADOR, quants rotates ens farien falta perquè l'1 que està al bit 0 torni a aquesta posició? Doncs exactament 8.

Llavors aprofitant això si posem un rotate al mig del codi i un bit test file al final, ens assegurem que aquest bucle es faci 8 cops.

```
LED_GREEN
    BSF LATD,RD0,0
    rlncf LED_Green_Mirall,1,0
    NOP
    BTFSS LED_Green_Mirall,7,0
    BCF LATD,RD0,0
    rlncf COMPTADOR,1,0
    NOP
    NOP
    BCF LATD,RD0,0
    BTFSS COMPTADOR,0,0
    GOTO LED_GREEN
```

Ara si repetim aquesta funció 2 cops més pels colors Vermell i Blau, ja podem assignar qualsevol color al led.

```
PRINT_LED

LED_GREEN
    BSF LATD,RD0,0
    rlncf LED_Green_Mirall,1,0
    NOP
    BTFSS LED_Green_Mirall,7,0
    BCF LATD,RD0,0
    rlncf COMPTADOR,1,0
    NOP
    NOP
    BCF LATD,RD0,0
    BTFSS COMPTADOR,0,0
    GOTO LED_GREEN
LED_RED
    BSF LATD,RD0,0
    rlncf LED_Red_Mirall,1,0
    NOP
    BTFSS LED_Red_Mirall,7,0
    BCF LATD,RD0,0
    rlncf COMPTADOR,1,0
    NOP
    NOP
    BCF LATD,RD0,0
    BTFSS COMPTADOR,0,0
    GOTO LED_RED
LED_BLUE
    BSF LATD,RD0,0
    rlncf LED_Blue_Mirall,1,0
    NOP
    BTFSS LED_Blue_Mirall,7,0
    BCF LATD,RD0,0
    rlncf COMPTADOR,1,0
    NOP
    NOP
    BCF LATD,RD0,0
    BTFSS COMPTADOR,0,0
    GOTO LED_BLUE
    RETURN
```

Primer assignem els colors a les variables LED_X_Mirall i tot seguit fem un CALL a PRINT_LED cada cop que volem mostrar el color en el led.

Polling Pulsadors

En aquesta part del codi hem organitzat les accions de cada polsador del taulell per poder moure'ns i atacar. Per començar, un cop hàgim entrat al loop dels polsadors al main, anirem constantment mirant si ens han premut algun dels 5 polsadors, i quan en premem un anirem a la part corresponent de cada polsador (la lògica serà la mateixa per a tots els polsadors, només canviarà a l'hora d'executar l'acció en concret).
Primer de tot, quan hàgim rebut l'acció que se'ns ha premut un polsador, anirem al seu apartat corresponent i llavors realitzarem una espera de 16ms per comprovar que no hagin sigut rebots.
Un cop comprovat, executarem l'acció d'aquell polsador, i quan tornem, esperarem que es deixi de prémer el polsador i tornarem a fer una espera de 16ms per comprovar que realment s'ha deixat de prémer.
Aquí veiem un exemple amb el polsador Right:

```
CONTROL_RIGHT
    CALL    ESPERA_16MS
    BTFSS   PORTD,RD1,0
    RETURN
    CALL    MOURE_RIGHT
ESPERA_POLSADOR_RD1
    BTFSC   PORTD,RD1,0
    GOTO    ESPERA_POLSADOR_RD1
    CALL    ESPERA_16MS
    BTFSC   PORTD,RD1,0
    GOTO    ESPERA_POLSADOR_RD1
    RETURN
```

Mòdul de control de moviment del cursor + atacar

En aquesta part hem estructurat la inicialització del cursor, els moviments d'aquest i el polsador d'atac. Com bé hem parlat a l'apartat de polling, allà simplement fèiem la verificació del polsador premut, i en aquest ens dedicarem a fer les accions pertinents.
Primer de tot tindrem la inicialització del cursor, on inicialitzarem la seva posició a 1,1 cada vegada que canviem de jugador.
A continuació tenim la implementació d'Up, Down, Left i Right. Per a la implementació veurem que Down i Right tindran la mateixa lògica igual que Up i Left, per tant, només explicarem Down i Up.
Per a moure cap avall el cursor (Down), simplement incrementem en una unitat el valor de la coordenada Y i en cas que superes el valor de 8 el que faríem és actualitzar el valor pel número 1. Un cop actualitzat el valor del cursor, simplement cridem la funció per pintar la matriu actualitzada.

```
MOURE_DOWN
    ;DOWN
    MOVLW .1
    ADDWF COORD_Y, 1,0      ;SUMEM 1 AL VALOR DE LA COORD_Y
    ;SI VOLEM SALTAR DEL 8 AL 1
    MOVLW .8
    CPFSGT COORD_Y, 0       ;COMPARA EL WREG AMB COORD_Y, SI COORD_Y ES MES GRAN SALTA DE LINEA
    GOTO LECTURA_VALORS
    MOVLW .1
    MOVWF COORD_Y, 0
    GOTO LECTURA_VALORS
```

En el cas de moure cap amunt el cursor simplement anem decrementant el valor de la coordenada Y en una unitat, i en cas que al decrementar valgues 0, substituïm el valor per un 8. Un cop actualitzat el valor del cursor, simplement cridem la funció per pintar la matriu actualitzada.

```
MOURE_UP
    ;UP + SALTAR DE 1 A 8
    DECFSZ COORD_Y, 1, 0     ;RESTEM 1 AL VALOR DE COORD_Y
    GOTO LECTURA_VALORS    ;TORNEM SI NO ESTEM A 1
    MOVLW .8
    MOVWF COORD_Y, 0         ;SI ESTAVEM A 1 EL POSEM A 8
    GOTO LECTURA_VALORS
```

Per a l'acció d'atacar, primer de tot buscarem la coordenada del cursor a la RAM, i observarem el valor de la posició seleccionada, en cas de tenir 00 voldrà dir que hem tocat aigua i en cas de tenir 01 voldrà dir que hem tocat un vaixell. En cas que no es cap dels dos, simplement netegem el valor del punter de la RAM i tornem a esperar rebre una acció.

```
ATACAR
    ;CONVERTIR A RAM LA COORDENADA ACTUAL
    CALL CONVERT_COORD_TO_RAM

    MOVFF RAM_COORD,FSR0L       ;AGAFEM L'ADREÇA DE LA COORDENADA A ATACAR I LA CARREGUEM AL PUNTER

    ;COMPROVAR EL VALOR QUE TE 00 O 01
    MOVFF   INDF0, CONTINGUT_RAM ;COPIEM EL CONTINGUT DE LA RAM A LA VARIABLE CONTINGUT_RAM
    MOVLW .0
    CPFSGT  CONTINGUT_RAM, 0    ;SI EL CONTINGUT DE LA RAM EL MAJOR A 00 S'HO SALTA
    GOTO HAS_TOCAT_AIGUA
    MOVLW .1
    CPFSGT  CONTINGUT_RAM, 0    ;SI EL CONTINGUT DE LA RAM EL MAJOR A 01 S'HO SALTA
    GOTO HAS_TOCAT_BARCO
    CLRF FSR0L,0
    RETURN
```

En cas d'haver tocat aigua posarem la coordenada de color vermell (10 a la RAM) i mourem el cursor fora de la matriu a mostrar. Tot seguit, mostrarem el taulell sencer amb la coordenada actualitzada i realitzarem un so greu amb l'altaveu durant un segon, després esperem dos segons més i finalment canviarem el jugador i mostrarem el seu taulell d'atac.

```
HAS_TOCAT_AIGUA
    ;POSAR LA POSICIO EN VERMELL
    MOVLW .2
    MOVWF INDF0, 0          ;POSEM 10 A LA COORDENADA ATACADA
    MOVLW b'00001001'               ;MOVEM EL CURSOR FORA DE LA MATRIU A MOSTRAR
    MOVWF COORD_Y,0
    ;MOSTREM LA MATRIU ACTUALITZADA
    CLRF FSR0L,0
    CALL LECTURA_VALORS
    ;FER EL SOROLL GREU 1S
    CLRF COMPT_1S,0
    MOVLW .78
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU
    CLRF COMPT_1S,0
    CALL ESPERA_1S
    CLRF COMPT_1S,0
    CALL ESPERA_1S

    CALL INIT_CURSOR
    BTG CURRENT_GRID,0,0
    BTG LATA,RA4,0
    CALL LECTURA_VALORS

    RETURN
```

En cas d'haver encertat un vaixell, haurem de fer més accions. Per començar, actualitzarem la posició del cursor de color verd (11 a la RAM) i mourem el cursor fora de la matriu visible. Tot seguit mostrarem el taulell sencer amb la coordenada actualitzada i actualitzarem el PWM del servo del jugador corresponent. Finalment, farem un soroll agut per l'altaveu durant un segon i esperarem dos segons més, i un cop passat aquest temps comprovarem si el PWM del servo d'algun dels dos jugadors és 1, és a dir, que ja hem enfonsat tota la seva flota i en cas de ser així finalitzarem el joc. En cas contrari canviarem el jugador i mostrarem el seu taulell d'atac.

```
HAS_TOCAT_BARCO
    ;POSAR LA POSICIO EN VERD
    MOVLW .3
    MOVWF INDF0, 0          ;POSEM 11 A LA COORDENADA ATACADA
    MOVLW b'00001001'               ;MOVEM EL CURSOR FORA DE LA MATRIU A MOSTRAR
    MOVWF COORD_Y,0

    ;MOSTREM LA MATRIU ACTUALITZADA
    CLRF FSR0L,0
    CALL LECTURA_VALORS
    BTFSC CURRENT_GRID ,0,0
    DECF PWM_S1,1,0                 ;PROVA   ????
    BTFSS CURRENT_GRID ,0,0
    DECF PWM_S0,1,0                 ;PROVA
    ;FER EL SOROLL AGUT 1S
    CLRF COMPT_1S,0
    MOVLW .10
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU
    CLRF COMPT_1S,0
    CALL ESPERA_1S
    CLRF COMPT_1S,0
    CALL ESPERA_1S
    MOVLW   .1
    CPFSGT   PWM_S0,0
    GOTO    END_GAME
    CPFSGT   PWM_S1,0
    GOTO    END_GAME

    CALL INIT_CURSOR
    BTG CURRENT_GRID,0,0
    BTG LATA,RA4,0
    CALL LECTURA_VALORS
    RETURN
```

Dins d'aquest apartat també hem inclòs l'espera d'un segon i també la generació del soroll de l'altaveu, que aquest dependrà de la freqüència que se li introdueixi. Més endavant s'explica en més detall el seu funcionament.

```
ESPERA_1S
    MOVLW .50
    CPFSGT COMPT_1S,0
    GOTO ESPERA_1S
    RETURN

SOROLL_ALTAVEU
    BSF LATA,RA5,0
    MOVFF   SOROLL_FREQ, COMPTADOR2_16MS
    CALL INCREMENTA_XS
    BCF LATA,RA5,0
    MOVFF   SOROLL_FREQ, COMPTADOR2_16MS
    CALL INCREMENTA_XS
    MOVLW .50
    CPFSGT COMPT_1S,0
    GOTO SOROLL_ALTAVEU
    RETURN
```

Control moviment servomotors

Per a fer els moviments dels servomotors, només podem utilitzar el Timer 0 i com que els Servomotors necessiten PWM constantment, no ho podem fer en el main, ja que ocuparíem tot el temps en generar aquests PWM i no podríem fer altres coses, per aquest motiu utilitzem interrupcions.

La idea principal és optimitzar el control dels servomotors el màxim possible, això és identificant que tenen en comú els 2 servomotors i que poden compartir.

Per aquest motiu hem dividit el període de 20ms en 3 regions, els primers 0,5ms sempre valdran '1' les sortides dels servos, mentre que els últims 17,5 ms sempre valdran '0'.

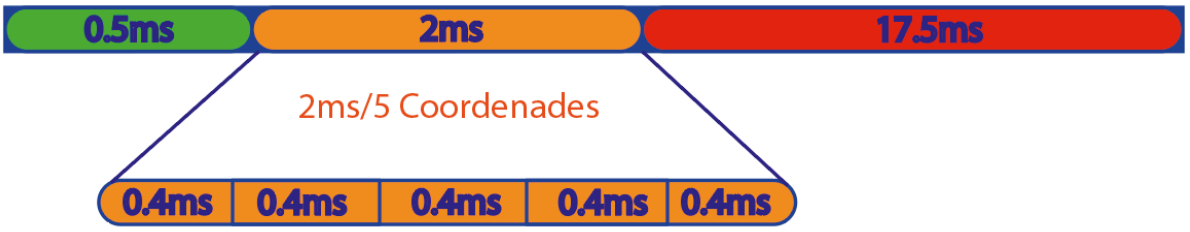
Llavors ens interessa més la part dels 2ms on indiquem l'angle dels servomotors, sabent de quant val '1' durant tota aquesta regió els servomotors estaran a 180º i si val '0' estaran a 0º.

El que defineix l'angle dels servomotors és la vida dels vaixells que tenen en el taulell, i aquests poden ser de mida 1 mínim i mida 5 màxim, cada jugador té 5 vaixells.

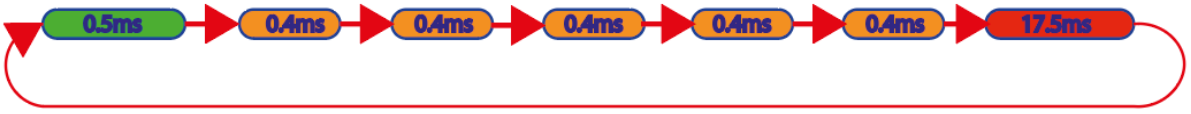
Sabent això les caselles mínimes que podem tindre son 5 i les màximes són 25, si per exemple tenim 5 caselles en un taulell, el servomotor estarà a 180º, si toquen una casella l'angle disminuirà en proporció al màxim de caselles, és a dir ⅕ de 180º.

Sabent això el que ens interessa és dividir els 2ms pel nombre de caselles dels vaixells que hi ha al taulell.

Si tenim 5 caselles ho dividirem en 5 regions, si tenim 25 caselles llavors en 25.



Donem com a exemple que tenim 5 coordenades, el que volem és que quan salti la interrupció per primer cop, baixar el flag del timer 0 i deshavilitar les interrupcions. Carregar 0,5ms al timer 0, posar a 1 els 2 servos, anar a un bucle d'espera per el flag del timer0, llavors carregar 0,4ms (que hem anomenat temps d'increment) i tornar a esperar, això 5 cops, finalment carregar els 17,5ms al timer i sortir de la interrupció activant abans la interrupcio del timer 0 així cíclicament.

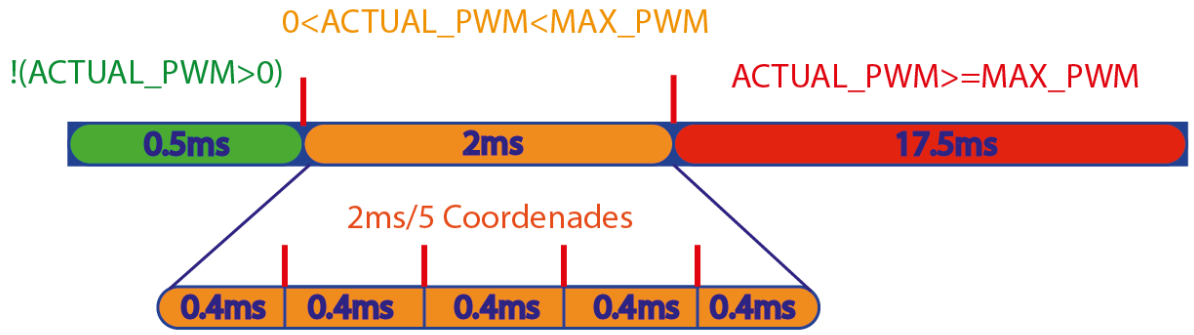


Per aquest motiu fem us d'una variable anomenada ACTUAL_PWM, que inicialment val 0 i que anem incrementant cada cop que cambiem el valor del timer 0. Llavors el que fem dintre de la interrupció és depenent del que valgui aquesta variable carregem el temps de la regió que toqui.

Si val 0, carreguem 0.5ms, si val entre 1 i 5 carregarem 0.4ms i si val 6 carregarem 17.5ms i en lloc de incrementar en 1 la variable en aquest últim cas, la tornarem a posar a 0.

```
;-------------------------------------------------------------------------------
TIMER0_LOW
    MOVLW   HIGH(.64286)                    ;(16b) 65536 - 1250 = 64286 (0.5ms)
    MOVWF   TMR0H,0
    MOVLW   LOW(.64286)
    MOVWF   TMR0L,0
    INCF ACTUAL_PWM,1,0
    INCF COMPT_1S,1,0
    CALL ESPERA_T0
    RETURN
TIMER0_INCREMENT
    MOVFF   TEMPS_INC_H, TMR0H
    MOVFF   TEMPS_INC_L, TMR0L
    CALL ESPERA_T0
    RETURN
TIMER0_HIGH
    MOVLW   HIGH(.21786)                    ;(16b) 65536 - 43750 = 21786 (17.5ms)
    MOVWF   TMR0H,0
    MOVLW   LOW(.21786)
    MOVWF   TMR0L,0
    CLRF ACTUAL_PWM,0
    RETURN
T0_LOW
    BSF LATA,1,0
    BSF LATA,2,0
    CALL TIMER0_LOW
    GOTO RSI_T0
T0_HIGH
    CALL TIMER0_HIGH
    BCF LATA,1,0
    BCF LATA,2,0
    GOTO END_RSI_T0

ESPERA_T0
    BTFSS INTCON,TMR0IF,0
    GOTO ESPERA_T0
    RETURN
```

En la següent imatge veiem com fem les diverses comparacions i depenent del valor d'ACTUAL_PWM cridem una de les funcions anteriors que toqui.
MAX_PWM és la que ens indica el nombre d'increments que tenim + 1.



Primer mirem si ACTUAL_PWM és més gran que 0, si no es compleix, carreguem 0.5ms i posem a '1' els 2 servomotors.

Si es compleix la condició, llavors mirem si és més petit que MAX_PWM, si no es compleix carreguem 17,5ms i forsem els 2 servomotors a '0'.

Si es compleix vol dir que estem dintre de la regió dels 2ms i llavors hem de comprovar els estats dels servomotors i carregar el temps d'increment.

```
RSI_T0
    BCF     INTCON,TMR0IE,0             ;Desactivem interrupcio timer0
    BCF     INTCON,TMR0IF,0
    MOVLW .0
    CPFSGT ACTUAL_PWM,0
    GOTO    T0_LOW
    MOVF ACTUAL_PWM,0
    CPFSGT MAX_PWM,0
    GOTO    T0_HIGH

    MOVF ACTUAL_PWM,0
    CPFSGT PWM_S0,0
    BCF LATA,1,0
    CPFSGT PWM_S1,0
    BCF LATA,2,0
    CALL TIMER0_INCREMENT
    INCF ACTUAL_PWM,1,0
    GOTO RSI_T0
```

Ara finalment és controlar els angles dels servomotors, això ho fem amb 2 variables que fan de vida dels servomotors. PWM_S0 i PWM_S1.

Aquestes 2 variables sempre estaran inicialitzades a MAX_PWM i anirem decrementant en 1 el PWM_S0 quan es toqui una casella del jugador 0 i el PWM_S1 quan es toqui la del jugador 1.

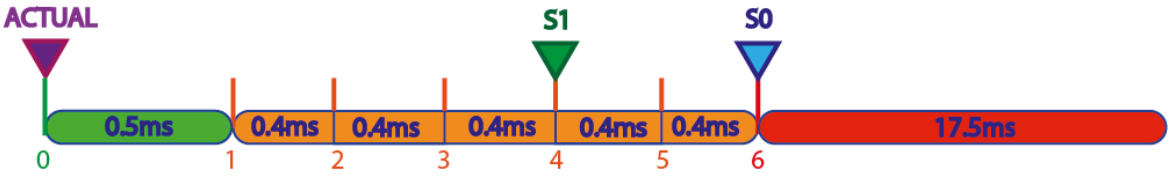
Aquestes dues variables les comparem amb l'ACTUAL_PWM cada cop que estiguem dintre de la regió dels 2ms.

Exemple funcionament:

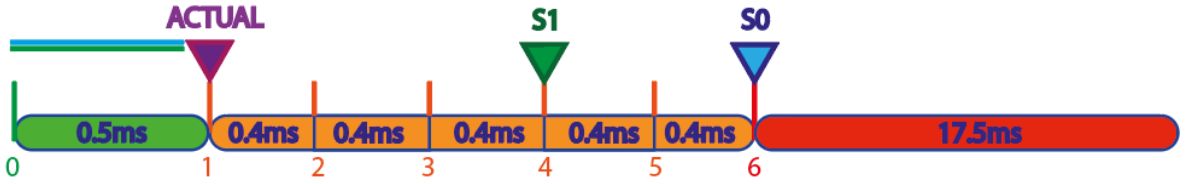
Seguint amb l'exemple anterior on tenim 5 coordenades donem el cas que el Jugador 0 té totes les caselles intactes, però, en canvi, el jugador 1 l'hi han tocat 2.

És a dir que PWM_S0 segueix valent MAX_PWM que seria 6 i PWM_S1 l'hi restem 2, així que 4.
El ACTUAL_PWM comensaria valent 0.

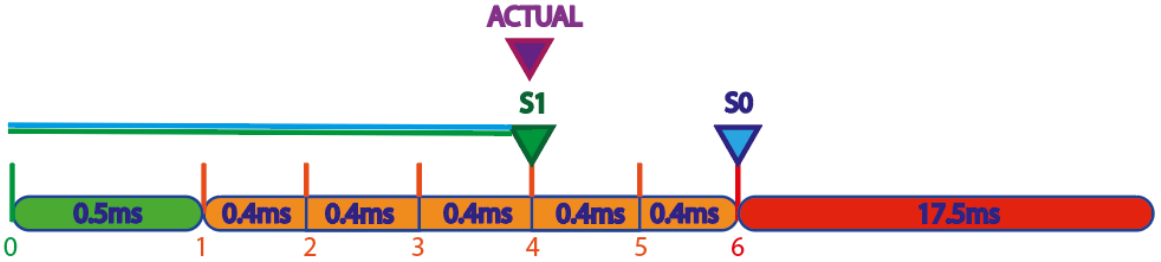
Llavors quan salti la interrupció com que ACTUAL_PWM val 0 carregem 0.5ms al Timer 0, l'incrementem en un i posem a '1' els 2 servomotors.



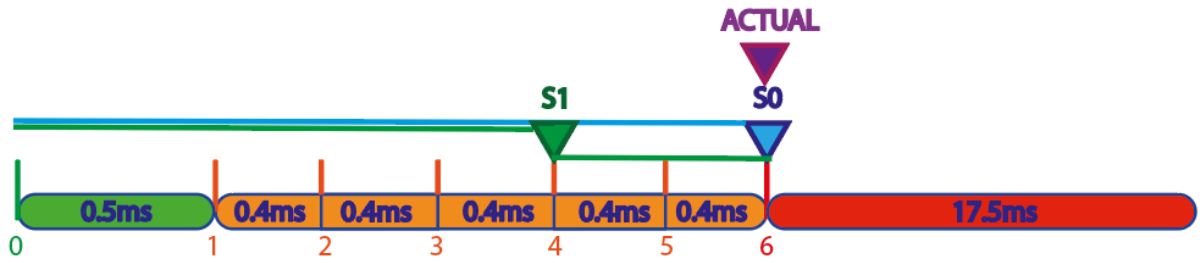
Després de 0.5ms, ara ACTUAL_PWM val 1, es a dir que carregem el temps d'increment al Timer 0 i comprovem els estats dels PWM_S0 i PWM_S1.
Com que els 2 segueixen sent més grans que ACTUAL_PWM no posem a '0' cap.
Incrementem en 1 l'ACTUAL_PWM.



Aquesta situació es repeteix 2 cops més fins que ACTUAL_PWM val 4 què és el mateix valor que te PWM_S1. En aquest cas no es compleix que PWM_S1 és més gran que ACTUAL_PWM així que la posem a '0'.



Durant els 2 següents trams PWM_S1 val '0', llavors quan ACTUAL_PWM valgui igual que MAX_PWM, no carreguem 0.4ms al Timer 0 sinó 17.5ms i forcem els dos PWM a '0'. Posem ACTUAL_PWM a 0 abans de sortir de la interrupció i quan torni a saltar d'aquí 17.5ms ja hauran passat 20ms en total i tornem a repetir el cicle.



Aquesta lògica s'ha d'aplicar per a totes les situacions de 5 coordenades a 25 coordenades.
La unica variació que haurà és el temps d'increment que depèn de les coordenades introduides, per aquest motiu hem fet una taula a la flash amb tots els casos i abans d'iniciar la fase 2, mirem quantes coordenades tenim i carregem el temps d'increment a les dues variables TEMPS_INC_H i TEMPS_INC_L.

L'explicació de com hem creat la taula, calculats els valors i inicialitzat les variables està explicat més endavant a l'apartat de Configuracions del Microcontrolador.

Comptador 1 segon

Necessitem poder comptar fins a 1 segon per poder fer els temps d'espera que s'ens demana en diferents llocs i per a la durada dels tons del Speaker.

La primera opció que està descartada és fer un bucle d'espera d'un segon, com que anem a una velocitat tan ràpida necessitaríem 10 milions d'instruccions per a fer 1 segon.

El que hem plantejat és aprofitar la interrupció del timer 0 per a comptar aquest segon, el que hem fet és cada cop que entrem en la funció del TIMER0_LOW, incrementem en 1 una variable anomenada COMPT_1S.

```
TIMER0_LOW
    MOVLW   HIGH(.64286)                ;(16b) 65536 - 1250 = 64286 (0.5ms)
    MOVWF   TMR0H,0
    MOVLW   LOW(.64286)
    MOVWF   TMR0L,0
    INCF ACTUAL_PWM,1,0
    INCF COMPT_1S,1,0
    RETURN
```

Llavors sabent que incrementem en 1 cada 20ms, necessitaríem 50 increments per a arribar a 1 segon.

Aixi que cada cop que vulguem comptar 1 segon, fem clear a aquesta variable i anem a un bucle d'espera fins que COMPT_1S valgui 50.

```
    CLRF COMPT_1S,0
    CALL ESPERA_1S

ESPERA_1S
    MOVLW .50
    CPFSGT COMPT_1S,0
    GOTO ESPERA_1S
    RETURN
```

Tons Speaker

Perquè el Speaker pugui generar un tó se li ha d'enviar un PWM de 50% dutty sicle a una determinada freqüència.
Per a un tó greu ho fem amb una freqüència d'aproximadament 250Hz, i per a un to agut a una freqüència de 2kH.

Primer hem calculat el temps d'instrucció dels períodes de 250Hz i 2kHz.

Comencem pels 250Hz, aquest en temps seria 1/250Hz= 0,004 segons, 4ms.
Ara mirem quantes instruccions són dividint-lo entre 100ns. 0,004/0,0000001=40.000 instruccions.
Hem creat un bucle on tenim 2 variables, el COMPTADOR1 que sempre estarà inicialitzat a 255 i el COMPTADOR2 que l'iniciem nosaltres amb el valor que vulguem.

La funció INCREMENTA_XS, va repetint un bucle de 255 decrementant el COMPTADOR1, tantes vegades com indiquem amb el COMPTADOR2.

```
INCREMENTA_XS
    DECFSZ  COMPTADOR1,1,0          ;miro si COMPTADOR1 és 0
    GOTO    INCREMENTA_XS           ;si no és 0, segueixo comptant
    SETF    COMPTADOR1,0
    DECFSZ  COMPTADOR2,1,0          ;miro si COMPTADOR2 és 0
    GOTO    INCREMENTA_XS           ;si no és 0, segueixo comptant
    RETURN
```

Així que si volem fer un to agut, necessitem posar a '1' la sortida del Speaker durant 20.000 instruccions i després posar-la a '0' durant 20.000 instruccions.

Llavors hem creat una funció genèrica pel to del Speaker on a la Variable SOROLL_FREQ tenim el valor que volem carregar al COMPTADOR2.

El que fa aquesta funció és estar-se 1 segon generant aquest to, fent clear a la variable COMPT_1S abans d'entrar, mentre estem a dintre anem generant un PWM de dutty sicle 50% amb el període que volem i al final comprovem si COMPT_1S val 50 que voldrà dir que ha passat 1 segon.

```
SOROLL_ALTAVEU
    BSF LATA,RA5,0
    MOVFF   SOROLL_FREQ, COMPTADOR2
    CALL INCREMENTA_XS
    BCF LATA,RA5,0
    MOVFF   SOROLL_FREQ, COMPTADOR2
    CALL INCREMENTA_XS
    MOVLW .50
    CPFSGT COMPT_1S,0
    GOTO SOROLL_ALTAVEU
    RETURN
```

Ara seguint amb la generació del to agut, el període són 40.000 instruccions, la meitat d'aquest és 20.000 que haurem de fer en bucles de 256 és a dir 20.000/256=78,125 aproximadament 78.

Aquest número és el que guardem a la variable SOROLL_FREQ abans de cridar la funció SOROLL_ALTAVEU.

```
HAS_TOCAT_AIGUA
    ;POSAR LA POSICIO EN VERMELL
    MOVLW .2
    MOVWF INDF0, 0          ;POSEM 10 A LA COORDENADA ATACADA
    MOVLW b'00001001'               ;MOVEM EL CURSOR FORA DE LA MATRIU A MOSTRAR
    MOVWF COORD_Y,0
    ;MOSTREM LA MATRIU ACTUALITZADA
    CLRF FSR0L,0
    CALL LECTURA_VALORS
    ;FER EL SOROLL GREU 1S
    CLRF COMPT_1S,0
    MOVLW .78
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU
    CLRF COMPT_1S,0
    CALL ESPERA_1S
    CLRF COMPT_1S,0
    CALL ESPERA_1S

    CALL INIT_CURSOR
    BTG CURRENT_GRID,0,0
    BTG LATA,RA4,0
    CALL LECTURA_VALORS

    RETURN
```

Mòdul Control de rebots

Per a controlar els rebots dels polsadors, hem creat una funció que és un bucle d'espera de 16ms anomenada ESPERA_16MS.
Tenim dos variables, una anomenada COMPTADOR1 que està sempre inicialitzada a 255 i el COMPTADOR2 l'hi carregem el número de quants cops volem fer el bucle de 256.
El d'aquest i el funcionament del bucle està explicat a l'apartat Tons Speaker.

També tenim una funció d'espera d'1ms per a quan vulguem enviar l'ACK.

```
;
ESPERA_16MS
    SETF    COMPTADOR1,0                ;a freq = 40MHz, tenim un Tinst = 100ns,
    MOVLW   .208                        ;amb increments de 255 i 208 arribem a 16ms
    MOVWF   COMPTADOR2,0
    GOTO INCREMENTA_XS

INCREMENTA_16MS
    DECFSZ  COMPTADOR1,1,0              ;miro si COMPTADOR1 és 0
    GOTO    INCREMENTA_16MS             ;si no és 0, segueixo comptant
    SETF    COMPTADOR1,0
    DECFSZ  COMPTADOR2,1,0              ;miro si COMPTADOR2 és 0
    GOTO    INCREMENTA_16MS             ;si no és 0, segueixo comptant
    RETURN

ESPERA_1MS
    SETF    COMPTADOR1,             ;a freq = 40MHz, tenim un Tinst = 100ns,
    MOVLW   .13                     ;amb increments de 255 i 13 arribem a 1ms
    MOVWF   COMPTADOR2,0
    GOTO INCREMENTA_XS

INCREMENTA_XS
    DECFSZ  COMPTADOR1,1,0              ;miro si COMPTADOR1 és 0
    GOTO    INCREMENTA_XS               ;si no és 0, segueixo comptant
    SETF    COMPTADOR1,0
    DECFSZ  COMPTADOR2,1,0              ;miro si COMPTADOR2 és 0
    GOTO    INCREMENTA_XS               ;si no és 0, segueixo comptant
    RETURN
```

Mòdul control RAM

Aquí explicarem tot el relacionat amb la lectura i escriptura de la RAM.

LECTURA TAULELL RAM I MOSTRAR MATRIU

Primer de tot, per poder llegir els valors de la RAM (i mostrar, per tant, els colors a la matriu de leds), comprovarem el jugador actual per saber a quines posicions haurem d'anar, ja que en el nostre cas disposem de 128 posicions de memòria del Banc 1, on les 64 primeres (de la 0 a la 63) seran del jugador 0 i les 64 següents (de la 64 a la 127) seran del jugador 1.

Un cop comprovat el jugador, començarem a llegir la part corresponent de la RAM, on realitzarem un bucle per a mostrar cada una de les coordenades amb el color que toca, abans de res, però, haurem de desactivar la interrupció del timer 0, ja que volem evitar que ens salti en mig de la lectura de la RAM i, per tant, no es mostri el taulell correctament.

Un cop fet això agafarem el valor del punter de la RAM (POSTINC0) i el guardarem en una variable per no perdre la posició que estem mirant actualment. Seguidament, comprovarem si l'usuari ha començat la partida, en cas d'haver començat mostrarem els valors de la fase 2 i en cas de no haver començat, mostrarem els de la fase 1.

Tot seguit comprovarem per a quin jugador estem mostrant el taulell, tot utilitzant la comprovació que hem fet a l'inici, fent que partim o bé del 0 (jugador 0) o bé del 65 (jugador 1). Tot això ho repetirem per totes les caselles del taulell fins que el tinguem tot mostrat.

En cas que comencem per la posició 0, mostrarem tot el taulell del jugador 0, però el disseny que tenim també acabarà "pintant" el jugador 1, encara que a la matriu no es veurà, ja que només mostrarà la primera part que ja s'estarà mostrant.

Finalment, comprovarem si estem a la fase 2 per saber si hem d'activar un altre cop la interrupció del timer 0 per al control dels servos.

```
LECTURA_VALORS
    MOVLW .65
    BTFSC CURRENT_GRID,0,0
    MOVWF   FSR0L,0

BUCLE_LECTURA
    BCF     INTCON,TMR0IE,0             ;Desactivem interrupcio timer0

    MOVFF POSTINC0, VALOR               ;Guardem la posicio de la RAM actual i incrementem el punter
    ;Tenim el color de tots els taulells.
    BTFSS PORTD, 2, 0  ;SI L'USUARI HA COMENÇAT EL JOC MOSTRAREM ELS VALORS DE LA FASE 2
    CALL PRINT_VALORS_F1
    BTFSC PORTD, 2, 0  ;SI L'USUARI NO HA COMENÇAT EL JOC MOSTRAREM ELS VALORS DE LA FASE 1
    CALL PRINT_VALORS_F2
    MOVLW .64
    CPFSGT FSR0L, 0                     ;Jugador 0
    GOTO BUCLE_LECTURA
    MOVLW .129
    CPFSEQ FSR0L, 0                     ;Jugador 1
    GOTO BUCLE_LECTURA
    CLRF FSR0L, 0
    BTFSC   PORTD,RD2,0                 ;Activem la interrupcio nomes si estem a la Fase 2
    BSF     INTCON,TMR0IE,0             ;Activo interrupcio timer0

    RETURN
```

PRINT_VALORS_F1

Aquesta funció la utilitzem per il·luminar la casella en la fase 1, simplement comprovem el valor de la variable VALOR, si val '00' vol dir que és una casella d'aigua i volem mostrar el color blau, si val '01' vol dir que és una casella d'un vaixell i volem posar-la de color verd.

```
;-------------------------------------------------------------------------
; 5. MODUL CONTROL DE LED
;-------------------------------------------------------------------------
PRINT_VALORS_F1
    MOVLW .0
    CPFSGT VALOR,0
    GOTO ASSIGN_BLUE
    MOVLW .1
    CPFSGT VALOR,0
    GOTO ASSIGN_GREEN
    RETURN
```

Això ho fem cridant les funcions ASSIGN_X on X és el color que volem.

On dintre d'aquesta funció assignem el color que volem a les 3 variables LED_X_Mirall i llavors cridem la funció PRINT_LED per enviar aquest color a la Matriu.

El funcionament d'aquesta funció està explicat més endavant.

```
ASSIGN_WHITE
    MOVLW  .3
    MOVWF LED_Blue_Mirall,0
    MOVWF LED_Red_Mirall,0
    MOVWF LED_Green_Mirall,0
    GOTO PRINT_LED

ASSIGN_BLUE
    MOVLW   .1
    MOVWF LED_Blue_Mirall,0
    CLRF  LED_Red_Mirall,0
    CLRF  LED_Green_Mirall,0
    GOTO PRINT_LED

ASSIGN_GREEN
    MOVLW   .3
    MOVWF LED_Green_Mirall,0
    CLRF  LED_Red_Mirall,0
    CLRF  LED_Blue_Mirall,0
    GOTO PRINT_LED

ASSIGN_RED
    MOVLW  .3
    MOVWF LED_Red_Mirall,0
    CLRF  LED_Blue_Mirall,0
    CLRF  LED_Green_Mirall,0
    GOTO PRINT_LED
```

PRINT_VALORS_F2

En aquesta funció assignem el color de la casella en la fase 2, aquí podem il·luminar el led de 4 colors diferents, blanc si volem el cursor, blau si és aigua, vermell si ha fallat i verd si ha encertat.

El primer que fem és cridar la funció CONVERT_CORD_TO_RAM per convertir la coordenada del cursor a adreça de la RAM, tot seguit la comparem amb el punter FSR0L, el motiu pel qual l'hi sumem un 1 a l'adreça del cursor és perquè utilitzem el POSTINC per llegir els valors de la RAM, per aquest motiu el punter FSR0L no està apuntant a l'adreça que estem mostrant sinó una de més.

Llavors si RAM_COORD i FSR0L són iguals llavors mostrem color blanc.
Si són diferents saltem a l'etiqueta PRINT_V_2, aqui mirem si la variable VALOR val '11' llavors mostrem verd, si val '10' mostrem vermell, i si no es cap de les 2 mostrem blau.

```
PRINT_VALORS_F2
    CALL CONVERT_COORD_TO_RAM
    MOVF RAM_COORD, 0
    INCF WREG,0,0

    CPFSEQ FSR0L,0          ;A la fase 2 printarem el taulell i a la posició
    GOTO PRINT_V_2          ;on tinguem el cursor la mostrarem de color blanc
    GOTO ASSIGN_WHITE

PRINT_V_2
    MOVLW .3
    CPFSLT VALOR,0
    GOTO ASSIGN_GREEN
    MOVLW .2
    CPFSLT VALOR,0
    GOTO ASSIGN_RED
    GOTO ASSIGN_BLUE
```

ESCRIPTURA RAM FASE 1

Aquesta part del codi es cridarà amb una interrupció generada pel senyal *NewCoord*, que vindrà de la Fase 1, indicant que hi ha una nova coordenada en espera de ser guardada, i és exactament el que farem nosaltres aquí.

Primer de tot cridarem la funció COORD_F1 on obtindrem la coordenada introduïda en valor d'adreça de RAM (més endavant explicarem la seva implementació). Tot seguit, aquest valor el carregarem al punter de la RAM, per a poder guardar en aquesta adreça un 1 (01, que farà referència al color verd a l'hora de mostrar el taulell sencer) i tot seguit posar a 0 el punter de la RAM. Finalment, només caldrà tornar a mostrar la matriu sencera actualitzada amb la nova coordenada.

```
ESCRIPTURA_VALORS_F1        ;CRIDEM LA FUNCIÓ QUAN REBEM EL SENYAL NewCoord

    CALL COORD_F1

    MOVFF RAM_COORD, FSR0L      ;ASSIGNEM EL VALOR AL PUNTER DE LA RAM

    MOVLW .1
    MOVWF INDF0,0               ;POSEM DE COLOR VERD LA POSICIÓ AL TAULELL

    CLRF FSR0L, 0               ;POSEM EL PUNTER DE LA RAM A 0

    CALL LECTURA_VALORS         ;MOSTRAR ACTUALITZAR LA MATRIU

    RETURN
```

OBTENCIÓ NEW COORD

Per a l'obtenció d'una nova coordenada haurem de llegir el PORTC, ja que és el port que nosaltres hem configurat per a rebre les coordenades de la Fase 1.
Rebrem la coordenada X i Y en un conjunt de 8 bits, i per tant ho haurem de separar individualment. Per fer-ho nosaltres sabem que els 4 MSB són la coordenada X, mentre que els 4 LSB són la coordenada Y, per tant, primer de tot guardarem el valor del PORTC a l'acumulador, el multiplicarem

per una màscara d'1's als 4 LSB i 0 als 4 MSB i així obtindrem la coordenada Y que la guardarem a la variable COORD_Y.
Per a la coordenada X guardem el valor del PORTC a una variable, i aquesta la rotarem 4 vegades per obtenir la coordenada X als 4 LSB. Seguidament, copiarem aquesta variable a l'acumulador, aplicarem la mateixa màscara i obtindrem així la coordenada X que la guardarem a la variable COORD_X.
Finalment, convertirem aquesta nova coordenada a la posició corresponent de la RAM cridant la funció CONVERT_COORD_TO_RAM.

```
COORD_F1
    MOVF PORTC, 0           ;POSEM EL VALOR DEL PORT C AL REGISTRE WREG
    ANDLW 0x0F              ;MULTIPLIQUEM ELS 4 LSB PER 1 I ELS 4 MSB PER 0
    MOVWF COORD_Y, 0        ;GUARDEM EL RESULTAT A COORD_Y

    MOVFF PORTC, COORD_X    ;MOVEM EL VALOR DEL PORT C A LA VARIABLE COORD_X
    rlncf COORD_X,1,0       ;ROTEM 4 VEGADES EL VALOR
    rlncf COORD_X,1,0
    rlncf COORD_X,1,0
    rlncf COORD_X,1,0
    MOVF COORD_X, 0, 0      ;COPIEM EL VALOR AL REGISTRE WREG
    ANDLW 0x0F              ;MULTIPLIQUEM ELS 4 LSB PER 1 I ELS 4 MSB PER 0
    MOVWF COORD_X, 0        ;GUARDEM EL RESULTAT A COORD_X

    CALL CONVERT_COORD_TO_RAM

    RETURN
```

CONVERSIÓ COORDENADA A POSICIÓ MATRIU

Per a la conversió de la coordenada a la seva respectiva posició en la memòria RAM ens guiarem per la següent fórmula: Adreça RAM = ((coord_Y - 1) * 8) + coord_X - 1
Aquesta fórmula ens donarà l'adreça de la RAM corresponent a la coordenada indicada, fent que mai es guardin dos valors en una mateixa adreça, per exemple: si coord_Y = 2 i coord_X = 1 veurem que tindrem Adreça RAM = (1 * 8) + 1 - 1 = 8, i si coord_Y = 1 i coord_X = 2 veurem que tindrem Adreça RAM = (0 * 8) + 2 - 1 = 2.

Per a la implementació de la fórmula en assembler seguirem un ordre; Primer guardarem el valor de la coordenada Y en una variable RAM_COORD, on seguidament decrementarem en una unitat el seu valor, i tot seguit ho multiplicarem per 8. Després agafarem la coordenada X i la guardarem a l'acumulador, per fer després una suma amb PRODL (variable on estarà guardat el resultat de la multiplicació) i seguidament guardar-ho a RAM_COORD. Finalment, només ens faltarà decrementar el resultat en una unitat perquè l'adreça coincideixi, i comprovar si es tracta del jugador 0 o jugador 1 mirant el *current grid*, i incrementant el resultat en 65 posicions si fos necessari.

```
CONVERT_COORD_TO_RAM

    MOVFF COORD_Y, RAM_COORD
    DECF RAM_COORD, 1, 0

    MOVLW .8
    MULWF RAM_COORD,0

    MOVF COORD_X, 0
    ADDWF PRODL, 0, 0
    MOVWF RAM_COORD, 0
    DECF RAM_COORD, 1, 0

    MOVLW .65
    BTFSC CURRENT_GRID, 0, 0
    ADDWF RAM_COORD, 1, 0

    RETURN
```

Finalització del joc

Aquesta part del codi es cridarà quan la vida de la flota d'un jugador arribi a 0, és a dir, que el PWM del servo d'algun jugador sigui 0 ms.
Un cop hagi estat cridada la funció, doncs, ens tocarà posar la melodia de final de joc, que consisteix en 5 notes diferents durant 5 segons mitjançant l'altaveu. Per a cada nota configurarem un valor, anant del més greu fins a l'agut, i per cada nota cridarem la funció SOROLL_ALTAVEU, mencionada anteriorment, i cada nota sonarà durant un segon.
Un cop fetes les 5 notes, entrarem en un bucle final, on estarem ancorats allà fins que es premi el botó de PCI, fent que es comenci una nova partida.

```
END_GAME
    CLRF COMPT_1S,0         ;FAREM SONAR LES 5 NOTES DE L'ALTAVEU UN COP ACABEM
    MOVLW .4
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU     ;1 NOTA
    CLRF COMPT_1S,0
    MOVLW .10
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU     ;2 NOTA
    CLRF COMPT_1S,0
    MOVLW .30
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU     ;3 NOTA
    CLRF COMPT_1S,0
    MOVLW .50
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU     ;4 NOTA
    CLRF COMPT_1S,0
    MOVLW .80
    MOVWF   SOROLL_FREQ,0
    CALL SOROLL_ALTAVEU     ;5 NOTA
END_BUCLE                   ;DEIXEM EL PROGRAMA EN AQUEST BUCLE FINS QUE ES
    CLRF COMPT_1S,0         ;TORNI A APRETAR PCI
    GOTO END_BUCLE
```

3. Configuracions del microcontrolador

Oscil·lador

El primer de tot és saber a quina freqüència volem treballar amb el PIC, tenim 2 components que els hi hem de generar PWM, són els Servomotors i la matriu de LEDs.

La matriu de LEDs és la que necessita un temps d'instrucció més ràpida, per aquest motiu utilitzem l'oscilador extern de 10MHz amb PLL per obtenir 40MHz, amb això tenim un temps d'instrucció de 100ns i podem generar els polsos amb les durades que necesita.

```
    CONFIG OSC = HSPLL          ;utilitzem XSTALL extern de 10MHz amb PLL per anar a 40MHz
```

Interrupcions

Tenim 3 senyals que ens fan saltar la interrupció. No fem ús de les prioritats, és a dir que totes les interrupcions són altes.

La primera és la del Timer 0, per a poder controlar els servomotors hem de generar 2 PWM pels 2 servomotors constantment , això no ho podem fer en el nostre programa principal, ja que estarem tot el temps fent el PWM i no faríem res més.
Per aquest motiu necessitem una interrupció que ens avisi de tant en tant per a fer els PWM i així poder fer altres coses.
Aquesta interrupció salta quan el Timer 0 ha comptat fins al nombre màxim i hi ha hagut un overflow.

Les dues altres són externes i per flanc, l'INT0 i l'INT1 que pertanyen al RB0 i RB1 respectivament.
En aquests pins hi tenim connectats els senyals que rebem de la fase 1 que són New Coord i New Boat, com que són senyals que van per flanc, per a gestionar-les correctament utilitzem aquestes interrupcions.
Com que la fase 1 va a una freqüència de 1KHz i en la fase 2 anem a 10MHz en temps d'instrucció, mitjançant Pooling tenim temps de sobres per a interceptar aquests senyals, però per ser correctes utilitzem interrupcions.

L'INT0 l'hem configurat com a flanc de baixada pel New Coord i el INT1 com a flanc de pujada pel New Boat.

```
;-------------------------------------------------------------------------------
; 3. INTERRUPCIONS
;-------------------------------------------------------------------------------
CONFIG_INT
    BCF     RCON,IPEN,0                 ;desactivo prioritats sempre per tenir nomes la alta
    BSF     INTCON,GIE,0                ;activem les interrupcions internes
    BSF     INTCON,PEIE,0               ;activem masked i unmasked interrupts sempre
    BCF     INTCON,TMR0IE,0             ;de moment desactivem la interrupcio del timer0
    BSF     INTCON,INT0IE,0             ;activem interrupcio RB0

    BCF     INTCON2,INTEDG0,0           ;INT0 per flanc de baixada
    BSF     INTCON2,INTEDG1,0           ;INT1 per flanc de pujada

    BSF     INTCON3,INT1IE,0            ;activem interrupcio RB1
```

Quan comença l'execució en el main, el primer que fem és fer les configuracions dels ports, ram i interrupcions. Dintre de les configuracions de les interrupcions el que fem és treure les prioritats, i llavors activar les internes i externes.
Durant el temps que estem rebent valors de la fase 1, no ens interessa controlar els servomotors, així que la interrupció del Timer 0 la deixem desactivada, però, en canvi, sí que volem les INT0 i INT1.
Quan acabem d'introduir les coordenades i rebem Start Game llavors sí que activem la interrupció del timer 0.

Ports d'entrada

A part dels senyals New Coord i New Boad que per força han d'anar amb el RB0 i RB1 perquè són d'Interrupció, les altres entrades poden anar en qualsevol dels altres pins.

Polsadors Moviment Up, Down, Right i Left i Attack:
Aquests hem intentat posar-los tots en el Port B per aprofitar els Pull-Ups interns que tenen aquest port, en principi els hem posat de l'RB3 a l'RB7, però els 2 últims ens donaven problemes, ja que s'utilitzen per al port de programació del PIC per això hem acabat posant aquests dos al RD1 i RD3.

Start Game i Current Player:
Els tractem com a senyal de nivell, per això els hem connectat en 2 pins qualssevol, el Start Game al RD2 i el Current Player al RB2.

Boat Coords:
Per tractar aquest senyal de 8 bits còmodament hem ocupat un port sencer, per això l'hem connectat al Port C.

Ports de Sortida

Hem utilitzat tot el port com a senyals de sortida excepte l'RA6 i RA7 que és on va connectat l'oscil·lador, i ens faltava un pin més de sortida que l'hem connectat al RD0.

```
;-------------------------------------------------------------------------------
; 1. CONFIGURACIO DE PORTS
;-------------------------------------------------------------------------------

CONFIG_PORTS

CONFIG_PWM_PORTS
    BCF     TRISD,RD0,0                 ;Grid
    BSF     TRISD,RD2,0                 ;StartGame
    BCF     TRISA,RA1,0                 ;FleetStatus0
    BCF     TRISA,RA2,0                 ;FleetStatus1
    BCF     TRISA,RA3,0                 ;ACK
    BCF     TRISA,RA4,0                 ;CurrentGrid
    BCF     TRISA,RA5,0                 ;Speaker
CONFIG_BUTTONS
    SETF TRISB, 0               ;0-NewCoord, 1-NewBoat, 3-Up, 4-Left, 5-Down
    BSF TRISD,RD1,0                     ;Right
    BSF TRISD,RD3,0                     ;Attack
    BCF     INTCON2,RBPU,0              ;activem pull ups al port B
CONFIG_COORDS
    SETF TRISC, 0       ;inicialitzem tot el port C com a entrada
    RETURN
```

En la configuració dels ports el que fem és mitjançant el registre TRIS del port que ens interessa indicar que sigui d'entrada amb un Set, o de sortida amb un Clear.
I al registre INTCON2 posant a 0 el bit RBPU activem els pull-ups.

Timer 0

Com hem mencionat en l'apartat d'interrupció, hem utilitzat el Timer 0 perquè ens faci saltar la interrupció per controlar els servomotors. Es configura amb el registre T0CON on podem dir si volem que sigui un comptador de 8bits o 16, i si volem Prescaler.

Els PWM dels Servomotors s'han de generar cada 20Hz, amb una periodicitat de 20ms.
És a dir que si com a molt volguéssim fer saltar la interrupció cada 20ms, en temps d'instrucció necessitaríem comptar fins a 200.000, ja que anem a 100ns de temps d'instrucció.

Amb 16 bits podem comptar fins a 65536, és a dir que necessitem Prescaler, hem utilitzat prescaler de 4 d'aquesta manera en lloc de 200.000 serien 50.000.

```
MOVLW   b'10000001'
MOVWF   T0CON,0
RETURN
```

En la configuració simplement indiquem que volem 16 bits, sí que volem PLL i que ha de ser de 4.

RAM

Per a la RAM, hem fet ús de dos registres que fan de punter per apuntar a una regió de la memòria, que són el FSR0H i FSR0L. Tenim disponibles 3 punters, però només utilitzem un.

Podem accedir a 16 bancs de 256 adreces cadascuna. Però entre totes aquestes només podem utilitzar el Banc 1 sense i la meitat inferior del Banc 0 i la meitat superior del Banc 15.

Nosaltres per a guardar la informació dels taulells utilitzem el Banc 1 de l'adreça 0 a la 128.
Per aquest motiu en la configuració que fem de la RAM inicialment, carregem un 1 al FSR0H per indicar el Banc 1 i llavors amb el FSR0L anem recorrent la RAM de la 0 al 128 posant a 0 el seu contingut mitjançant el POSTINC0 que no és un registre sinó una manera de dir-li al PIC que llegeixi o escrigui a la regió de la memòria on apunta el punter.

```
INIT_RAM
    MOVLW 0x01
    MOVWF FSR0H, 0 ;Indiquem que volem utilitzar el banc 1.

    CLRF FSR0L, 0 ;Comencem en l'adressa 0 del banc 1
    CALL RESET_RAM
    RETURN

RESET_RAM
    MOVLW .0
    MOVWF POSTINC0, 0
    MOVLW .129
    CPFSEQ FSR0L, 0
    GOTO RESET_RAM
    CLRF FSR0L, 0
    RETURN
```

FLASH

La taula flash no és una configuració que es faci en temps d'instrucció sinó que es genera en temps de compilació.
Hem creat una taula on tenim els temps que carregem al timer 0 depenent del número de coordenades introduides.
Abans de començar amb el joc de la fase 2, mirem quantes coordenades ens ha arribat de la fase 1 i llavors llegim de la flach els temps que l'hi pertoca i els guardem amb 2 variables que utilitzem al llarg de la fase 2, que són TEMPS_INC_H i TEMPS_INC_L, perquè el temps és de 16 bits per aixo necesitem 2 posicions a la flash per valor.

```
;-------------------------------------------------------------------------------
; TAULES DE LA FLASH
;-------------------------------------------------------------------------------
TAULASERVO EQU 0x20             ;definim taula de la flash per guardar els increments dels servos

   ORG TAULASERVO
   DB b'11111100',b'00011000'          ; Valor del comptador per tindre 5 increments   0x20,0x21
   DB b'11111100', b'10111111'         ; Valor del comptador per tindre 6 increments   0x22,0x23
   DB b'11111101', b'00110110'         ; Valor del comptador per tindre 7 increments   0x24,0x25
   DB b'11111101', b'10001111'         ; Valor del comptador per tindre 8 increments   0x26,0x27
   DB b'11111101', b'11010100'         ; Valor del comptador per tindre 9 increments   0x28,0x29
   DB b'11111110', b'00001100'         ; Valor del comptador per tindre 10 increments  0x2A,0x2B
   DB b'11111110', b'00111001'         ; Valor del comptador per tindre 11 increments  0x2C,0x2D
   DB b'11111110', b'01011111'         ; Valor del comptador per tindre 12 increments  0x2E,0x2F
   DB b'11111110', b'01111111'         ; Valor del comptador per tindre 13 increments  0x30,0x31
   DB HIGH(.65179), LOW(.65179)        ; Valor del comptador per tindre 14 increments  0x32,0x33
   DB HIGH(.65203), LOW(.65203)        ; Valor del comptador per tindre 15 increments  0x34,0x35
   DB HIGH(.65224), LOW(.65224)        ; Valor del comptador per tindre 18 increments  0x36,0x37
   DB HIGH(.65273), LOW(.65273)        ; Valor del comptador per tindre 16 increments  0x38,0x39
   DB HIGH(.65242), LOW(.65242)        ; Valor del comptador per tindre 17 increments  0x3A,0x3B
   DB HIGH(.65258), LOW(.65258)        ; Valor del comptador per tindre 19 increments  0x3C,0x3D
   DB HIGH(.65286), LOW(.65286)        ; Valor del comptador per tindre 20 increments  0x3E,0x3F
   DB HIGH(.65298), LOW(.65298)        ; Valor del comptador per tindre 21 increments  0x40,0x41
   DB HIGH(.65309), LOW(.65309)        ; Valor del comptador per tindre 22 increments  0x42,0x43
   DB HIGH(.65319), LOW(.65319)        ; Valor del comptador per tindre 23 increments  0x44,0x45
   DB HIGH(.65328), LOW(.65328)        ; Valor del comptador per tindre 24 increments  0x46,0x47
   DB HIGH(.65336), LOW(.65336)        ; Valor del comptador per tindre 25 increments  0x48,0x49
```

Els valors de la taula els hem calculat de la següent manera:

Nº Vaixells	Increment º	Incr/4	65536-incr	Nº Instrucions de 0,2ms:	20.000
5	4000	1000	64536		
6	3333,33333	833,333333	64703		
7	2857,14286	714,285714	64822		
8	2500	625	64911		
9	2222,22222	555,555556	64980		
10	2000	500	65036		
11	1818,18182	454,545455	65081		
12	1666,66667	416,666667	65119		
13	1538,46154	384,615385	65151		
14	1428,57143	357,142857	65179		
15	1333,33333	333,333333	65203		
16	1250	312,5	65224		
17	1176,47059	294,117647	65242		
18	1111,11111	277,777778	65258		
19	1052,63158	263,157895	65273		
20	1000	250	65286		
21	952,380952	238,095238	65298		
22	909,090909	227,272727	65309		
23	869,565217	217,391304	65319		
24	833,333333	208,333333	65328		
25	800	200	65336		

El que ens interessa és la regió dels 2ms que hi ha entre 0,5ms-2,5ms. És la que determina l'angle del servomotor.

Per aquest motiu hem dividit aquests 2ms pel nombre de coordenades que tenim guardats.

Primer hem calculat el temps d'instrucció que equivalen 2ms, que són 20.000 instruccions.

Aquest temps el dividim pel nombre de coordenades que pot ser mínim 5 si tots els vaixells són de mida 1 i màxim 25 si tots els vaixells són de mida 25.

Llavors després de fer la divisió, tornem a dividir aquest valor entre 4 perquè tenim activat el PLL del timer0. I finalment 65536 l'hi restem aquest valor i obtenim el que li hem de carregar al timer 0 per a comptar aquest temps d'increment.

Finalment per a carregar el valor que toca de la flash ho fem de la següent manera:

```
INIT_INTERUPT_VARIABLES
    MOVLW 0x00 ; Load TBLPTR with the base
    MOVWF TBLPTRU ; address of the word
    MOVLW 0x00
    MOVWF TBLPTRH

    MOVLW .5
    SUBWF MAX_PWM,0,0
    MULLW .2
    MOVF PRODL,0
    ADDLW TAULASERVO
    MOVWF TBLPTRL,0
READ_WORD
    TBLRD*+ ; read into TABLAT and increment
    MOVF TABLAT, 0 ; get data
    MOVWF TEMPS_INC_H,0
    TBLRD* ; read into TABLAT
    MOVF TABLAT, 0 ; get data
    MOVWF TEMPS_INC_L,0

    CLRF ACTUAL_PWM,0
    INCF MAX_PWM,1,0
    MOVFF   MAX_PWM, PWM_S0
    MOVFF   MAX_PWM, PWM_S1

    RETURN
```

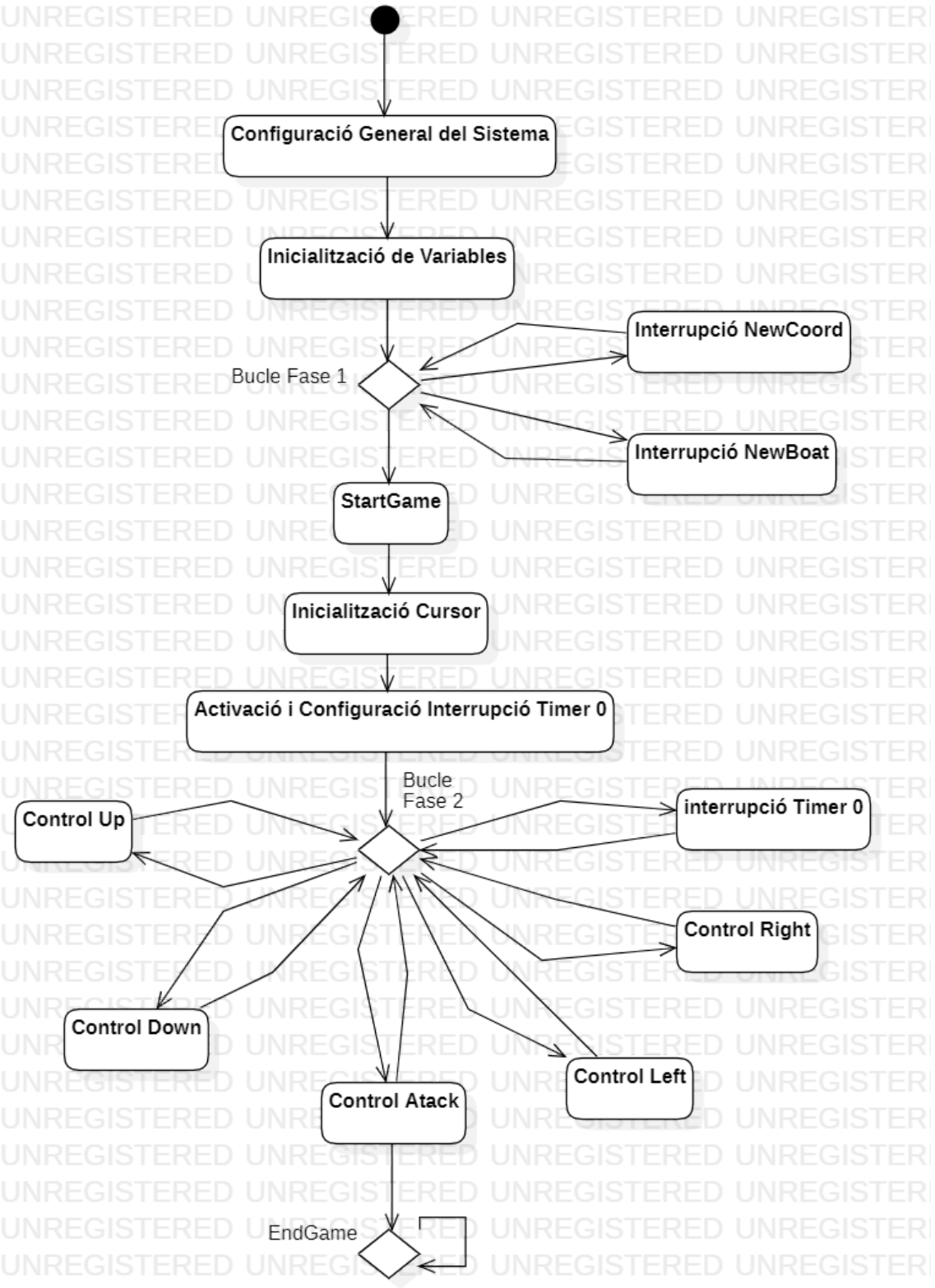
La flash té un punter de 3 registres TBLPTR, l'Uper, High i Low. L'Uper i High els posem a 0.
La nostra taula flash la tenim declarada a partir de l'adreça 0x20. Per aquest motiu només utilitzem el TBLPTRL.

A la variable MAX_PWM hi tenim el nombre de coordenades que ens han introduït a la fase 1. Fem un de la següent fórmula per a obtenir el temps que l'hi pertoca a la flash:
Coordenada Flash = (MAX_PWM - 5)*2 + 20.

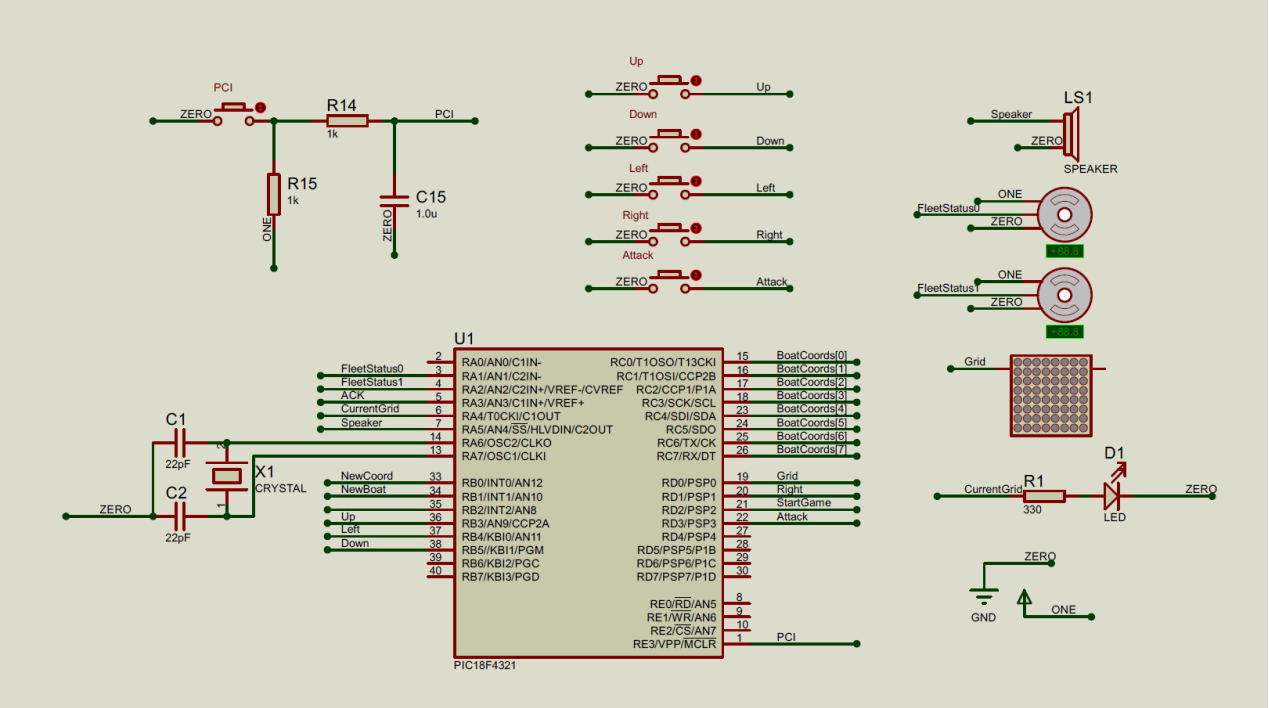
Exemple:
Si MAX_PWM = 5
Llavors volem els valors que estan a la posició 0x20 i 0x21 veiem com es compleix (5-5)*2+20=20.
Si MAX_PWM = 25
Llavors volem els valors que estan a la posició 0x48 i 0x49, (25-5)*2+20=60 que convertit en hexadecimal equival a 0x48.

Finalment utilitzant la Instrucció TBLRD*+ llegim la primera adreça i incrementem el punter, guardem el valor a TEMPS_INC_H des del registre TABLAT que és on es guarda el contingut de l'adreça de la taula en fer TBLRD. El següent valor el guardem a TEMPS_INC_L.

4. Diagrama d'activitat del software



5. Esquema elèctric



6. Problemes observats

Els principals problemes que hem tingut en aquesta fase han estat respecte a les connexions amb la fase 1.

Per començar, teníem un error que ens va aparèixer de nou, que ja vàrem solucionar a la fase 1 en el seu dia, però ens va tornar a sortir. L'error en qüestió era del nombre aleatori, ja que el led central no es mostrava correctament, tot i arribar bé el senyal als xips que el generaven. Per tal que funciones bé, aquesta vegada vàrem afegir un condensador per a estabilitzar el senyal del canvi del nombre aleatori, i així ho vàrem poder arreglar.

També vàrem trobar errors a l'hora de passar senyals de la fase 1 a la 2, ja que alguns cables no feien bé el contacte i els vàrem haver de substituir.

Un altre problema ha estat que nosaltres vàrem plantejar els servos amb una interrupció que entrava i sortia constantment per optimitzar el codi, però ens varen dir que no ho podíem fer així, llavors vàrem canviar el codi per fer que interrumpeixi els 2,5 primers mil·lisegons i llavors els 17,5 restants que els fes el codi ell sol fins que passats 20 ms, torni a interrompre.

7. Conclusions

Un cop acabada la pràctica 1 fase 2, hem estat capaços d'implementar la nostra solució en assembler i que funciones amb les especificacions demanades, i sobretot acabar-la en el termini que ens esperàvem, ja que per a nosaltres era una prioritat acabar abans de les vacances de Nadal.

8. Planificació

Hores estimades i planificació:

	Hores estimades	13/11/23	20/11/23	27/11/23	4/12/23	11/12/23	18/12/23	25/12/23
Comprensió de l'enunciat	1							
Disseny sobre paper	5							
Programació MPLAB	15							
Soldar	6							
Debugar	8							
Realització de la memòria	5							

Hores totals dedicades:

	Hores estimades	13/11/23	20/11/23	27/11/23	4/12/23	11/12/23	18/12/23	25/12/23
Comprensió de l'enunciat	1							
Disseny sobre paper	6							
Programació MPLAB	20							
Soldar	4							
Debugar	5							
Realització de la memòria	7							

Per a la realització de la pràctica vàrem estimar unes hores per a la part del disseny i programació en MPLAB, que al final, un cop feta la pràctica han resultat estar més de les que havíem planificat, com que a mesura que anàvem dissenyant anàvem programant i provant el que fèiem, d'aquí que al final hagi resultat estar més de l'esperat.

Per la part de soldar i debugar, en canvi, vàrem estimar més temps del que realment ha estat, tot i que no difereix tant del que ja vàrem pensar.

Finalment per a la memòria hem estat més estona de la que pensàvem, ja que ho hem volgut explicar tot al detall i no deixar-nos res.