Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Microprocesadores

IE0623

Proyecto Final

Autores:
Robin González Ricz B43011
Michelle Gutierrez B43195

Profesor:
Esteban Badilla

28 de febrero de 2021

Índice

Índice de figuras

1. Resumen

Este proyecto consiste en el diseño e implementación de una aplicación que despliega información de un velódromo, llamado Run Meter 623. Se tienen 4 modos de funcionamiento: libre, resumen, configuración y competencia. Al iniciar la aplicación, estará primero en modo configuración por defecto y se podrá cambiar de modo únicamente al ingregar un valor de número de vueltas que se encuentre en el rango de 3 a 23. En el modo competencia, se tienen dos sensores S1 y S2, que corresponden a los botones PH3 y PH0 de forma correspondiente. Por medio de estos sensores, que detectan en que momento un ciclista pasa por la pista, se calcula la velocidad que lleva el ciclista y se cuenta una vuelta, para que este mecanismo funcione siempre se debe activar el sensor S1 y luego el sensor S2, cualquier otra combinación en el orden de activación de los sensores será ignorada. Cuando el número de vueltas alcanza el número que fue programado en el modo Configuración, los sensores se deshabilitan. En el modo Resumen, se despliega información acerca de la cantidad de vueltas y la velocidad promedio, según se dió en el modo Competencia. Por aspectos de diseño, para poder acceder a la información del Resumen, se debe pasar a este modo justo después del modo Competencia, sino se hace de esta forma, los datos serán borrados. En cualquier momento, se puede volver al modo Configuración y programar un valor de vueltas nuevo. El modo Libre no despliega información, más que el mensaje de que se encuentra en ese modo. Existe un rango de velocidad válida que va de 35 Km/h a 95 Km/h, en caso de que la velocidad del ciclista no se encuentre en este rango, no se contara una nueva vuelta, no se considera la velocidad para el cálculo de la velocidad promedio tampoco e inmediatamente se desplegará un mensaje de Alerta de 3 s. En el modo Competencia, mientras no se active el primer sensor, se desplegará un mensaje que indica que se encuentra en espera, en el momento que el primer sensor es activado, el mensaje desplegado indica que se está calculando, una vez activado el segundo sensor, este mensaje no cambia, hasta que haya transcurrido un tiempo en el que el ciclista se encuentre a unos 100 m de la pantalla que muestra la información, en este momento se despliega la información de vueltas y velocidad, luego vuelve al mensaje de esperando, hasta repetir el ciclo. Solo se puede estar en un modo a la vez y se asume que solo puede haber un ciclista en la pista a la vez.

2. Diseño de la aplicación

2.1. Estructuras de datos

Banderas: Modo1:Modo0:Calc_TICKS:CambMod:Pant_FLAG:ARRAY_OK:TCL_LEIDA:TCL_
YULS: apagarBIN2:x:velocidadValida:Direccion:PantFLG:HabraCalculo:x:x

NumVueltas: Byte, Cantidad de vueltas a contar.
ValorVueltas: Byte, Vueltas ingresadas en configuración.
MAX_TCL: Byte, Se define el valor maximo del arreglo de teclas.
Tecla: Byte, Se utiliza en las subrutinas de la tarea 4.
Tecla_IN: Byte, Tecla que se ingreso en la iteracion anterior.
Cont_Reb: Byte, Contador para rebotes al leer botones.
Cont_TCL: Byte, Contador de teclas ingresadas.
Patron: Byte, Patron para revisar el teclado matricial.
Num_Array: Arreglo de teclas ingresadas.
BRILLO: Byte, Variable para controlar brillo de leds y 7 segmentos.
POT: Byte, Variable de lectura del potenciometro.
TICK_EN: Word, Contador word para poner PantFLG.
TICK_DIS: Word, Contador word para borrar PantFLG.
Veloc: Byte, Byte para guardar velocidad actual.
Vueltas: Byte, Variable para guardar cantidad de vueltas dadas.
VelProm: Byte, Guarda promedio de velocidad.
TICK_MED: Word, Word para contar tiempo entre presion de ph3 y ph0.
BIN1: Byte, Número para poner en derecha de pantalla en binario.
BIN2: Byte, Número para poner en izquierda de pantalla en binario.
BCD1: Byte, Número para poner en derecha de pantalla en bcd.
BCD2: Byte, Número para poner en izquierda de pantalla en bcd.
BCD_L: Byte, Byte utilizado para guardar conversion de binario a bcd.
BCD_H: Byte, No fue usado.
TEMP: Byte, Variable temporal para cálculo a bcd.
LOW: Byte, Usado en bin bcd como variable temporal.
DISP1: Byte, Valor en 7 seg a poner en disp1.
DISP2: Byte, Valor en 7 seg a poner en disp2.
DISP3: Byte, Valor en 7 seg a poner en disp3.
DISP4: Byte, Valor en 7 seg a poner en disp4.
LEDS: Byte, Valor a poner en las 8 leds.
CONT_DIG: Byte, Contador de digitos/leds que enciende oc4.

CONT_TICKS: Byte, Contador de 100 mS en OC4.
DT: Byte, Variable innecesaria para brillo.
CONT_7SEG: Word, Contador de 5000 para cambiar valores en pantalla.
CONT_200: Byte, Cantidad de entradas en RTI para activad ADT0.
Cont_Delay: Byte, Contador delays en oc4.
D2mS: Byte, Cantidad de entradas en oc4 para esperar 2 ms.
D260uS: Byte, cantidad de entradas en oc4 para esperar 260 us.
D40uS: Byte, cantidad de entradas en oc4 para esperar 40 us.
Clear_LCD: Byte, Comando para borrar pantalla LCD.
ADD_L1: Byte, Comando para direccionar LCD a linea 1 en memoria.
ADD_L2: Byte, Comando para direccionar LCD a linea 1 en memoria.
Teclas: Arreglo con los valores de Tecla.
SEGMENT: Arreglo con valores en 7 seg para cada numero, ordenado por indice, pos 10 = - , pos 11 = apagado.
initDisp: Arreglo con comandos para inicializacion de la LCD.
MSGConfig_L1: Mensaje para configuración, Línea 1.
MSGConfig_L2: Mensaje para configuración, Línea 2.
MSGINICIAL_L1: Mensaje inicial de esperando, Línea 1.
MSGINICIAL_L2: Mensaje inicial de esperando, Línea 2.
MSGCOMPETENCIA_L1: Mensaje de competencia, Línea 1.
MSGCOMPETENCIA_L2: Mensaje de competencia, Línea 2.
MSGCALCULANDO_L1: Mensaje de calculando, Línea 1.
MSGCALCULANDO_L2: Mensaje de calculando, Línea 2.
MSGALERTA_L1: Mensaje de alerta, Línea 1.
MSGALERTA_L2: Mensaje de alerta, Línea 2.
MSGRESUMEN_L1: Mensaje de resumen, Línea 1.
MSGRESUMEN_L2: Mensaje de resumen, Línea 2.
MSGLIBRE_L1: Mensaje de modo libre, Línea 1.
MSGLIBRE_L2: Mensaje de modo libre, Línea 2.

2.1.1. Variables adicionales usadas

YULS: Variable de banderas, en lugar de TEMP1.
CURIE: Variable word para guardar primer parte de calculo de promedio, en lugar de TEMP2.
CURIE2: Variable word para guardar primer parte de calculo de promedio.
HZD: Usado para contador de 200 para ATD.

Sobró un byte adicional de los disponibles que no fue usado.

2.2. Configuración de hardware

A continuación se detallan los valores escritos en los registros de periféricos y su por qué. La explicación de cómo configurar cada periférico es dada por Tom Almy en su [1].

DDRH.7.6 <- 0: de esta manera configuramos los bits 7 y 0 como entradas en el puerto H. Nótese que esto es de todas formas redundante pues al encender este puerto es por defecto una entrada, así que este comando no es siquiera necesario pero se pone para recordar que esos bits serán entradas.

PIEH.3.0 <- 1: con este comando habilitamos las interrupciones key-wakeups del puerto H.

PPSH.3.0 <- 1: al haber encendido las interrupciones en estos pads, este comando indica que la interrupción se activará por flanco creciente, ya que se colocan resistencias de *Pull-Down*.

TSCR1.8 <- 1: pone 1 en TEN: Timer Enable, enciende el Timer Counter de 16 bits del MCU. TSCR2.7.1.0 <- 1: pone 1 en TOI: habilita Timer Overflow Interrupt, pone 3 en PRS <- selecciona prescalador de 8.

TIOS.4 <- 1: habilita la salida en el canal 4 del timer.

TIE.4 <- 1: habilita la interrupción en el canal 4, al darse igualdad entre su contenido y el contenido del TCNT.

LDD #60
ADDD TCNC
STD TC4: estas instrucciones ponen en el registro del canal 4, el valor actual en el timer + 60.

DDRB <- $FF: ponemos todo el puerto B como salida. De esta manera podremos poner valores en los 8 leds rojos o en la pantalla de 7 segmentos.

DDRJ.1 <- 1: ponemos 1 en el pad 2 del puerto J para usarlo como salida, de esta forma podremos poner valores en este pad, este pad es el cádoto común de las

8 leds.

PTJ.2 <- 1: ponemos 1 en el pad anterior, de esta manera apagamos las 8 leds.

DDRP.[3:0] <- $F: ponemos 1 en el nibble inferior de P, de esta manera habilitamos como salida los pads del puerto P que funcionan como cátodo común para cada uno de los 4 dígitos de la pantalla de 7 segmentos.

DDRK <- $FF: ponemos el puerto K como salida, para comunicarnos con la LCD.

DDRA <- $F0: ponemos el nibble superior del puerto A como salida y el inferior como entrada. Esto es para escribir en las filas y leer las columnas del teclado matricial.

PUCR.0 <- 1:habilitamos las resistencias de *Pull-Up* en el puerto A.

ATD0CTL2.7.6.1 <- ponemos 1 en ADPU,AFFC y ASCIE. Encendemos ATD, permitimos el borrado rápido de banderas el leer los registros del ATD y habilitamos las interrupciones al terminar un ciclo de conversión.

LDAA #180
DBNE A,esperar: estas dos instrucciones esperan en total de 180 mS por si acaso algo más de los 160 mS mínimos para endender el ATD.

ATD0CTL3 <- $03 : apagamos el FIFO y hacemos que el ciclo de conversiones sea de 6.

ATD0CTL4 <- $B3 :ponemos la resolución en 8 bits y que se tome 4 ciclos de reloj por cada una, con el prescalador en 20.

ATD0CTL5 <- $87 :justificamos el resultado a la derecha, sin signo y controlado por software el encendido, sin usar el MUX y que sea el canal 7, donde está conectado el potenciómetro.

CRGINT.7 <- 1: encendmos la interrupción RTI.

RTICTL <- $17: ponemos 1 y 7 como valores para calcular el periódo de interrupción de RTI.

DLS #$3BFF colocamos el stack pointer en la última posición de memoria RAM disponible. No puede ser $4000 pues a partir de $3E00 se encuentran relocalizados los vectores de interrupción de FLASH a RAM por el DBUG12.

CLI: clear interrupt flag: habilita interrupciones mascarables.

2.2.1. Memorias de cálculo

2.2.2. RTI

Para obterner un perioro de 1 ms en la interupción Real Time Interrupt:
M = 1
N = 7
En la dragon el Osc_CLK es 8 MHZ

$$T_{RTI} = \frac{(N+1)\cdot 2^{(M+9)}}{Osc\_CLK} = \frac{8x2^{10}}{8\cdot 10^6} = 1{,}024\cdot 10^{-3}s \tag{1}$$

Por lo cual hay un error, debido a la baja granularidad en configuracion de los tiempos de interrupción, dado en la ecuación 2.

$$\left|\frac{V_{REAL} - V_{APROXIMADO}}{V_{REAL}}\right| * 100 = \left|\frac{0{,}001 - 0{,}001024}{0{,}001}\right| * 100 = 2{,}4\,\% \tag{2}$$

2.2.3. TOI

Para calcular el periodo de la interrupción Timer Overflow Interrupt, debemos usar la ecuación 3.

$$T_{TOI} = \frac{PRS\cdot 2^{16}}{BUS\_CLK} \tag{3}$$

Con el PRS configurado en 8. Y el BUS_CLK de al Dragon que es 24 MHz.

$$T_{TOI} = \frac{8\cdot 2^{16}}{24\cdot 10^6} \tag{4}$$

Entonces de la euación 4 obtenemos que el periodo será 21.8453 milisegundos. Como se pedía un periódo de 20 milisegundos entonces tenemos un error, dado en la

ecuación 5.

$$\left|\frac{V_{REAL} - V_{APROXIMADO}}{V_{REAL}}\right| * 100 = \left|\frac{0{,}02 - 0{,}0218453}{0{,}02}\right| * 100 = 9{,}2265\,\% \qquad (5)$$

2.2.4. OC4

Para obtener el preiodo de la interrupción Output Compare se toma en cuenta la ecuación 7. Como deseamos un periodo de interrupción de 10 Hz por dígito/-leds y queremos que cada dígito/leds se barra 100 veces, en realidad el periodo de interrupción total es dad0 por la ecuación 6.

$$\frac{1}{500 \cdot 100} = 20 \cdot 10^{-6} s \qquad (6)$$

El prescalador anteriormente se había seleccionado en 8, $PRS = 8$.

$$T_{OC} = \frac{PRSxTC_n}{Bus\_Clk} \qquad (7)$$

Entonces, el valor a colocar en T_{C4} es dado por la ecuación 8.

$$T_{C4} = \frac{T_{OC} Bus\_Clk}{PRS} = \frac{20 \cdot 10^{-6} \cdot 24 \cdot 10^{6}}{8} = 60 \qquad (8)$$

Debido a la alta granularidad en la configuración del contador el error $= 0\,\%$.

2.2.5. ATD0

Lost ATD presentes en el microcontrolador MC9S12DG256B requieren de 10 microsegundos para encender. Y según la hoja de instrucciones de la arquitectura la instrucción DBNE toma 3 ciclos. Por lo cual si cargamos 160 en el registro A, con un core clock de 48 MHz, esto se tomará 10 micro segundos. La comprobación se observa en la ecuación 9.

$$\frac{3 \cdot 160}{48 \cdot 10^{6}} = 10\mu S \qquad (9)$$

También debemos configurar el prescalador del ARD, que define la velocidad de las lecturas. Para esto se utiliza la ecuación 10.

$$PRS = \frac{BUS\_CLK}{2 \cdot fs} \qquad (10)$$

Con los datos de la dragon, y la frecuencia de muestreo de 600 KHz.

$$PRS = \frac{24 \cdot 10^6}{2 \cdot fs} = 20 \tag{11}$$

Entonces en la ecuación 11 obtenemos que valor del prescalador del ATD0.

2.2.6. Ecuaciones para calcular la velocidad del ciclista

Sabemos que los sensores están separados por 55 m y queremos obtener una velocidad en Km/h. De esta manera solo tenemos que contador los segundos transcurridos entre que el ciclista pasa el sensor S1 y luego el sensor S2. Pero para medir este tiempo utilizamos la subrutina TCNT que tiene el contador TICK_MED que aumenta cada periodo de interrupción (21.8453 mS). Como ya sabemos desde el colegio la ecuación 12 es todo lo que necesitamos.

$$v = \frac{d}{t} \tag{12}$$

Donde: v= rapidez, d= distancia, t= tiempo.
Sabemos que d = 55 m.
t estará dado por la cantidad de TICKS multiplicado por el periodo de interrupción de TOI, el cual es $\frac{1024}{46875}$.
Sabemos además que para convertir una velocidad de m/s Km/h se debe multiplicar por 3.6 = 18/5. Entonces deducimos la ecuación 13.

$$v = \frac{55 \cdot 18}{TICK\_MED \cdot 5 \cdot \frac{1024}{46875}} = \frac{1}{TICK\_MED} \cdot \frac{55 \cdot 18 \cdot 46875}{5 \cdot 1024} = \frac{9063{,}72}{TICK\_MED} \tag{13}$$

Esta ecuación se puede realizar con la operación IDIV de la arquitectura.

2.2.7. Ecuaciones para calcular el promedio de la velocidad del ciclista

Para calcular la velocidad promedio del ciclista simplmente se usa la encuación para este valor de estadística: suma de las muestras entre cantidad de muestras. Sin embargo en este caso tenemos el reto de que cada iteración contamos con la velocidad promedio calculada en la iteración anterior y no con el total de velocidades. Para esto se dedujo una fórmula que permitiera calcular el promedio correcto en cada ocasión, dicha fórmula se presenta en la ecuación 14.

$$\frac{VelProm \cdot (Vueltas - 1) + Velocidad}{Vueltas} \tag{14}$$

Esta ecuación se obtuvo comprendiendo la definición del promedio o media y además la de promedio ponderado, lo cual es en realidad aplicado en este caso.

Ya que se tiene un valor promediado anteriormente una simple división entre 2 no sería un promedio válido.

Se debe otorgar un peso al promedio anterior que valore la cantidad de muestras, entonces se multiplica el promedio anterior por al cantidad de muestras que representa y se le suma la muestra actual, este resultado se divide entre la cantidad de muestras actuales, y así se obtiene un promedio válido.

2.2.8. Ecuaciones para calcular el tiempo que dura la pantalla encendida

Para calcular el periodo de encendido de la pantalla LCD y de 7 segmentos tras el paso de un ciclista por los sensores, necesitamos obtener 3 valores: TICK_EN, TICK_DIS y un valor que equivalga a 3 segundos para el mensaje de alerta.

Primero debemos obtener el tiempo hasta que el ciclista llegue a 100 m antes de la pantalla, osea que ha recorrido 200 m y luego el tiempo que dura en llegar justo bajo la pantalla o sea ha recorrido 300 m.

Para el primer caso volvemos a usar la ecuación clásica 12 pero despejamos el tiempo, pues conocemos la velocidad y la distancia. De esta manera deducimos la ecuación 15, hay que notar que en este caso la velocidad usada está en Km/h por lo cual debe convertirse a m/s primero. Esto se logra diviendo la velocidad entre 3,6.

$$TICK\_EN = \frac{200 \cdot 18 \cdot 46875}{velocidad \cdot 5 \cdot 1024} = \frac{32958{,}9843}{velocidad} \tag{15}$$

De manera similar deducimos la ecuación para TICK_DIS en 16.

$$TICK\_DIS = \frac{300 \cdot 18 \cdot 46875}{velocidad \cdot 5 \cdot 1024} = \frac{49438{,}4765}{velocidad} \tag{16}$$

Para mantener el mensaje de alerta por 3 s, necesitamos saber a cuántos TICKS de TCNT equivale esto, para esto usamod al ecuación 17.

$$\frac{3}{T_{TOI}} = \frac{3 \cdot 46875}{1024} = 137{,}3291 \tag{17}$$

Así que basta con esperar a que TICKS_MED aumente en 137.

3. Programa principal

Nuestro programa principal realiza tareas importantes y quita peso de las subrutinas de modo de manera que éstas sean más sencillas. En la figura 1 se pbserva la configuración de hardware e inicialización de variables. Tenemos dos nodos con saltos, el primero es para esperar al menos 160 mS para que encienda el ATD y el segundo es para recorrer el arreglo de inicialización de la pantalla LCD. Luego de esto comienza el programa principal a ejecutarse.

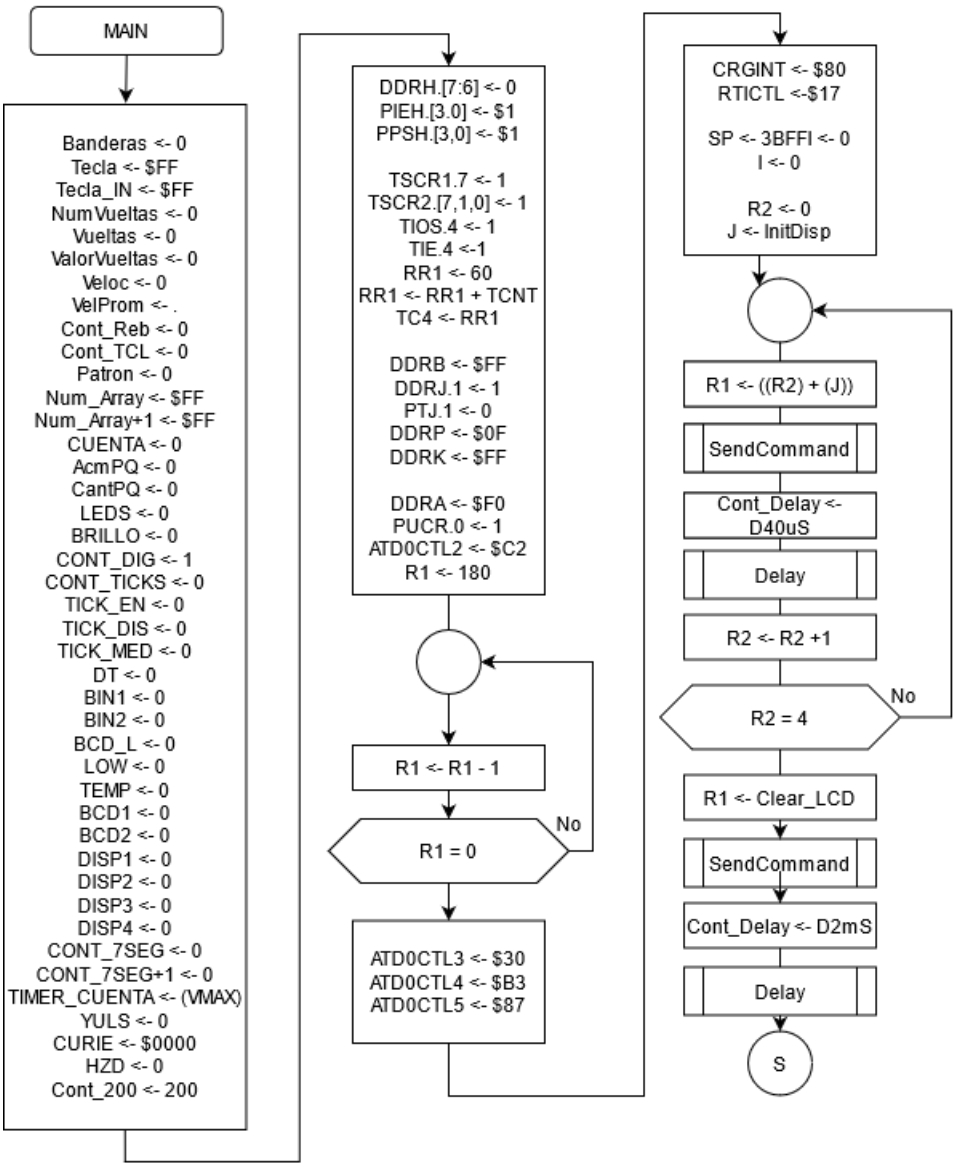

Figura 1: Diagrama de flujo del programa principal parte 1.

Primero el pgrograma entra de manera forzada al modo configuración y no sale hasta obtener un valor válido de vueltas. Luego de esto entonces entra en un ciclo donde revisa el modo seleccionado, revisa si ha cambiado con el modo que se tenía

seleccionado el ciclo anterior y de ser así entonces actualiza la pantalla, variables y las interrupciones acorde al modo y luego entra en ese modo. Esto se observa en la figura 2. En esta porción del diagrama se revisa constantemente la posición de los DIP swithes 7 y 6 que corresponden con los bits 7 y 6 del puerto H. Estos bits fueron declarados como entrada. Por este motivo podemos leer si valor por pooling en el registro PTIH. Luego lo comparamos con el valor en el registro de banderas para ver si ha cambiado y de ser así se actualizan y se pone el bit 4 de Banderas para indicar el cambio de modo poniendo un 1 en ese bit.

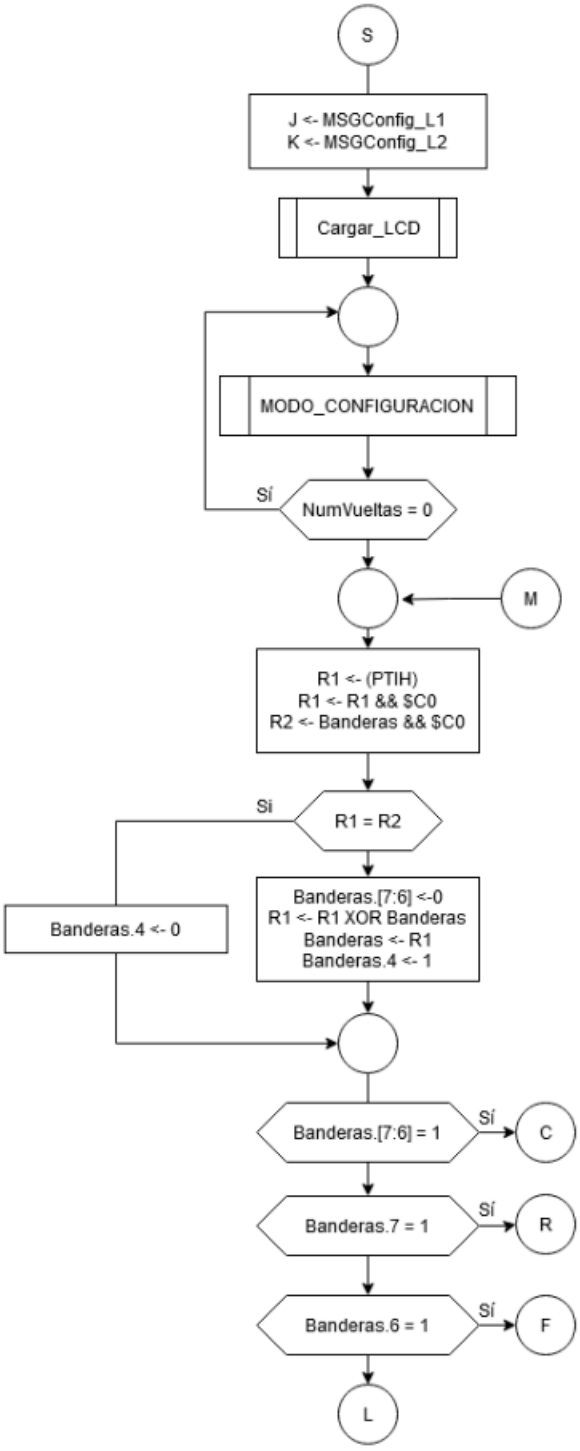

Figura 2: Diagrama de flujo del programa principal parte 2.

En la figura 3 se observa lo que se ejecuta cuando se entra al modo seleccionado en los switches. ANtes de el al modo se revisa si el bit 4 de Banderas está puesto, de ser así entonces significa que el modo cambió recientemente por lo tanto se debe refrescar la pantalla y modificar algunas posiciones de memoria. En el caso de los modos Libre y Resumen: apagamos las interrupciones de PTH, TCNT y RTI. Para el modo configuración apagamos las interrupciones de PTH y TCNT y se enciende la RTI. Para el modo competencia se encienden todas. En todos los modos excepto competencia borramos las variables para cálculos de velocidad y tiempos. Ademas en cada modo se pone una led diferente.

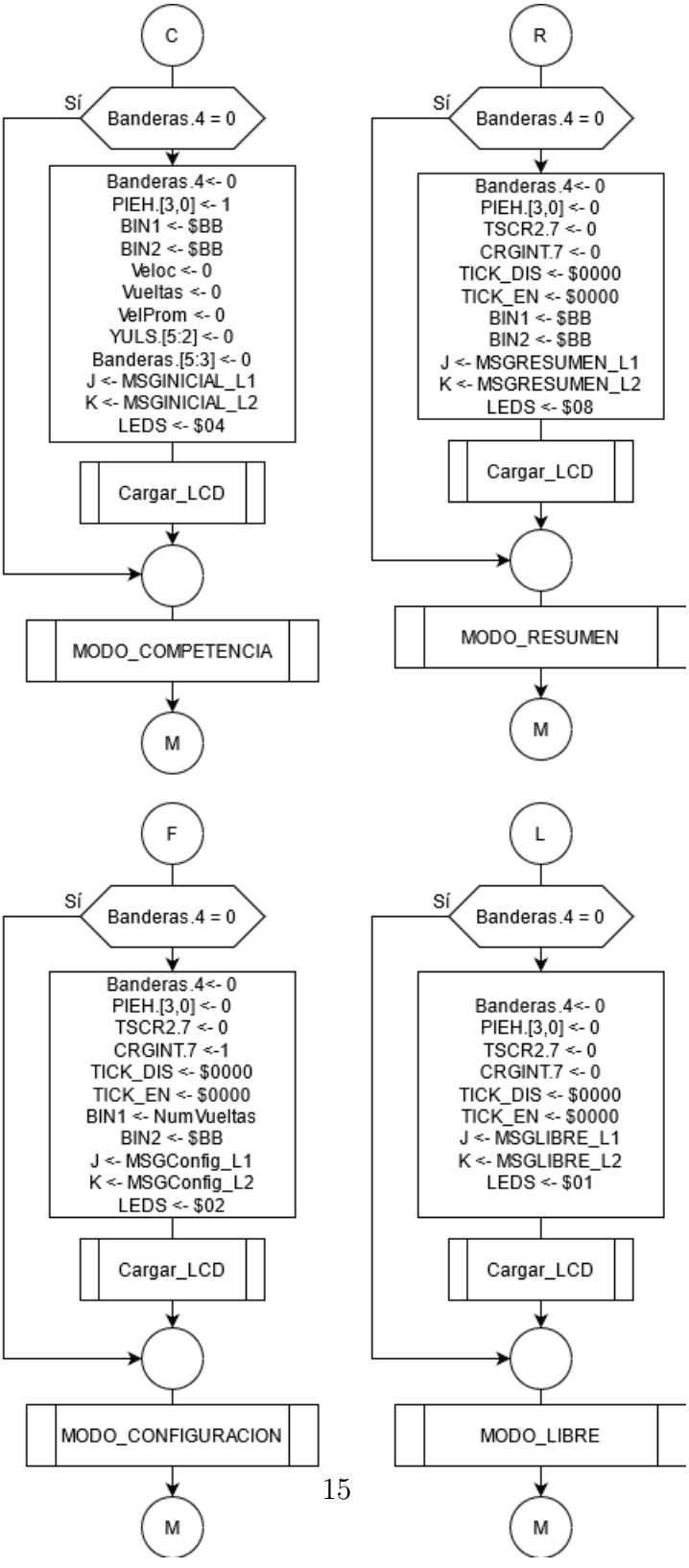

Figura 3: Diagrama de flujo del programa principal parte 3.

3.1. Subrutinas

3.1.1. Subrutinas para los diferentes modos de operación

3.1.2. Subrutina MODO_CONFIG

Esta subrutina, cuyo diagrama se puede ver en la Figura 4, se encarga de llamar a la subrutina Tarea_Teclado para ingregar un valor a través del teclado matricial, al finalizar esta acción, se pone en 1 el bit 2 de banderas, el cuál corresponde a Array_Ok, luego llama a la subrutina BCD_BIN, la cuál se encarga de tomar el número que fue almacenado en Num_Array y pasarlo de BCD a Binario y guardarlo en la variable ValorVueltas, luego de esto pone en 0 la bandera de Array_Ok para poder repetir el ciclo, seguido de esto, se verifica que el valor se encuentra dentro del rango válido (3-23), si está en el rango, entonces se guarda el valor en la variable NumVueltas, y luego en BIN1, luego de esto, se borra Num_Array y se retorna. En el caso de que el valor no sea válido, se pone 0 en Num_Vueltas y se retorna.

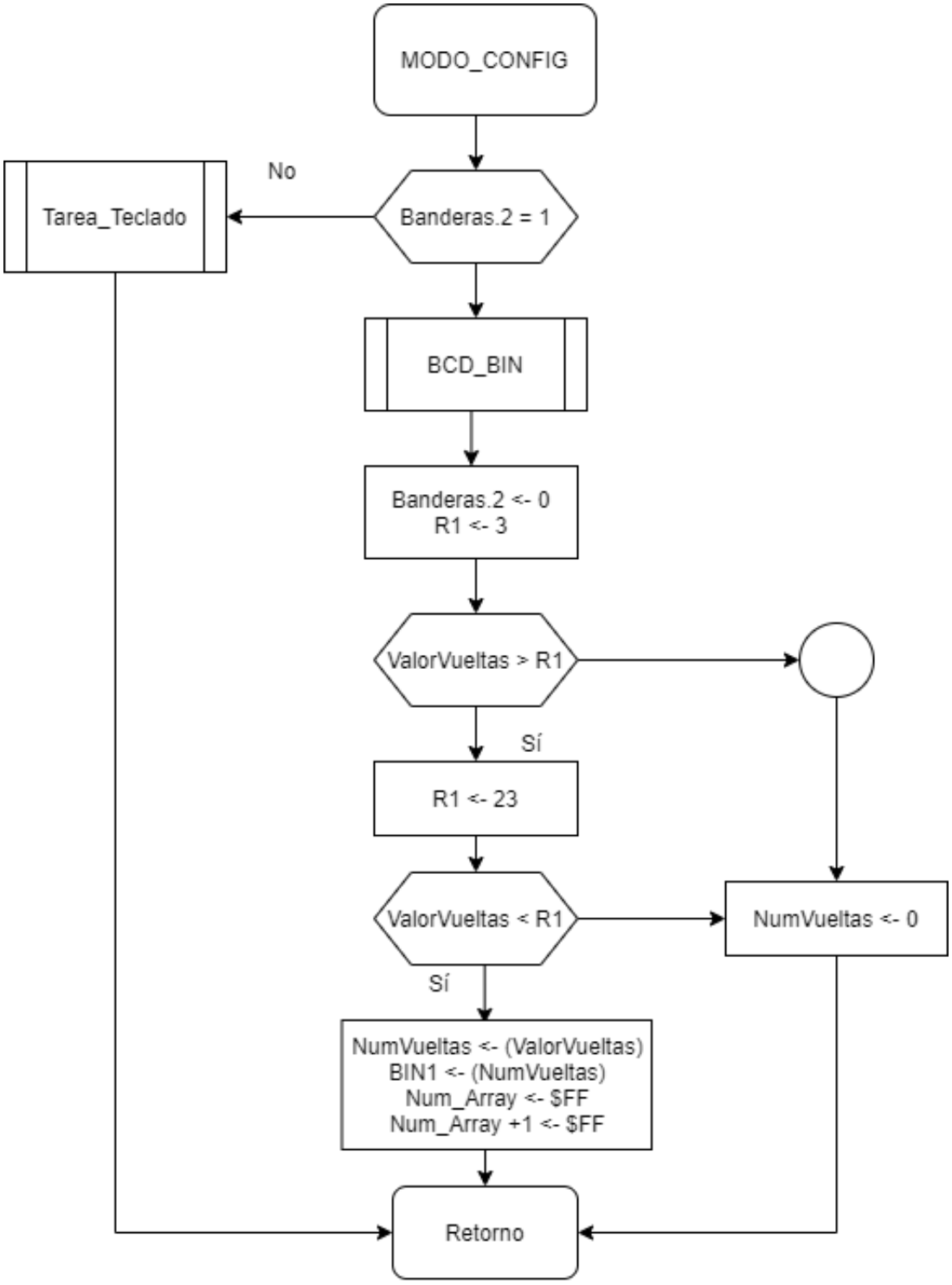

Figura 4: Diagrama de flujo de Modo$_CONFIG$.

3.1.3. Subrutina MODO_COMPETENCIA

Esta subrutina, cuyo diagrama se muestra en la Figura 5, se encarga de desplegar en la pantalla el mensaje de calculando y de llamar a la subrutina PANT_CRTL. Primero, revisa el estado de la bandera Habrá cálculo, esta se pone en 1 en la subrutina CALCULAR cada vez que se presiona PH3 por primera vez, entonces cada vez que la bandera se ponga en 1, esta subrutina enviará el mensaje de calculando a la pantalla LCD y además pondrá $BB en BIN1 y BIN2 para apagar los displays de la pantalla de 7 segmentos, luego de esto, si la velocidad es diferente de 0, llama a la subrutina PANT_CTRL y retorna.

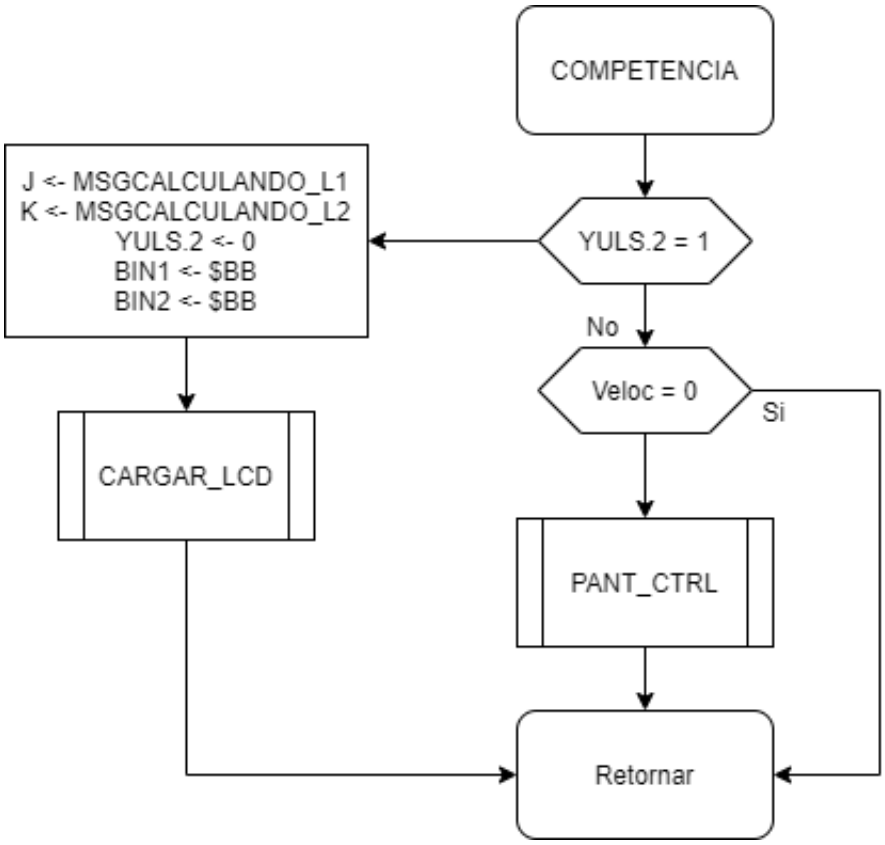

Figura 5: Diagrama de flujo de COMPETENCIA

3.1.4. Subrutina MODO_RESUMEN

El diagrama del modo resumen se observa en la figura 6. El modo resumen únicamente coloca en los leds BIN1 y BIN2 los valos de VelProm y Vueltas respectivamente

para mostrarse en la pantalla de 7 segmentos cada vez que Oc4 los actualiza y retorna.

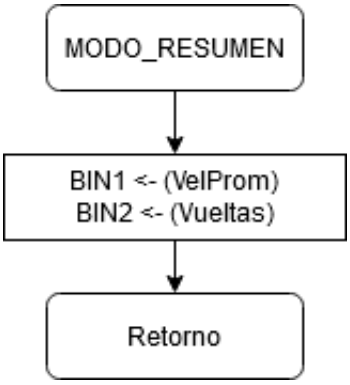

Figura 6: Diagrama de flujo del MODO_RESUMEN parte 3.

3.1.5. Subrutina MODO_LIBRE

Esta subrutina, cuyo diagrama se puede ver en la Figura 7, se encarga de poner $AA en BIN1 y BIN2, para apagar los displays de la pantalla de 7 segmentos.

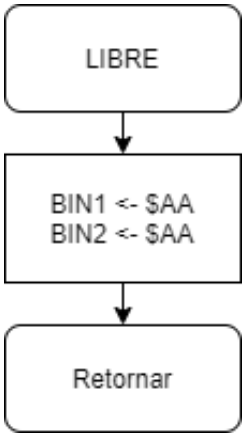

Figura 7: Diagrama de flujo del MODO_LIBRE.

3.1.6. Subrutinas de interrupción

3.1.7. Subrutina Timer Overflow Interrupt

Esta interrupción es la encargada de contar tiempos para poder obtener la velocidad del ciclista y para controlar el despleque de mensajes en la pantalla dentro de

la subrutina Pant_CTRL. Lo primero que debemos hacer es borrar la bandera deinterrupción poniendo un 1 en el bit 7 de TFLG2. Luego incrementamos TICK_MED para esto lo cargamos en el registro índice X ya que es una variable tipo word. Luego decrementamos TICK_EN y TICK_DIS si no valen 0. Cuando la primera llega a 0 pones en 1 el bit 3 de Banderas llamado PantFLG y cuando la segunda llega a cero lo borramos. Esto se usa dentro de Pant_CTRL para indicar cúando se debe poner o quitar un mensaje en las pantallas respectivamente. Su diagrama se observa en la figura 8.

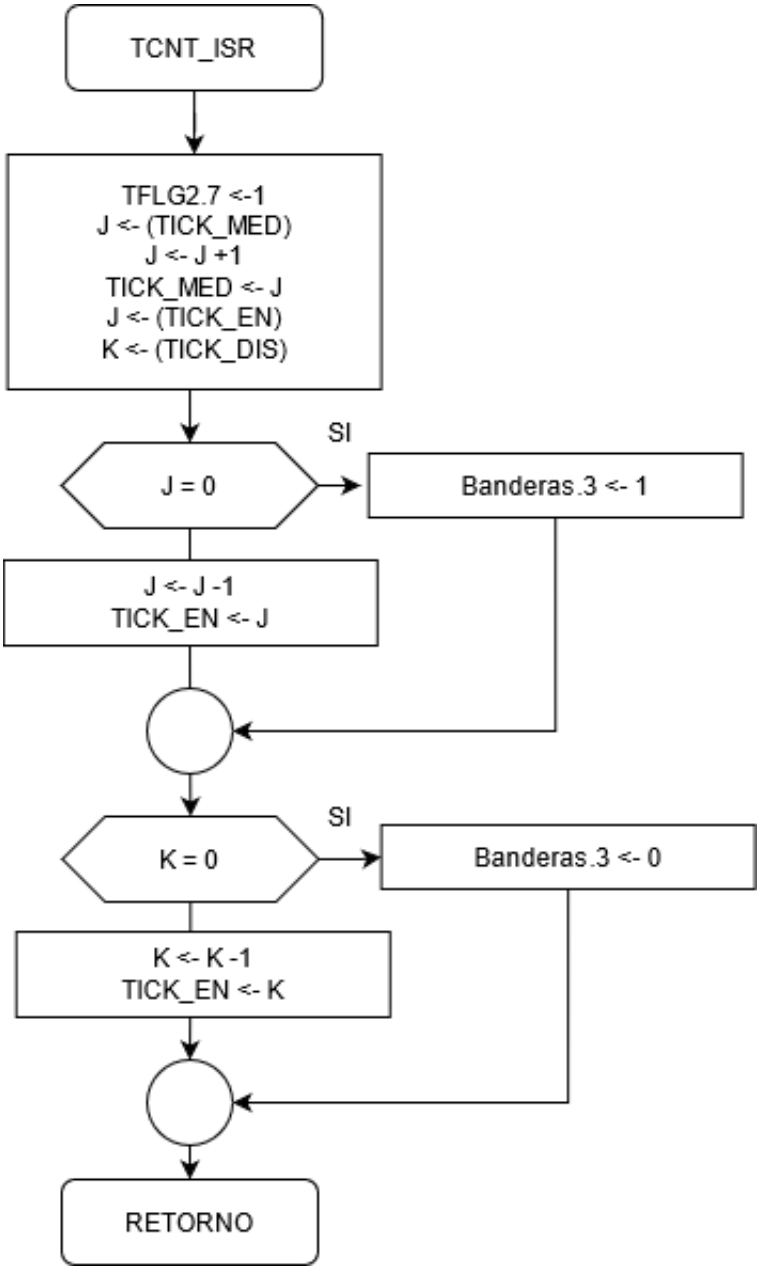

Figura 8: Diagrama de flujo de TNCT ISR.

3.1.8. Subrutina Timer Output Compare canal 4

Esta subrutina de interrupción, cuyos diagramas corresponden a las Figuras 9 y 10, se encarga de descontar tres valores: Cont_Delay, para el conteo de tiempo en Delay, Cont_Ticks, para la configuración del brillo y Cont_7seg para la manipulación de la pantalla de 7 segmentos. Además, se encarga de desplegar en los displays de la pantalla de 7 segmentos los valores correspondientes.

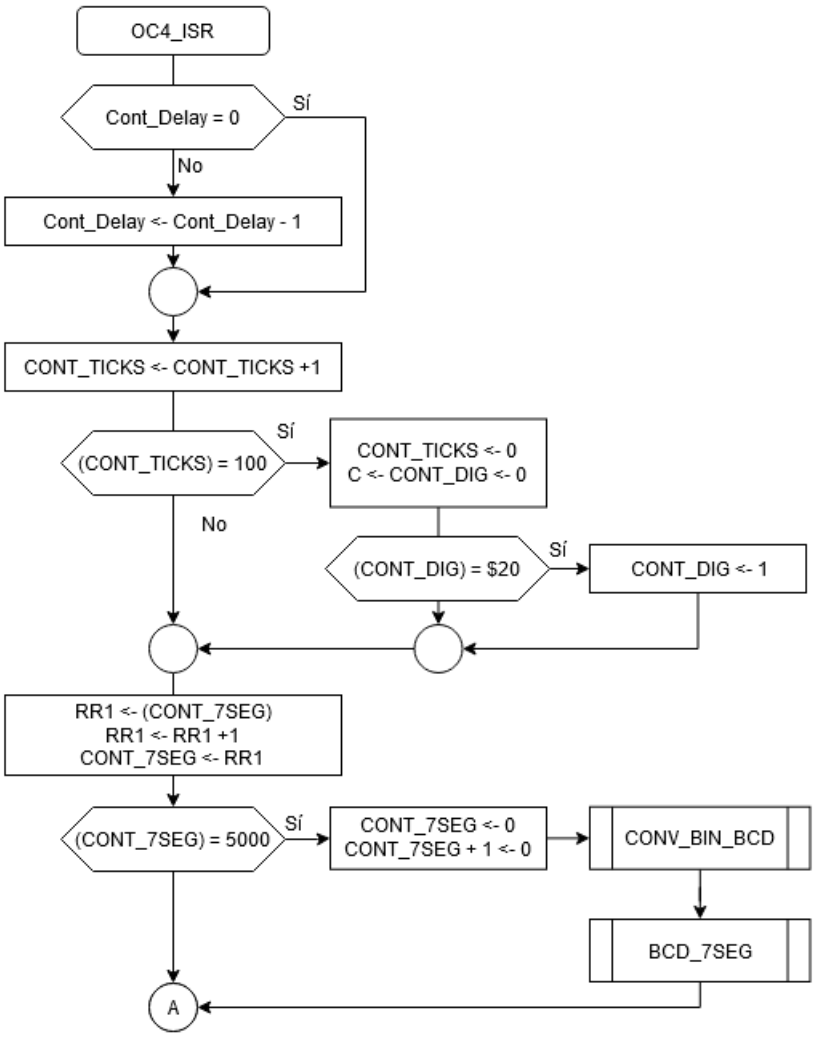

Figura 9: Diagrama de flujo de OC4, parte 1.

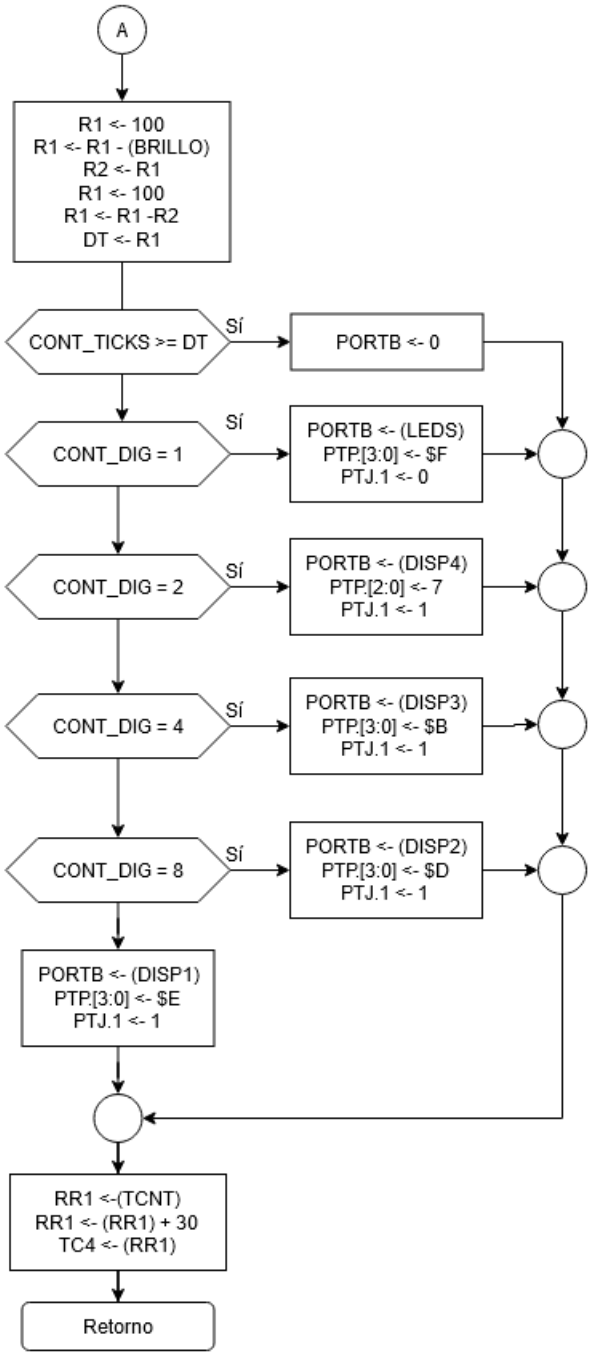

Figura 10: Diagrama de flujo de OC4, parte 2.

3.1.9. Subrutina Real Time Interrupt

Esta subrutina, cuyo diagrama se muestra en la Figura 11, se encarga de descontar la variable Cont_Reb cada 1 ms y espera cada 200 ms para llamar a ATD07, incrementando la variable HZD y luego comparando con CONT_200.

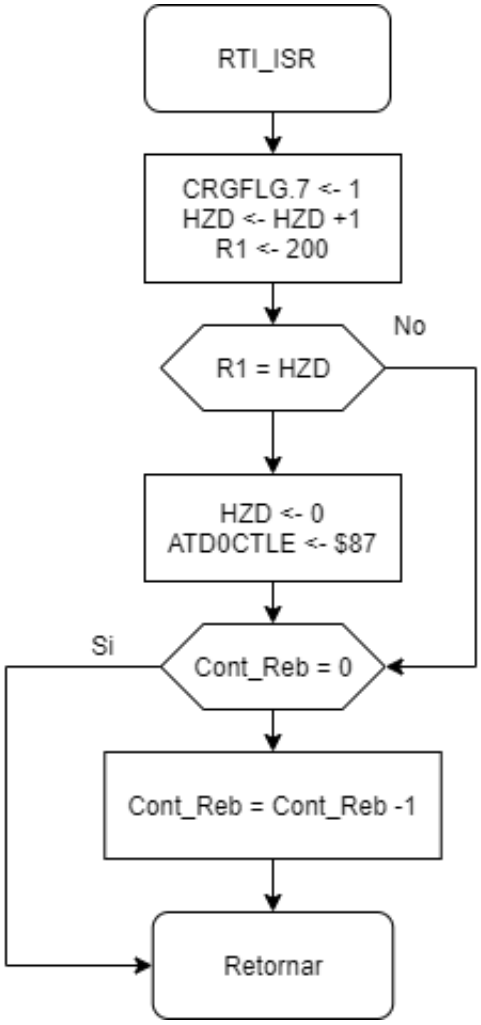

Figura 11: Diagrama de flujo de RTI.

3.1.10. Subrutina Key Wake Ups Puerto H (CALCULAR)

Esta subrutina, cuyo diagrama se muestra en la Figura 12, se encarga de atender las interrupciones provocadas al presionar los botones PH3 y PH0, los cuáles corres-

ponden a los sensores S1 y S2 correspondientemente. Primeramente se implementa una forma de lidiar con los rebotes, utilizando la variable Cont_Reb. Seguido de esto se pregunta si se presionó PH3, en caso de que sí, se pregunta también por el bit 4 de la variable YULS, el cuál es una bandera que indica si es la primera vez que se presiona, en caso de que no sea la primera vez, retorna, ya que el orden de activación de los sensores siempre deberá ser PH3 y luego PH0. Si es la primera vez, entonces pone la variable TICK_MED en 0, como referencia al inicio del tiempo que el ciclista va a durar en llegar al segundo sensor. Aumenta en 1 la variable VUELTAS y además pone la bandera de dirección, es decir, el bit 4 de la variable YULS en 1, para indicar que ya fue accionado este botón, pone la bandera de Habrá cálculo y retorna. Si no se ha presionado PH3, seguirá a preguntarse si se ha presionado PH0, si este no es el caso, retorna, si se presionó PH0, entonces se pregunta por la bandera de dirección, si está en 1 quiere decir que ya PH3 fue presionado y puede continuar, en caso contrario, debe retornar. Si prosigue, entonces borra la bandera de dirección, para permitir que se repita el ciclo, también la bandera de Habrá cálculo, bit 2 de YULS. Luego de esto, se calcula la velocidad, donde TICK_MED representa el tiempo transcurrido entre que se presionó PH3 y PH0, ya que se puso en 0 al presionar PH3 como se mencionó antes y la subrutina TCNT_ISR se encarga de aumentar esta variable. Luego de calcular la velocidad, se comprueba que esta se encuentra en el rango solicitado (35 a 95 Km/h), en caso de que no, pone la bandera de velocidad válida, bit 5 de la variable YULS en 0, descuenta la variable VUELTAS y retorna. Después de esto, si la velocidad se encuentra en el rango, pone en 1 más bien la bandera de velocidad válida y procede a calcular la velocidad promedio nueva. Por último se deshabilitan las interrupciones de los botones y se retorna. Se tomó la decisión de implementar la verificación de la velocidad en esta subrutina, para no incluir en el cálculo de velocidad promedio, datos de velocidad no permitida.

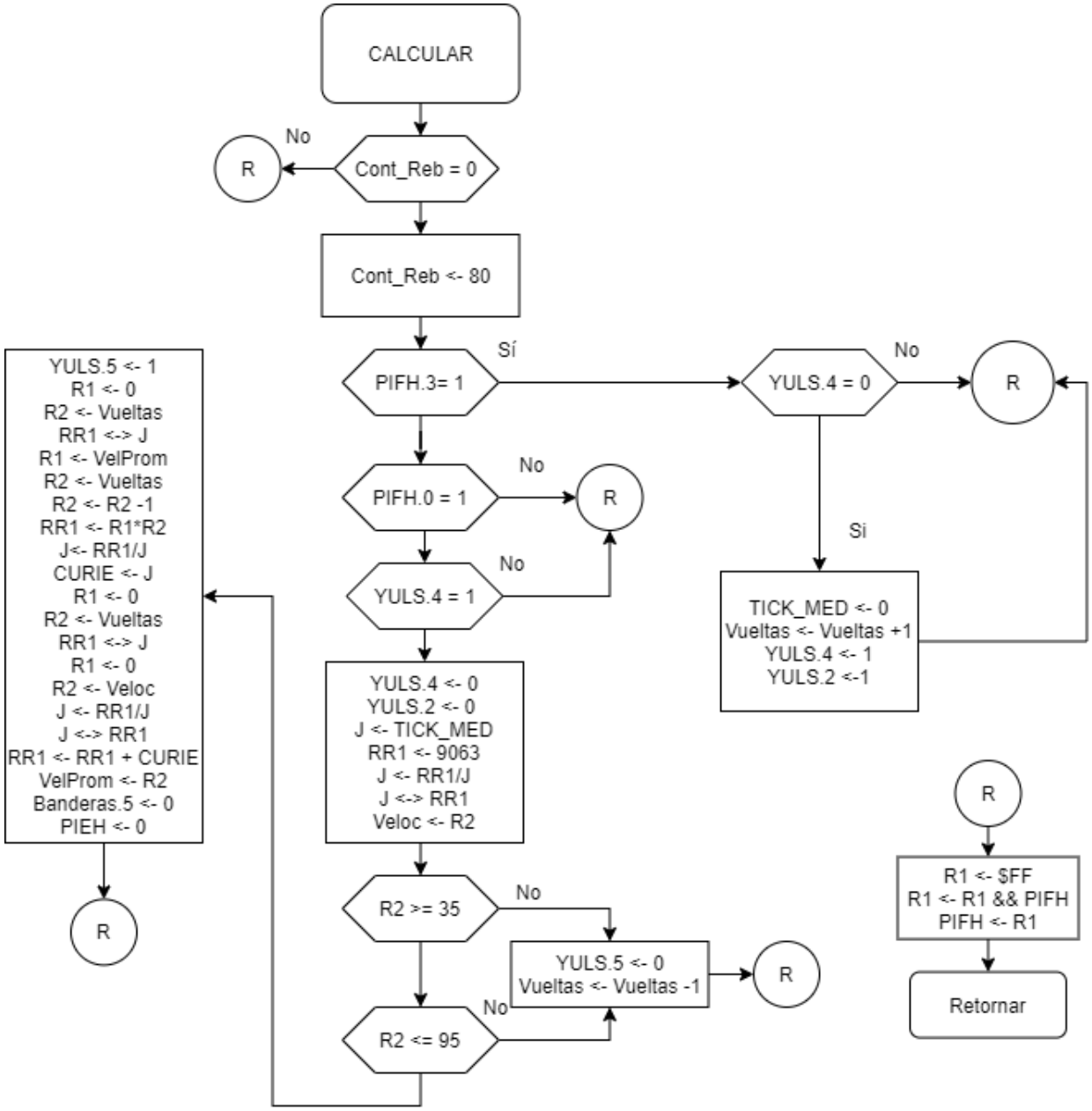

Figura 12: Diagrama de flujo de CALCULAR.

3.1.11. Subrutina Analog to Digital converter 0

Esta subrutina se encarga de leer el canal 7 del convertidor analógico a digital 0, este pad está conectado al trimpot de la parte superior derecha de la tarjeta Dragon. Ya que se activó la bandera AFFC no es necesario borrar de manera manual la bandera de interrupción, ya que se borra al leer los registros donde se guardan los resultados de las conversiones y eso es lo primero que se hace en esta subrutina. Luego calculamos un promedio de las 6 lecturas, las lecturas son de 8 bits, sin embargo deben cargarse en RR1 ya que 6 veces 8 bits podría no caber en 1 byte. Luego convertimos mediante regla de 3 el valor dee entre un rango de 0 a 255 a uno de 0 a 20 y por último lo multiplicamos por 5 para que ahora sea de 0 a 100. Luego comparamos este valor para ver si está en un rango válido [5,95] y de ser así se guarda en la variable BRILLO, de no ser así se mantiene en ese rango e igualmente se guarda en la variable BRILLO. El diagrama se obsera en la figura 13.

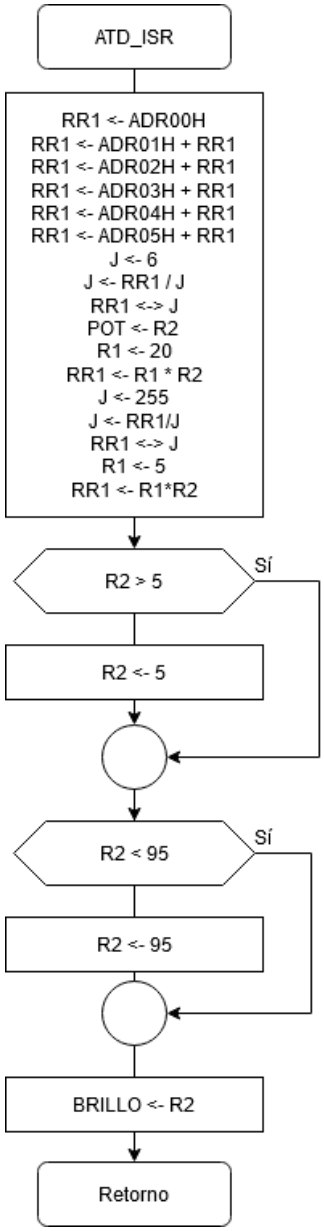

Figura 13: Diagrama de flujo de ATD= ISR.

3.1.12. Subrutinas para manejar las pantallas LCD y de 7 segmentos

3.1.13. Subrutina Pant_CTRL

Esta subrutina, cuyo diagrama se puede ver en la Figura 14, se encarga de calcular los valores de las variables TICK_END y TICK_DIS, utilizadas para calcular el tiempo en que se deben poner los mensajes correspondientes a la velocidad y número de vueltas y cuando el mensaje inicial de nuevo, además, el caso del mensaje de alerta para cuando la velocidad no se encuentra en el rango establecido. El cálculo de TICK_END y TICK_DIS se debe hacer solo la primera vez que se ingresa a la subrutina, para esto se usa la bandera CALC_TICKS, la cuál se pone en 1 después de realizar el cálculo. Entonces primero se pregunta por esta bandera, si es 0, es la primera vez que entra, y debe hacer el cálculo, antes de hacer el cálculo, se pregunta por el bit 5 de la variable YULS, que corresponde a velocidad válida, si la velocidad no es válida asigna siempre los mismos valores a TICK_END y TICK_DIS que harán que el mensaje correspondiente para este caso (el de Alerta) se ponga inmediatamente en la pantalla por 3 s y luego vuelva al mensaje inicial. En caso contrario, calcula los valores correspondientes según la velocidad promedio del ciclista. Después de asignar los valores correspondientes a las variables de TICKS, pone en 1 la bandera de CALC_TICKS para indicar que ya fueron calculados. La segunda vez que entra, esta bandera estará en 1, impidiendo que repita los cálculos, y proseguirá a transferir el valor del bit 3 de banderas, que corresponde a la bandera PANT_FLAG al bit 3 de la variable YULS, si son iguales, quiere decir que no hay cambio y no se deben actualizar las pantallas, si no lo son, entonces si el valor del bit 3 de YULS (PANT_FLAG) es 0, manda el mensaje inicial a la pantalla LCD, carga $BB en BIN1 y BIN2 para apagar los displays de la pantalla de 7 segmentos y pone el valor de velocidad en 0. Por último verifica si el valor de Vueltas ya alcanzó el valor programado de NumVueltas, si es el caso, deshabilita los puertos, pone en 0 la bandera de CALC_TICKS y retorna. Si en cambio el valor de PANT_FLAG es 1, indica que se debe poner el mensaje de competencia en la pantalla LCD y los valores de Velocidad y Vueltas en BIN1 y BIN2 para que se desplieguen en la pantalla de 7 segmentos, sin embargo, antes de esto, se pregunta por el valor del bit 5 de la variable YULS, el cuál corresponde a velocidad válida, si la velocidad no es válida, pondrá en cambio en mensaje de Alerta en la pantalla LCD y $AA en BIN1 y BIN2, para que aparezcan guiones en la pantalla de 7 segmentos, tal como fue solicitado.

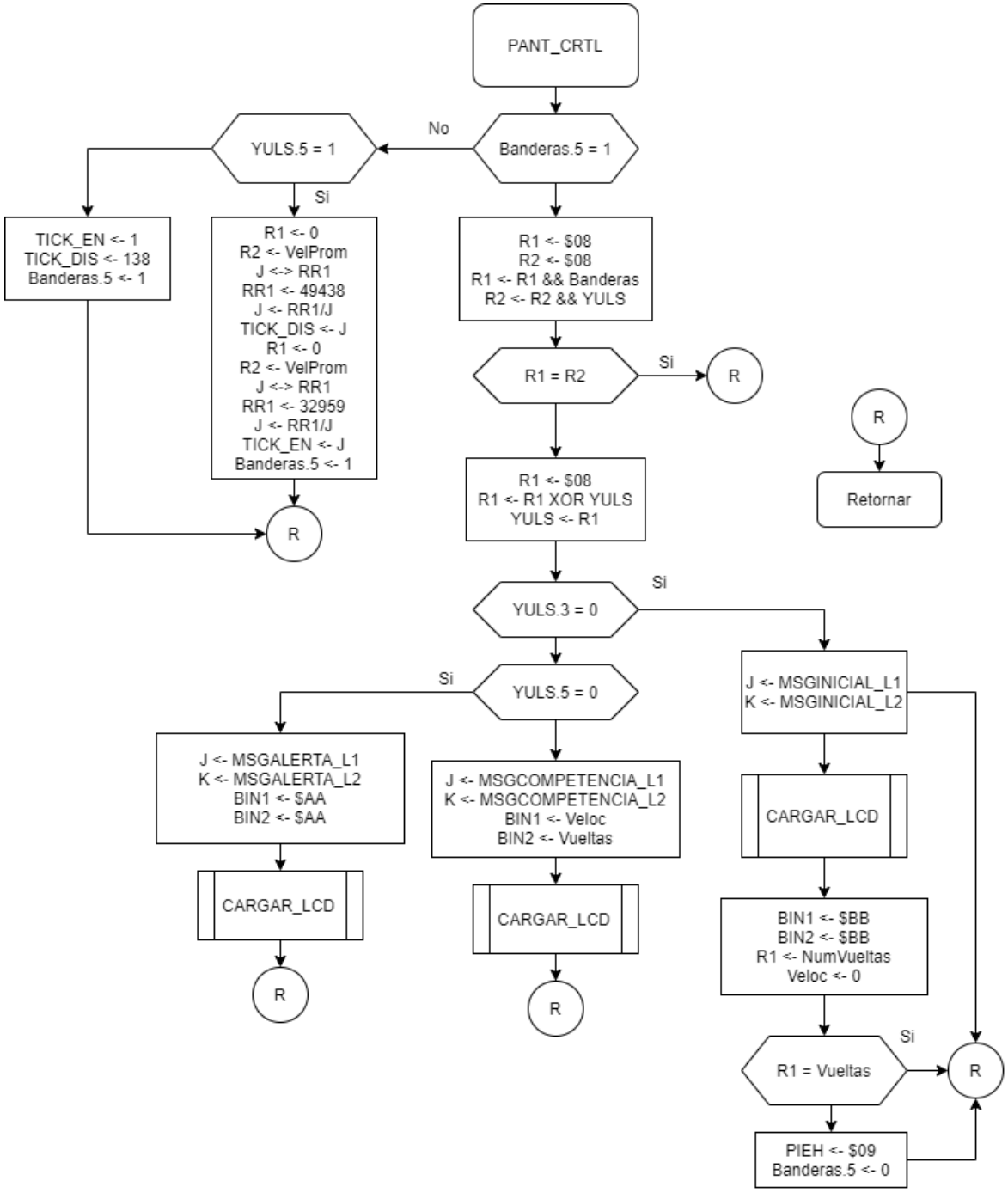

Figura 14: Diagrama de flujo de PANT_CRTL

3.1.14. Subrutina BCD_7SEG

Esta subrutina se encarga de tomar los valores en BCD1 y BCD2 y colocarlos en las variables DISP1, DISP2, DISP3 y DISP4. Primero cargamos en J la dirección de la tabla SEGMENT y luego usamos direccionamiento de offset por acumulador [2] para colocar el valor de la posición SEGMENT + el valor en R1 en cada una de las variables DISP. Los valores de R1 son ambos nibbles de BCD1 y BCD2, ya que cada nibble corresponde a un dígito. En el casso del nibble superior primero debe desplazarse 4 bits a la derecha. Las variables DISP se colocan en el puerto B en OC4 en el diragrama en 9 para mostrarse en el display de 7 segmentos. El diagrama de flujo de esta subrutina se observa en la figura 15.

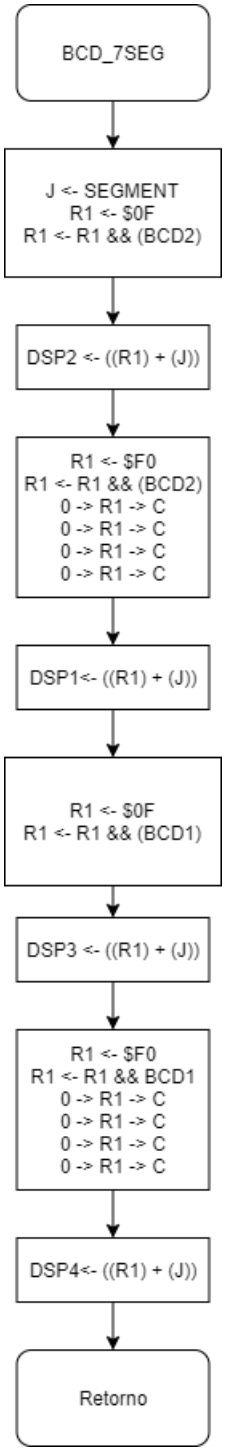

Figura 15: Diagrama de flujo de BCD 7 SEG.

3.1.15. Subrutina Cargar_LCD

Esta subrutina se encarga de cargar los mensajes en la pantalla LCD según las especificaciones de la tarea 5. Para esto se cargan en R1 bytes a enviar en la pantalla y se envian llamando a las subrutinas SendCommand y SendData ya sea el caso de datos o comandos. En el caso de datos se barre una tabla de caracteres constantes hasta que se llega al caracter de final o EOM, cada vez que se envía un byte hay que esperar 40 micro segundos. Luego de enviar el texto de cada línea de la pantalla se retorna. La direción del texto de la primera línea se carga en J y el de la segunda en K. Esto se observa en la figura 16.

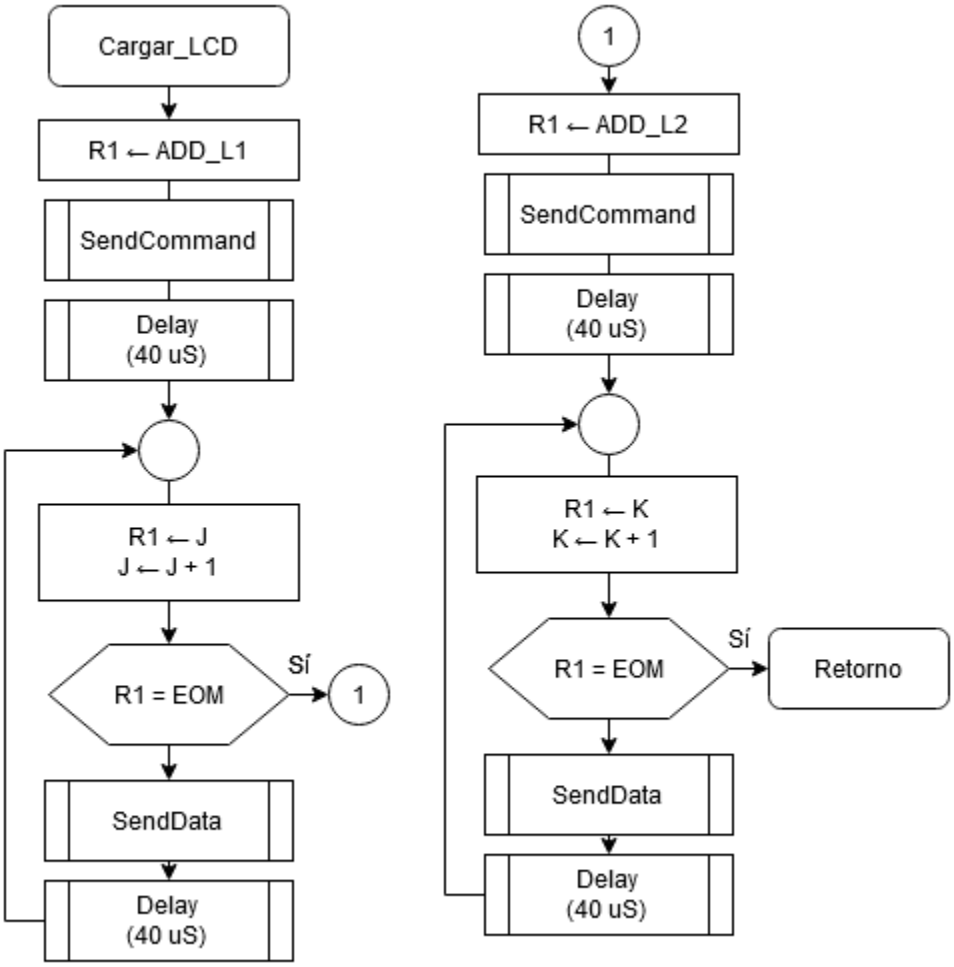

Figura 16: Diagrama de flujo del CARGAR_LCD.

3.1.16. Subrutina Delay

La subrutina Delay, cuyo diagrama está en la figura 17, únicamente revisa si la variable Con_Delay es 0 y de ser así retorna, de no ser así bloquea el código en un salto a ella misma.

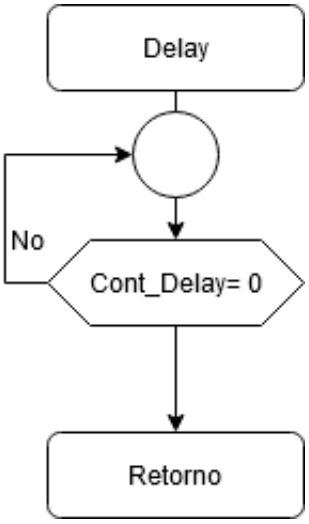

Figura 17: Diagrama de flujo de Delay.

3.1.17. Subrutina Send_Data y Send_Command

Las subrutinas SendCommand y SendData son bastante similares como puede observarse en sus diagramas en la figura 18. De hecho la única diferencia es que en la subrutina SendData se pone el bit 0 del puerto k para indicar al controlador de la pantalla que se trada de un dato a poner en su memoria interna. Ambas subrutians colocan R1 en la pila al entrar para guarar su valor inicial, luego toman el nibble superior y lo corren dos bits a la izquierda para acomodarlo con los bits de datos de la pantalla los cuales van del bit 2 al 5. Luego de poner los datos y el valor adecuado en los bits 0 y 1 del puerto K se espera 260 mS para que se lean los datos. Se desapilar R1 para recuperar su valor original. Luego se corren a esa posición los bits de nibble inferior y se hace lo mismo.

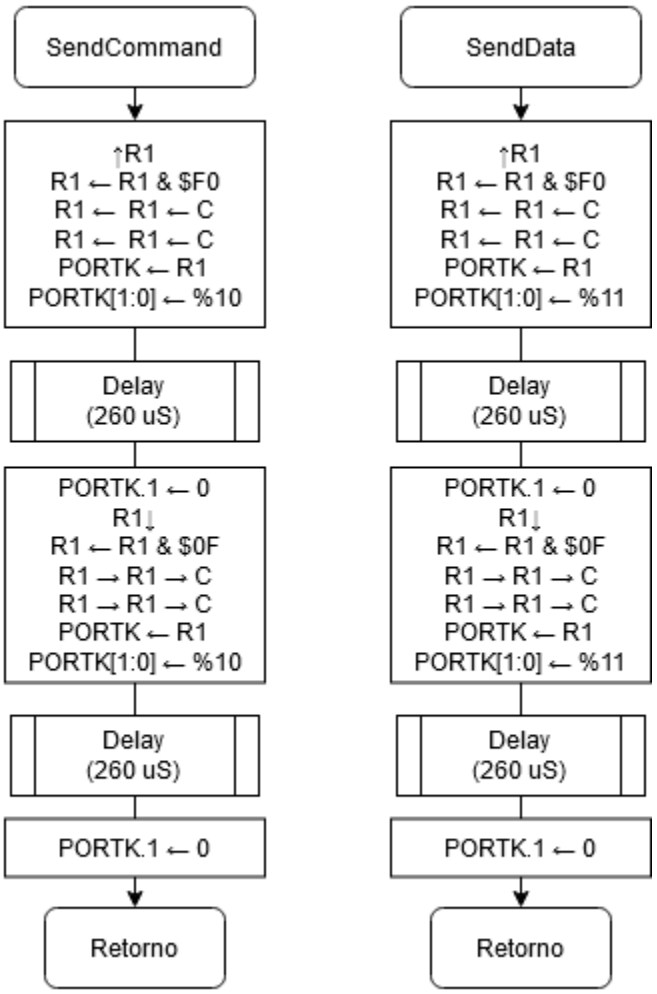

Figura 18: Diagrama de flujo de Send Command y Data.

3.1.18. Subrutinas para manejar el teclado

3.1.19. Subrutina Tarea_Teclado

Esta subrutina, cuyo diagrama se muestra en la Figura 19, se encarga de manejar los rebotes, de forma que hasta que el contador de rebotes sea cero, llama a la subrutina Mux_Teclado, la cuál se encarga se leer la tecla que se presiona, en casox de que presione una tecla, o indicar que no está presionando una tecla (al colocar el

valor de $FF), debido a esto, después de la llamada a Mux_Teclado, se pregunta si Tecla es igual o no a $FF,en caso de que lo sea, puede ser porque el usuario ya dejó de presionar la tecla, entonces verifica con la bandera Tecla Lista, y si está Lista procede a reiniciar las banderas y llamar a la Subrutina Formar_Array, para guardar el valor digitado correspondiente. Si el valor no es igual a $FF, quiere decir que una tecla fue presionada, luego de esto dependiendo de la bandera Tecla Leída puede discrimar si se trata de un error, de rebote u otro, o si efectivamente es un dato válido.

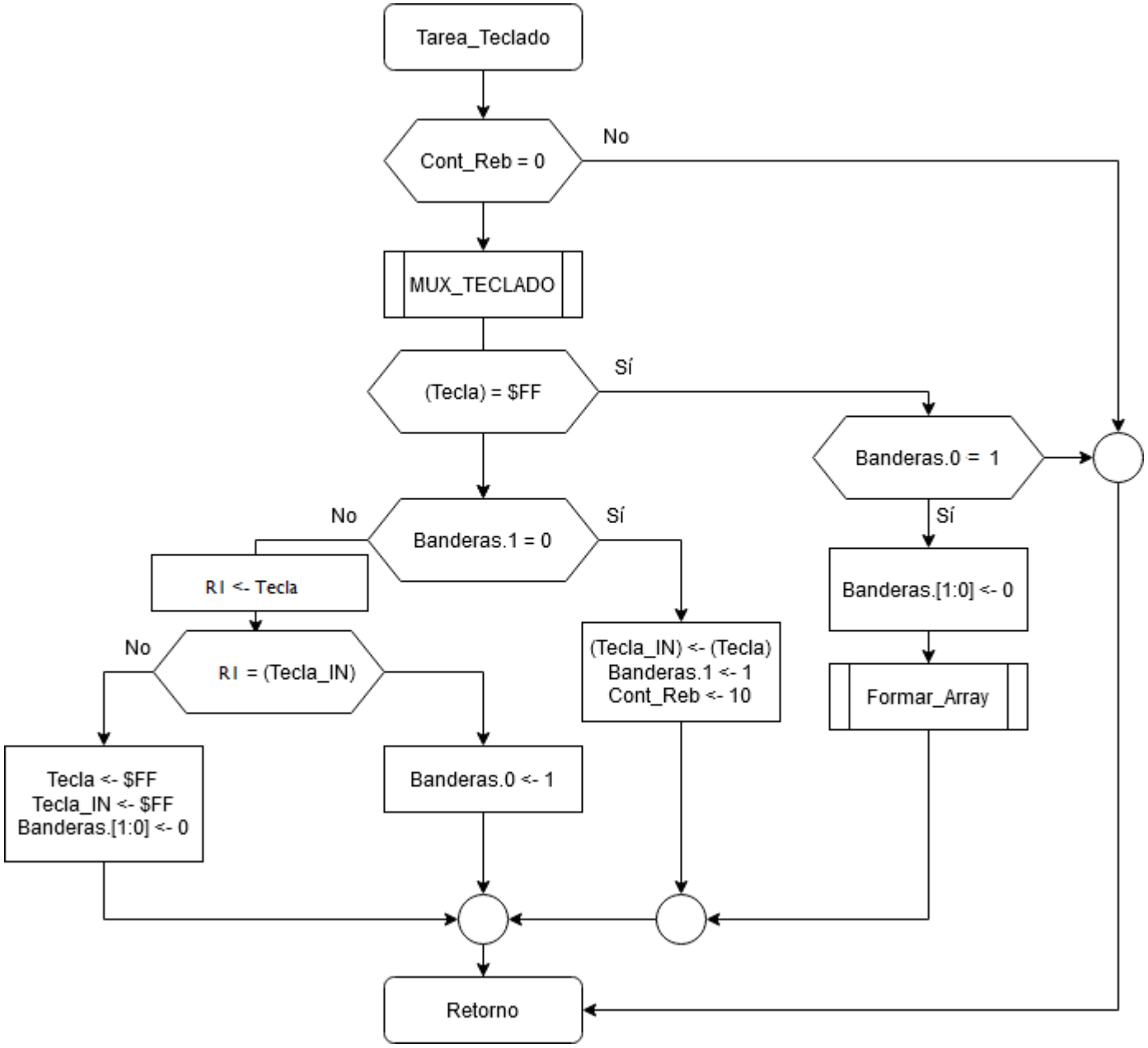

Figura 19: Diagrama de flujo de Tarea Teclado.

3.1.20. Subrutina Mux_Teclado

Esta subrutina, cuyo diagrama se muestra en la Figura 20, se encarga de leer la tecla del teclado matricial que fue presionado, asigna un patrón específico al puerto A y al rotarlo, puede verificar cuál tecla de la columna y fila correspondiente fue presionada, según el número que se encuentra en el contador R1, según ese número se codificara el valor correspondiente, tal como se definió en el array de los valores de Teclas.

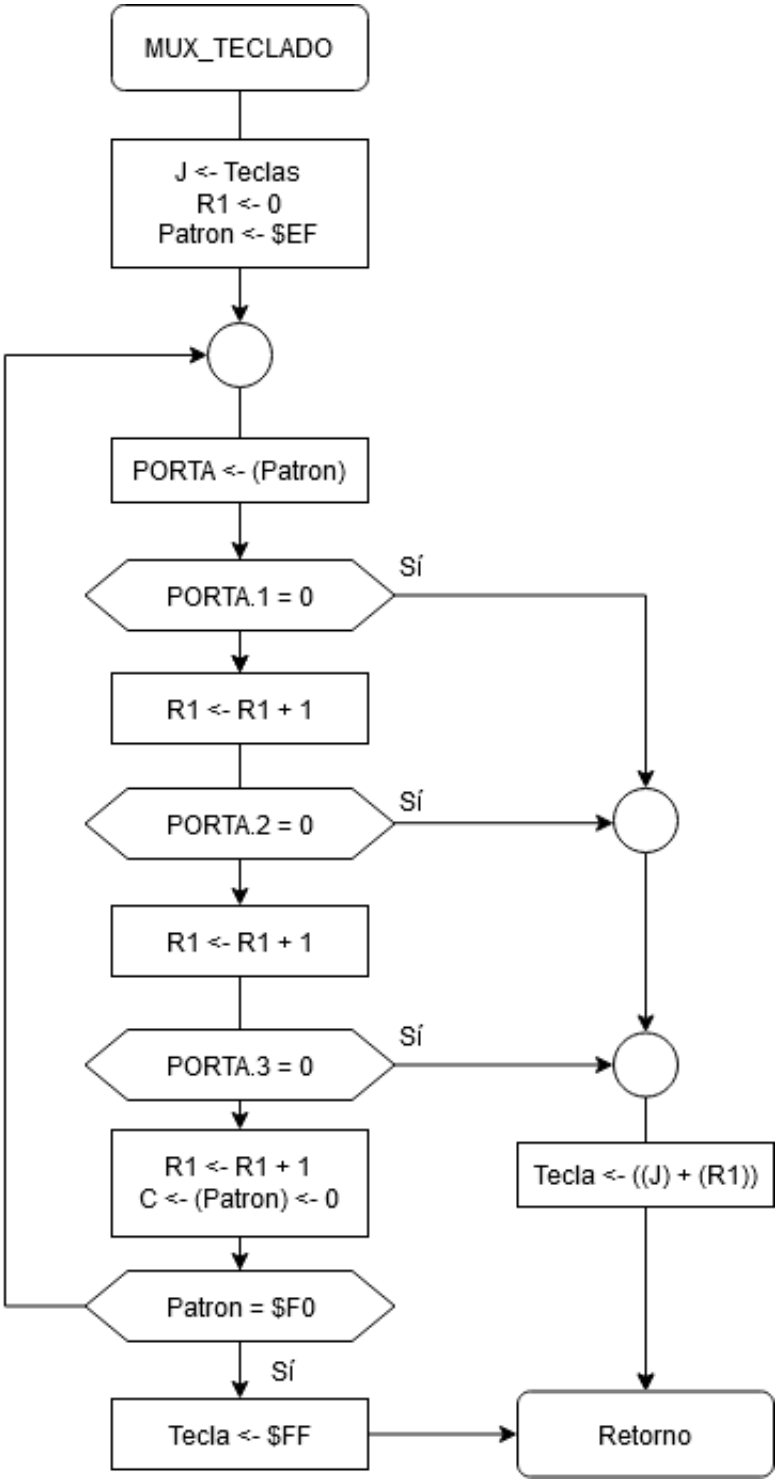

Figura 20: Diagrama de flujo de Mux_Teclado.

3.1.21. Subrutina Formar_Array

Esta subrutina, cuyo diagrama se muestra en la Figura 21, se encarga de formar el array correspondiente, primero si no ha llegado al máximo permitido (2 teclas), se pregunta si es la primera vez que se digita una tecla, si e la primera vez y se presiona Enter o Borrar, las ignora, sino, guarda el dato en el array. Si no es la primera vez y presiona Borrar, procede a reduir el contador de teclas y además pone $FF en esa última posición para borrar el dato del Num_Array, si es Enter, actualiza la Bandera Array_Ok y reinicia el contador de teclas, si no es Borrar ni Enter guarda el valor correspindiente en el arreglo y aumenta el contador de teclas. Si ya se llegó al máximo, se pregunta una vez más, si es Borrar o Enter y procede a hacer lo mismo que se explicó anteriormente. Antes de retornar se pone el valor $FF en Tecla_IN.

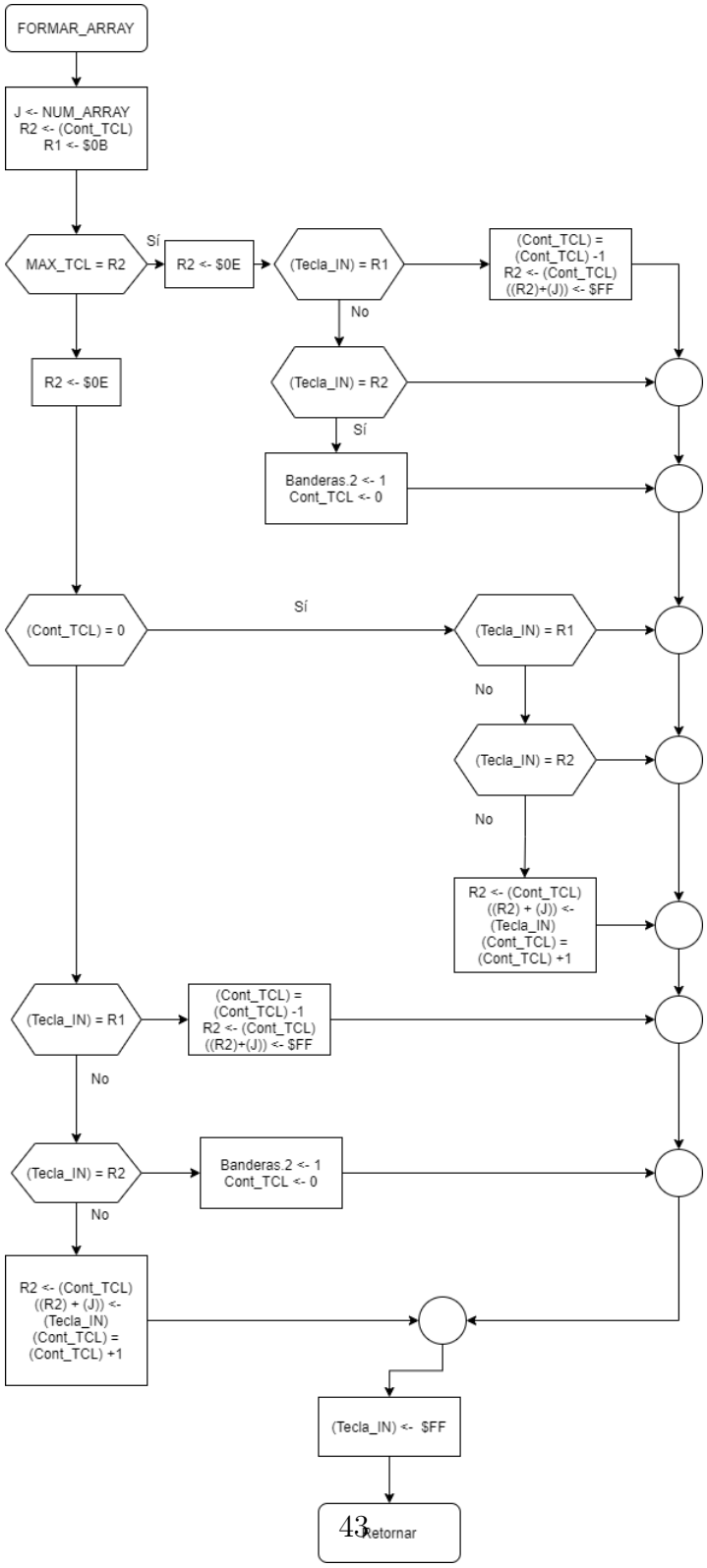

Figura 21: Diagrama de flujo de Formar_Array.

3.1.22. Subrutinas para convertir binario y BCD

3.1.23. Subrutina BIN_BCD

Esta subrutina, cuyo diagrama se muestra en la Figura 22, se encarga de convertir un número de Binario a BCD, utilizando el algoritmo XS3, devuelve el resultado en la variable BCD_L.

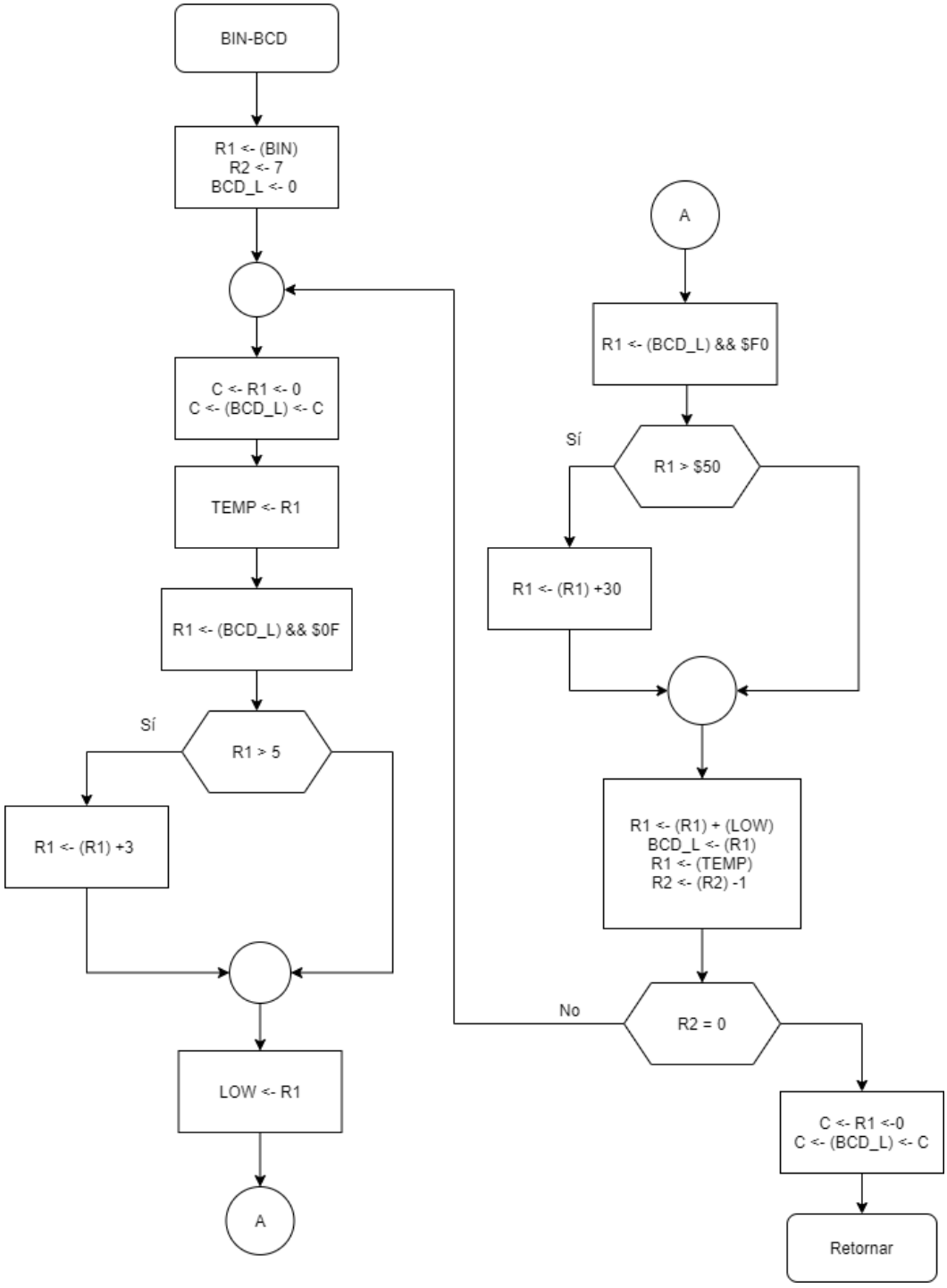

Figura 22: Diagrama de flujo de BIN_BCD.

3.1.24. Subrutina BCD_BIN

Esta subrutina se encarga de convertir el valor en BCD en el arreglo Num_Array a binario y lo guarda en la variable ValorVueltas. En el caso de que se ingrese un sólo dígito este es aceptado sin la necesidad de presinar el 0 antes. Si la posición 2 del arreglo no es $FF, o sea se ingresaron dos dígitos, entonces se toma el dígito de las decenas (el primero) se multiplica por 10 y se guarda en R1, luego se le suma el dígito de las unidades y se guarda en ValorVueltas el resultado. Lo anterior no se encuentra en el enunciado del proyecto [3], sin embago es una optimización elegante que permite el uso natural de parte del usuario. De esta manera queda convertido a binario.

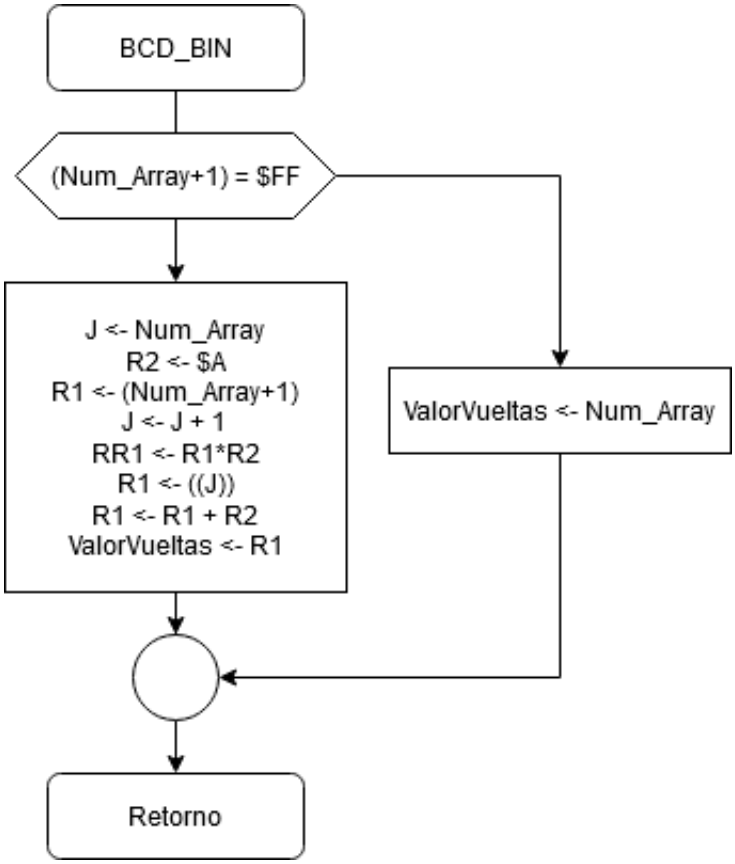

Figura 23: Diagrama de flujo de BCD_BIN.

3.1.25. Subrutina CONV_BIN_BCD

Esta subrutina mostrada en la figura 24 se encarga de tomar los valores en las variables BIN1 y BIN2 convertir sus valores a BCD colocándolos en las variables BCD1 y BCD2. En el caso de que BIN2 sea $BB. Se pone la bandera ApagarBCD2 o el bit 7 de YULS. De esta manera al final de la subrutina se coloca $BB en BCD2 ya que la subrutina BCD_7SEG colocará el valor necesario para apagar los dígitos de la izquierda en la pantalla de 7 segmentos. Por otro lado si se encuentra con BIN1 en $AA o $BB se coloca ese valor en ambos BCD1 y BCD2 para ponerlos en la pantalla de 7 segmentos y retorna. En otro caso se convierte cada valor a BCD llamando a BIN_BCD. Luego de la conversión se debe revisar si la dígito de las decenas es 0 en ese caso hay que poner un $B en ese nibble para saber que debe permanecer apagado el dígito correspondiente. Esto se logra poniendo un $B0 en R1 y luego haciendo una O exclusiva con cada BCD.

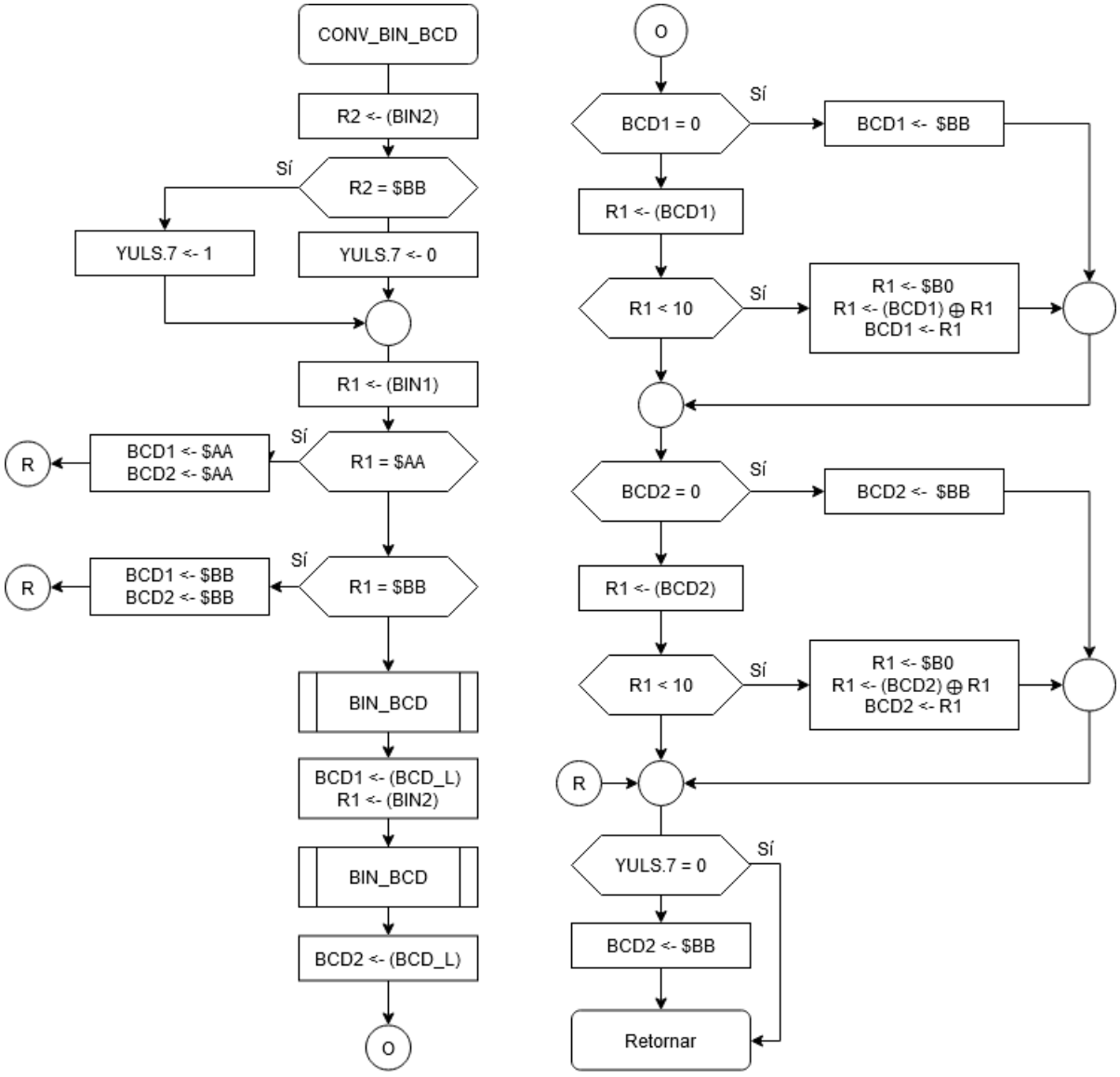

Figura 24: Diagrama de flujo del CONV_BIN_BCD.

3.2. Pruebas de Funcionalidad

4. Conclusiones y comentarios

Se logró diseñar de manera satisfactoria un programa en ensamblador para la arquitectura HCS12 encargado de controlar el conteo de vueltas y velocidad de un ciclista en un velódromo.

Se configuraron una variedad de periféricos para realizar tareas de conteos de tiempo que funcionan para crear controladores de hardware para otros circuitos integrados como lo es el contralador de la pantalla LCD. Para ese caso se configuró el timer counter para realizar interrupciones cada 20 microsegundos y así satisfacer las necesidades de tiempos de datos válidos en su bus de datos.

Se implentó el uso de bits en registros destinados como registros de banderas para indicar el estado del dispositivo y las acciones requeridas en cada caso como lo son activar salidas (poner valores en el puerto B), habilitar y deshabilitar interrupciones y colocar valores en diferentes variables para realizar cálculos de manera adecuada en todo el programa.

Se verificó el funcionamiento de programa mediante una batería de pruebas pensada en diferentes casos para verificar el funcionamiento de las partes programadas y de los casos para los cuales el programa no fue diseñado propiamente para revisar si su funcionamiento era el adecuado según los requerimientos.

El programa es totalmente funcional y fue gratificante que funcionara tras la enorme cantidad de tiempo y esfuerzo que requirió tener el cuidado de colocar cada símboloo de # o $ y cada bit como el programa lo requiería.

5. Recomendaciones

Usar un sistema de control de versiones como git, usar github fue indispensable para poder realizar este proyecto de manera exitosa.

Se recomienda preparar una cantidad de tiempo semanal de al menos 55 horas para poder realizar el trabajo de diseño codificacion y pruebas.

Se recomienda realizar pruebas por partes pequeñas del programa para encontrar errores más rápidamente en vez de codificar una gran parte y probar todo al mismo tiempo.

Referencias

[1] T. Almy, *Designing with microcontrollers The 68HCS12*, 2011st ed. Tom Almy, 2011.

[2] R. González, "Modos de direccionamiento en la arquitectura hcs12," 2020.

[3] G. Delgado, *RunMeter623*, verano ed. UCR, 2021.