

UNIVERSIDAD DE COSTA RICA
Escuela de Ingeniería Eléctrica
IE623 – Micrprocesadores

Proyecto final RADAR 623

Yeison Rodríguez Pacheco, B56074

09/12/19

Índice

1. Declaración de estructuras de datos del programa	7
2. Diseño de la aplicación	13
2.1. Programa principal: Inicialización del hardware	13
2.1.1. Descripcion:	13
2.1.2. Parámetros de entrada:	13
2.1.3. Parámetros de salida:	13
2.1.4. Explicación detallada:	13
2.2. Programa principal: Main Loop	15
2.2.1. Descripcion:	15
2.2.2. Parámetros de entrada:	15
2.2.3. Parámetros de salida:	15
2.2.4. Explicación detallada:	15
2.3. Subrutina MODO_CONFIG	18
2.3.1. Descripcion:	18
2.3.2. Parámetros de entrada:	18
2.3.3. Parámetros de salida:	18
2.3.4. Explicación detallada:	18
2.4. Subrutina BCD_BIN	19
2.4.1. Descripcion:	19
2.4.2. Parámetros de entrada:	19
2.4.3. Parámetros de salida:	19
2.4.4. Explicación detallada:	20
2.5. Subrutina TAREA_TECLADO	20
2.5.1. Descripcion:	20
2.5.2. Parámetros de entrada:	20
2.5.3. Parámetros de salida:	21
2.5.4. Explicación detallada:	21
2.6. Subrutina MUX_TECLADO	22
2.6.1. Descripcion:	22
2.6.2. Parámetros de entrada:	22
2.6.3. Parámetros de salida:	22
2.6.4. Explicación detallada:	23
2.7. Subrutina FORMAR_ARRAY	24
2.7.1. Descripcion:	25
2.7.2. Parámetros de entrada:	25
2.7.3. Parámetros de salida:	25
2.7.4. Explicación detallada:	25
2.8. Subrutina LIBRE	26
2.8.1. Descripcion:	26
2.8.2. Parámetros de entrada:	26
2.8.3. Parámetros de salida:	26
2.8.4. Explicación detallada:	27
2.9. Subrutina MODO_MEDICION	27

2.9.1. Descripcion:	27
2.9.2. Parámetros de entrada:	28
2.9.3. Parámetros de salida:	28
2.9.4. Explicación detallada:	28
2.10. Subrutina PANT_CTRL	28
2.10.1. Descripcion:	28
2.10.2. Parámetros de entrada:	28
2.10.3. Parámetros de salida:	29
2.10.4. Explicación detallada:	29
2.10.5. Memoria de cálculo	29
2.11. Subrutina CONV_BIN_BCD	31
2.11.1. Descripcion:	31
2.11.2. Parámetros de entrada:	31
2.11.3. Parámetros de salida:	31
2.11.4. Explicación detallada:	31
2.12. Subrutina BIN_BCD	32
2.12.1. Descripcion:	32
2.12.2. Parámetros de entrada:	32
2.12.3. Parámetros de salida:	32
2.12.4. Explicación detallada:	32
2.13. Subrutina BCD_7SEG	33
2.13.1. Descripcion:	33
2.13.2. Parámetros de entrada:	33
2.13.3. Parámetros de salida:	33
2.13.4. Explicación detallada:	34
2.14. Subrutina PATRON_LEDs	35
2.14.1. Descripcion:	35
2.14.2. Parámetros de entrada:	35
2.14.3. Parámetros de salida:	35
2.14.4. Explicación detallada:	35
2.15. Subrutina LCD	35
2.15.1. Descripcion:	36
2.15.2. Parámetros de entrada:	36
2.15.3. Parámetros de salida:	36
2.15.4. Explicación detallada:	36
2.16. Subrutina CARGAR_LCD	37
2.16.1. Descripcion:	37
2.16.2. Parámetros de entrada:	37
2.16.3. Parámetros de salida:	37
2.16.4. Explicación detallada:	37
2.17. Subrutina SEND	38
2.17.1. Descripcion:	38
2.17.2. Parámetros de entrada:	38
2.17.3. Parámetros de salida:	39
2.17.4. Explicación detallada:	39

2.18. Subrutina DELAY	39
2.18.1. Descripcion:	39
2.18.2. Parámetros de entrada:	39
2.18.3. Parámetros de salida:	40
2.18.4. Explicación detallada:	40
2.19. Subrutina de atención de interrupciones ATD_ISR	40
2.19.1. Descripcion:	40
2.19.2. Parámetros de entrada:	40
2.19.3. Parámetros de salida:	40
2.19.4. Explicación detallada:	41
2.19.5. Memoria de cálculo:	41
2.20. Subrutina de atención de interrupciones CALCULAR	41
2.20.1. Descripcion PH3:	42
2.20.2. Parámetros de entrada PH3:	42
2.20.3. Parámetros de salida PH3:	42
2.20.4. Explicación detallada PH3:	42
2.20.5. Descripcion PH0:	42
2.20.6. Parámetros de entrada PH0:	42
2.20.7. Parámetros de salida PH0:	42
2.20.8. Explicación detallada PH0:	43
2.20.9. Memoria cálculo PH0:	43
2.21. Subrutina de atención de interrupciones TCNT_ISR	44
2.21.1. Descripcion:	45
2.21.2. Parámetros de entrada:	45
2.21.3. Parámetros de salida:	45
2.21.4. Explicación detallada:	45
2.21.5. Memoria de cálculo:	45
2.22. Subrutina de atención de interrupciones RTI_ISR	46
2.22.1. Descripcion:	46
2.22.2. Parámetros de entrada:	46
2.22.3. Parámetros de salida:	47
2.22.4. Explicación detallada:	47
2.22.5. Memoria de cálculo:	47
2.23. Subrutina de atención de interrupciones OC4_ISR	47
2.23.1. Descripcion:	47
2.23.2. Parámetros de entrada:	48
2.23.3. Parámetros de salida:	48
2.23.4. Explicación detallada:	48
2.23.5. Memoria de cálculo:	48
3. Protocolo de pruebas realizado.	50
3.1. Prueba 1.	50
3.2. Prueba 2.	51
3.3. Prueba 3.	51
3.4. Prueba 4.	51
3.5. Prueba 5.	51

3.6. Prueba 6.	51
3.7. Prueba 7.	51
3.8. Prueba 8.	51
3.9. Prueba 9.	51
3.10. Prueba 10.	51
3.11. Prueba 11.	52
3.12. Prueba 12.	52
3.13. Prueba 13.	52
3.14. Prueba 14.	52
4. Resultados	52
5. Conclusiones y recomendaciones	52

Índice de figuras

1. Inicialización del hardware y variables.	14
2. Programa principal.	17
3. Subrutina MODO_CONFIG	19
4. Subrutina BCD_BIN	20
5. Subrutina TAREA_TECLADO	22
6. Subrutina MUX_TECLADO	24
7. Subrutina FORMAR_ARRAY	26
8. Subrutina FORMAR_ARRAY	27
9. Subrutina MODO_MEDICION	28
10. Subrutina PANT_CTRL	30
11. Subrutina CONV_BIN_BCD	32
12. Subrutina BIN_BCD	33
13. Subrutina BCD_7SEG	34
14. Subrutina PATRON_LEDS	35
15. Subrutina LCD	36
16. Subrutina CARGAR_LCD	38
17. Subrutina SEND	39
18. Subrutina DELAY	40
19. Subrutina de atención de interrupción ATD_ISR	41
20. Subrutina de atención de interrupción CALCULAR	44
21. Subrutina de atención de interrupción TCNT_ISR	46
22. Subrutina de atención de interrupción RTI_ISR	47
23. Subrutina de atención de interrupción OC4_ISR	49
24. Tabla de tiempos esperados para el producto.	50

Resumen

Este proyecto consiste en el diseño de un sistema que despliega la información sobre la velocidad en Kilometros por hora que lleva un vehículo en carretera, utilizando dos sensores ultrasónicos colocados a 40 metros cada uno. La pantalla es colocada en la carretera a 200 metros del segundo sensor, en dicha pantalla se mostrará el mensaje "Modo medicion, Esperando" siempre que no haya pasado ningún vehículo por los sensores, en el momento en que un vehículo pase por el primer sensor se cambiará el mensaje "Esperando" por "Calculando".

Cuando el vehículo pase por el segundo sensor el sistema calcula la velocidad y en base a esto procederá a esperar el tiempo necesario para que el vehículo se encuentre a 100 metros de la pantalla, cuando esto ocurra se cambiará el mensaje "Calculando" por "SU VEL. VEL.LIM" y se mostrará en los display de 7 segmentos la velocidad límite a la que puede ir el vehículo y la velocidad actual que tiene.

En caso de que el vehículo vaya a mayor velocidad de la límite se hará un barrido de leds que sirven para indicar al conductor que debe reducir la velocidad. La pantalla muestra la velocidad hasta que el vehículo pasa la pantalla colocada en la carretera, en este momento se volverá a colocar el mensaje "Esperando" y se esperará a que vuelva a pasar otro conductor.

Debido a que la máquina necesita ser manipulada por los usuarios, se tiene un modo de configuración que es elegido por medio de dos switchs. Si ambos switchs están abajo (OFF,OFF) se entra en modo configuración, el cual permite configurar la velocidad máxima del sistema, esta velocidad puede estar entre 45 KM/H y 90 KM/H, en caso de ingresar un valor fuera de este rango se pondrá un 0 en dicha velocidad máxima. Al arranque del sistema la velocidad está en 0 por lo que no se puede cambiar a otro modo a menos que se asigne una velocidad máxima válida, en este momento se podrá ir al modo medicion (ON,ON) que fue descrito anteriormente. El mensaje mostrado en el modo configuración es "MODO CONFIG, VELOC. LIMITE" y se mostrará la velocidad límite programada en los display de 7 segmentos.

Por último se tiene el modo libre (ON,OFF ó OFF,ON) el cual solo muestra en la pantalla el siguiente mensaje "RADAR 623, MODO LIBRE", esto con el fin de que se quiera dejar la máquina sin funcionar en algún momento, como por ejemplo en las noches.

El diseño de este sistema se realizó en la tarjeta de desarrollo Dragon 12, que tiene integrado un procesador MC9S12DG256, los sensores fueron implementados como botones (PH3 y PH0), las pantallas consisten en una LCD y 4 display de 7 segmentos, el teclado de entrada es un teclado matricial de 3x4, los switchs consisten en PH7 y PH6, y por último las luces de alarma y modo son los leds de PB0 a PB7.

La codificación fue realizada en lenguaje ensamblador para la arquitectura FREESCALE, el programa mide 1813 bytes con un total de ciclos de 1979. Se realizó un robusto protocolo de pruebas para probar el funcionamiento del sistema y funciona correctamente en todos los escenarios como se mostrará en este informe.

1. Declaración de estructuras de datos del programa

En esta sección se declararán con detalle todas las estructuras de datos utilizadas en el programa, con el fin de que cuando se muestre el diseño de la aplicación se puedan realizar consultas a dichas variables para ver sus especificaciones. También se presentará su posición en hexadecimal que tiene en la memoria.

■ \$ 1000 a \$ 1001 .

BANDERAS: Variable tipo word que almacena todas las banderas del sistema. A continuación se presenta la distribución de cada uno de los bits comenzando por el MSB

X : X : X : MOD_LIB_ACTUAL : MOD_CONF_ACTUAL : MOD_MED_ACTUAL : PH3_EN : PH0_EN
COMANDO_DATO : X : CALC_TICKS : ALERTA : PANT_FLAG : ARRAY_OK : TCL_LEIDA : TCL_LISTA

Ahora se mostrará una explicación de en qué se utiliza cada uno de estos bits.

MOD_LIB_ACTUAL: Este bit permanece en 1 al encendido del sistema con el fin de que al entrar al modo libre se revise este bit y si está en 1 significa que se viene de un cambio de modo, por lo que se actualiza la LCD y demás variables. En dicho instante este bit se pone en 0 con lo cual solo se actualizan estos valores una vez cada cambio de modo. Dicho bit es utilizado en la subrutina LIBRE.

MOD_CONF_ACTUAL: Este bit permanece en 1 al encendido del sistema con el fin de que al entrar al modo configuración se revise este bit y si está en 1 significa que se viene de un cambio de modo, por lo que se actualiza la LCD y demás variables. En dicho instante este bit se pone en 0 con lo cual solo se actualizan estos valores una vez cada cambio de modo. Este bit es utilizado en el MAIN.

MOD_MED_ACTUAL: Este bit permanece en 1 al encendido del sistema con el fin de que al entrar al modo medición se revise este bit y si está en 1 significa que se viene de un cambio de modo, por lo que se actualiza la LCD y demás variables. En dicho instante este bit se pone en 0 con lo cual solo se actualizan estos valores una vez cada cambio de modo. Este bit es utilizado en el MAIN.

PH3_EN: Este bit se pone en 1 al encendido del sistema y cada vez que se entre en modo medición, ya que es un habilitador del botón PH3 que es el que indica que pasó un automóvil por el sensor. Al presionarse PH3 este bit se pone en 0 y es puesto en 1 nuevamente cuando el carro termina de pasar la pantalla o si se vuelve a entrar a modo medición.

PH0_EN: Este bit se pone en 0 al encendido del sistema y cada vez que se entre en modo medición, ya que es un habilitador del botón PH0 que es el que indica que pasó un automóvil por el sensor 2. Al presionarse PH3 este bit se pone en 1 y es puesto en 0 nuevamente cuando se presiona PH0 o si se cambia de modo.

COMANDO_DATO: Este bit indica con un 0 que se va a enviar un comando a la LCD en la subrutina SEND, con un 1 indica que se quiere enviar un dato.

CALC_TICKSL: Esta bandera indica con un 1 que ya se realizó el cálculo de cuánto tiempo debe pasar para que el automóvil llegue a la pantalla en 100m y cuánto tiempo debe pasar para que la pantalla se apague nuevamente. Este cálculo es hecho en la subrutina PANT_CTRL y la importancia de este bit radica en que el cálculo solo se haga una vez por auto.

ALERTA : Este bit cuando se pone en 1 provoca que se genere el encendido de los leds de ALARMA, debido a que la interrupción OC4 llama constantemente a la subrutina LEDS_ALARMA que controla este comportamiento revisando este bit.

PANT_FLAG: Este bit indica en el modo medición cuánto se debe encender la pantalla de 7 segmentos con un 1, si está en 0 la pantalla está apagada. Este bit es modificado por la interrupción

de TO.

ARRAY_OK: Esta bandera indica cuando un array ingresado en el teclado por el usuario ya es válido (se presionó enter).

TCL_LEIDA: Bandera utilizada para saber si ya se leyó una tecla en la subrutina TAREA_TECLADO.

TCL_LISTA: Este bit es utilizado para saber si ya existe una tecla lista, y solo se está esperando que el usuario suelte el botón para validarla revisando este bit.

- **\$ 1002.**

V_LIM: Variable tipo byte que almacena la velocidad límite del vehículo, solo tiene valores válidos del 45 al 90 para este programa. Otro valor válido es 0 y así está por defecto al encendido del sistema.

- **\$ 1003.**

MAX_TECLA: Constante tipo byte que almacena la cantidad máxima de teclas a poder ser ingresadas en el teclado. Para este programa tiene un valor de 2.

- **\$ 1004.**

TECLA: Variable tipo byte que es utilizada por la subrutina MUX_TECLADO para guardar la tecla leída (si es que hay alguna presionada). En caso de no encontrarse una tecla se devuelve el valor \$FF.

- **\$ 1005.**

TECLA_IN: Variable tipo byte que es utilizada por la subrutina TAREA_TECLADO para guardar una tecla leída en algún momento y validar posteriormente de que pase el control de rebotes que la tecla sigue presionada.

- **\$ 1006.**

CONT_REB: Variable tipo byte que es utilizada para controlar los rebotes de todos los botones mecánicos del sistema. Esta subrutina es decrementada aproximadamente cada 1ms por la subrutina RTI.

- **\$ 1007.**

CONT_TCL: Variable tipo byte utilizada por la subrutina FORMAR_ARRAY con el fin de llevar un control de la cantidad de teclas leídas para ver si ya se llegó al valor de la constante MAX_TECLA.

- **\$ 1008.**

PATRON: Variable tipo byte utilizada por la subrutina MUX_TECLADO para contar hasta 5 con el fin de realizar una correcta multiplexación del teclado.

- **\$ 1009 a \$ 100A .**

NUM_ARRAY: Tabla tipo byte que almacena al inicio los valores \$ff en todas sus posiciones, con el fin de indicar que el array está vacío.

- **\$ 100B.**

BRILLO: Variable tipo byte utilizada por la interrupción OC4 con el fin de controlar el brillo que tiene la pantalla. Es modificada por la interrupción del ATD leyendo el potenciómetro para modificar dicho brillo. Su valor solo oscila en una escala del 0 al 100.

▪ \$ 100C.

POT: Variable tipo byte que almacena el valor leído por el potenciómetro en la interrupción del convertidor analógico digital.

▪ \$ 100D a \$ 100E .

TICK_EN: Variable tipo word que se utiliza para contar ticks en la interrupción de TO. Cada tick equivale a 0.0218 s. Cuando el valor de esta variable llega a 0 pone la bandera PANT_FLAG en 1 con el fin de que se encienda la pantalla de 7 segmentos en el modo medición.

▪ \$ 100F a \$ 1010 .

TICK_DIS: Variable tipo word que se utiliza para contar ticks en la interrupción de TO. Cada tick equivale a 0.0218 s. Cuando el valor de esta variable llega a 0 pone la bandera PANT_FLAG en 0 con el fin de que se apague la pantalla de 7 segmentos en el modo medición.

▪ \$ 1011.

VELOC: Variable tipo byte utilizada para almacenar la velocidad a la que va el vehículo en KM/H.

▪ \$ 1012.

TICK_VEL: Variable tipo byte que se utiliza para contar ticks en la interrupción de TO. Cada tick equivale a 0.0218 s. Con el valor de esta variable al presionar PH0 se realiza el cálculo de la velocidad en KM/H para ser almacenada en VELOC.

▪ \$ 1013.

BIN1: Variable tipo byte que se utiliza para almacenar el valor en binario que será desplegado en los display de la derecha de la pantalla de 7 segmentos.

▪ \$ 1014.

BIN2: Variable tipo byte que se utiliza para almacenar el valor en binario que será desplegado en los display de la izquierda de la pantalla de 7 segmentos.

▪ \$ 1015.

BCD1: Variable tipo byte que se utiliza para almacenar el valor en BCD que será desplegado en los display de la derecha de la pantalla de 7 segmentos.

▪ \$ 1016.

BCD1: Variable tipo byte que se utiliza para almacenar el valor en BCD que será desplegado en los display de la izquierda de la pantalla de 7 segmentos.

▪ \$ 1017.

BCD_L: Variable tipo byte que se utiliza para almacenar el valor convertido a BCD por la subrutina BIN_BCD. Dicha subrutina retorna su resultado en este parámetro

▪ \$ 1018.

BCD_H: Variable tipo byte que se utiliza en la subrutina BIN_BCD como variable auxiliar para uno de los procedimientos internos de la misma.

▪ \$ 1019.

BCD_H: Variable tipo byte que se utiliza en la subrutina BIN_BCD como variable auxiliar para uno de los procedimientos internos de la misma.

■ \$ 101A.

DISP1: Variable tipo byte que se utiliza para almacenar los valores en codificación de 7 segmentos de los números a ser desplegados en el display más a la derecha de la dragon 12.

■ \$ 101B.

DISP2: Variable tipo byte que se utiliza para almacenar los valores en codificación de 7 segmentos de los números a ser desplegados en el display 2 de la dragon 12.

■ \$ 101C.

DISP3: Variable tipo byte que se utiliza para almacenar los valores en codificación de 7 segmentos de los números a ser desplegados en el display 3 de la dragon 12.

■ \$ 101D.

DISP4: Variable tipo byte que se utiliza para almacenar los valores en codificación de 7 segmentos de los números a ser desplegados en el display más a la izquierda de la dragon 12.

■ \$ 101E.

LEDS: Variable tipo byte que se utiliza para almacenar el valor de los leds que serán encendidos en la dragon 12.

■ \$ 101F.

CONT_DIG: Variable tipo byte que es utilizada por la interrupción OC4 con el fin de realizar el control de cuál de los display se va a encender, por lo que dicha variable solo toma valores de 0 a 4.

■ \$ 101F.

CONT_TICKS: Variable tipo byte que es utilizada por la interrupción OC4 con el fin de contar ticks y de esta forma apagar en cierto momento los display, con el fin de controlar su brillo reduciendo o aumentando su ciclo de trabajo.

■ \$ 1020.

DT: Variable tipo byte que es utilizada por la interrupción OC4 con el fin de calcular el ciclo de trabajo de los displays.

■ \$ 1021 a \$ 1022.

CONT_7SEG: Variable tipo word que es utilizada por la interrupción OC4 para contar hasta 100ms, el valor que debe alcanzar desde 0 para tomar este tiempo es de 5000. Cada cuenta de esta variable equivale a 20us.

■ \$ 1023 a \$ 1024.

CONT_200: Variable tipo word que es utilizada por la interrupción OC4 para contar hasta 20ms, el valor que debe alcanzar desde 0 para tomar este tiempo es de 10000. Cada cuenta de esta variable equivale a 20us.

■ \$ 1025.

CONT_DELAY: Variable tipo byte que es utilizada por la interrupción OC4 con el fin de controlar los delay requeridos por la pantalla LCD. Cada cuenta de esta variable equivale a 20us.

■ \$ 1026.

D2mS: Constante tipo byte con un valor de 100 que es utilizada para cargar CONT_DELAY y contar hasta 2ms con el fin de respetar la temporización de la pantalla LCD.

■ \$ 1027.

D260uS: Constante tipo byte con un valor de 100 que es utilizada para cargar CONT_DELAY y contar hasta 260us con el fin de respetar la temporización de la pantalla LCD.

■ \$ 1028.

D60uS: Constante tipo byte con un valor de 100 que es utilizada para cargar CONT_DELAY y contar hasta 60us con el fin de respetar la temporización de la pantalla LCD.

■ \$ 1029.

CLEAR_LCD: Variable utilizada en las subrutinas que controlan la pantalla LCD.

■ \$ 102A.

ADD_L1: Constante utilizada en las subrutina CARGAR_LCD con el fin de contener la dirección de inicio de la pantalla LCD para desplegar el mensaje en al parte superior.

■ \$ 102B.

ADD_L2: Constante utilizada en las subrutina CARGAR_LCD con el fin de contener la dirección de inicio de la pantalla LCD para desplegar el mensaje en al parte inferior.

■ \$ 1030 a \$ 103B.

TECLAS: Tabla tipo byte que contiene las traducciones de los valores de las teclas presionadas en el teclado matricial de 3x4. Tiene 12 campos de longitud, sus valores son: \$01, \$02, \$03, \$04, \$05, \$06, \$07, \$08, \$09, \$0B, \$0, \$0E.

■ \$ 1040 a \$ 104B.

SEGMENT: Tabla tipo byte que contiene las traducciones de los números del 0 al 9 en código de 7 segmentos. Los últimos dos valores son utilizados para poner guiones y apagar los display respectivamente. Tiene 12 campos de longitud, sus valores son: \$3f, \$06, \$5B, \$4F, \$66, \$6D, \$7D, \$07, \$7F, \$6E, \$40, \$00.

■ \$ 1050 a \$ 1055.

iniDsp: Tabla tipo byte que contiene códigos de inicialización de la pantalla LCD Tiene 6 campos de longitud, sus valores son: \$04, \$28, \$28, \$06, \$0C, \$FF.

■ \$ 1060 a \$ 1070.

Msj_config_1: String que contiene el mensaje " MODO CONFIG. " utilizado para el modo configuración. Este mensaje es desplegado en la pantalla LCD.

■ \$ 1071 a \$ 1081.

Msj_config_2: String que contiene el mensaje " VELOC. LIMITE " utilizado para el modo configuración. Este mensaje es desplegado en la pantalla LCD.

■ \$ 1082 a \$ 1092.

Msj_libre_1: String que contiene el mensaje " RADAR 623 " utilizado para el modo libre. Este mensaje es desplegado en la pantalla LCD.

- **\$ 1093 a \$ 10A3.**

Msj_libre_2: String que contiene el mensaje " MODO LIBRE " utilizado para el modo libre. Este mensaje es desplegado en la pantalla LCD.

- **\$ 10A4 a \$ 10B4.**

Msj_medicion_1: String que contiene el mensaje " MODO MEDICION " utilizado para el modo medición. Este mensaje es desplegado en la pantalla LCD.

- **\$ 10B5 a \$ 10C5.**

Msj_medicion_calculando_2: String que contiene el mensaje " CALCULANDO... " utilizado para el modo medición cuando se presiona el botón ph3 y hasta que se muestra la velocidad. Este mensaje es desplegado en la pantalla LCD.

- **\$ 10C6 a \$ 10D6.**

Msj_medicion_esperando_2: String que contiene el mensaje " ESPERANDO... " utilizado para el modo medición cuando se está en modo reposo. Este mensaje es desplegado en la pantalla LCD.

- **\$ 10D7 a \$ 10E7.**

Msj_medicion_su_vel_vel_lim_2: String que contiene el mensaje "SU VEL. VEL.LIM " utilizado para el modo medición cuando el carro se encuentra a 100 metros de la pantalla. Este mensaje es desplegado en la pantalla LCD.

2. Diseño de la aplicación

En esta sección se procederá a mostrar el diseño de software de toda la aplicación, explicando cada subrutina del programa así como mostrando sus parámetros de entrada y salida y su respectiva forma de pasarlo u obtenerlos.

2.1. Programa principal: Inicialización del hardware

A continuación se presenta con detalle las especificaciones de la inicialización del hardware. En la figura [1] tenemos el diagrama de flujos de este proceso.

2.1.1. Descripción:

Esta subrutina de inicialización de hardware solo se ejecuta una vez e inicializa todo el hardware necesario en la ejecución del programa. También se presenta la inicialización de las variables del sistema.

2.1.2. Parámetros de entrada:

Los que se inicializan en el bloque, como TECLA, TECLA_IN, CONT_REB, BANDERAS, CONT_TCL, V_LIM, VELOC, CONT_DIG, CONT_TICKS, CONT_7SEG.

2.1.3. Parámetros de salida:

No tiene.

2.1.4. Explicación detallada:

Como vemos en la figura 1 lo primero que se hace es inicializar el convertidor analógico digital y configurarlo con borrado automático. Se hace un loop que espera un tiempo con el fin de que el CAD se encienda de manera correcta. Por último se configuran 6 conversiones por canal a una frecuencia de 500KHZ y con 4 periodos de muestreo.

Seguidamente se activan las resistencias de pull up de los puertos core y se pone la parte alta del puerto A como salida, todo esto para configurar la lectura del teclado matricial.

Lo siguiente que se configura es la interrupción RTI con una frecuencia de 1ms aproximadamente. También se realiza la habilitación del módulo de timers con un preescalador igual a 8 y solo habilitando la interrupción de output compare del canal 4. Además de cargar el registro TC4 con TCNT + 60, esto con el fin de generar una interrupción cada 20us.

Se pone como salidas todo el puerto B y los primeros 4 bits del puerto P, con el fin de utilizar los display de 7 segmentos, además de que se pone como salida el bit 1 del puerto J que controla los leds. Lo último que se configura es el puerto K como salida, para utilizar la LCD.

Seguidamente se inicializan todas las variables necesarias para el correcto funcionamiento del programa y por último se llama la subrutina LCD que deja lista la pantalla LCD para desplegar mensajes en ella. Esta subrutina solo se ejecuta una vez y se salta al Main loop que se ejecutará siempre. Esto se discutirá más adelante.

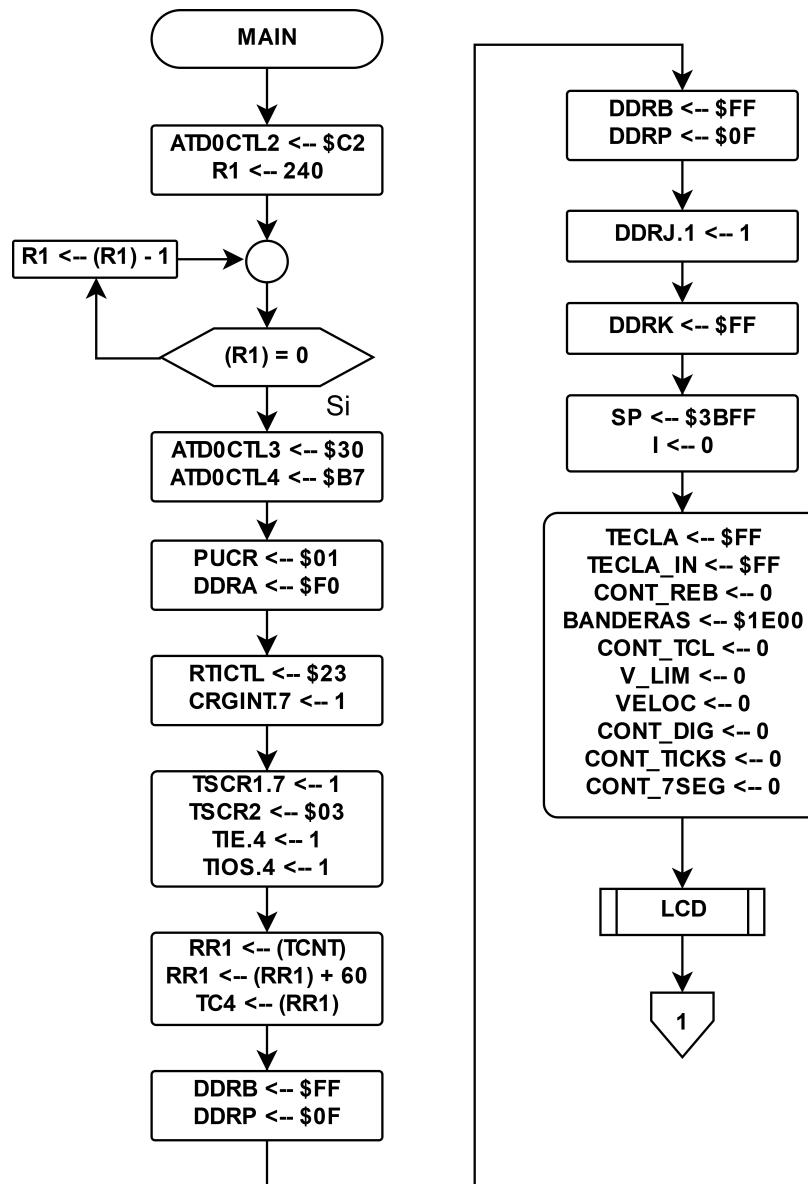


Figura 1: Inicialización del hardware y variables.

2.2. Programa principal: Main Loop

En la figura [2] se presenta el programa principal, que se ejecutará todo el tiempo.

2.2.1. Descripcion:

El programa principal se encarga de realizar el control de los modos y llamar a las subrutinas correspondientes. También realiza el control de la actualización de algunos mensajes de la LCD.

2.2.2. Parámetros de entrada:

Todos estos parámetros son pasados por memoria al programa principal.
PTIH, MOD_CONF_ACTUAL, MOD_LIB_ACTUAL ,MOD_MED_ACTUAL.

2.2.3. Parámetros de salida:

Todos estos parámetros son pasados por memoria al programa principal.
LEDS, PH3, PH0, VELOC,PANT_FLAG,ALERTA, TICK_EN, TICK_DIS,CALC_TICKS, TSCR2, PIEH, BIN1, BIN2.

2.2.4. Explicación detallada:

Después de haberse inicializado todo el hardware y las variables necesarias, se entra a esta sección del main que es un loop que se repetirá infinitamente (figura [2]). Lo primero que se hace es verificar si la velocidad límite es 0, si es así (como está programado al inicio del sistema) solo se puede estar en el modo configuración, esto debido a que se requiere que el usuario programe la velocidad límite. En caso de que la velocidad no sea 0, significa que el usuario ya programó la velocidad límite. En este escenario se pregunta por la posición de los switch primero revisando si se está en modo medición, de ser así se verifica la bandera MOD_MED_ACTUAL y la va a encontrar en 1, en este momento se pone MOD_MED_ACTUAL en 0 para no volver a actualizar la pantalla mientras no se cambie de modo, las banderas MOD_LIB_ACTUAL y MOD_CONF_ACTUAL se ponen en 1 para cuando se cambie de modo que se actualice la LCD respectivamente. Otra operación en modo medición que solo ocurre la primera vez es actualizar el led de este modo, activar el botón PH3 y activar las interrupciones de TO y Key Wake Ups. Por último se llamará recurrentemente a modo medición hasta que se encuentren los switch en otra posición por lo que se cambiará de modo.

En caso de que no se esté en modo medición, se borra la velocidad, la activación de los botones PH3 y PH0, la bandera PANT_FLAG, ALERTA, TICK_EN y TICK_DIS. Estas variables se deben borrar debido a que el cambio de modo medición a otro puede llegar en cualquier momento, por lo que se cancela todo posible cálculo o configuración que se haya hecho en modo medición. Seguidamente se pregunta si está en modo configuración, de ser así se pregunta por MOD_CONF_ACTUAL y lo va a encontrar en 1 solamente si es la primer entrada a este modo, de ser así se pone en 0 esta bandera y se ponen en 1 las banderas restantes de cambio de modo, También ya que se está en modo configuración se carga el mensaje a la LCD, se enciende el LED de modo, se apagan los display de 7 segmentos de la izquierda y se desactivan las interrupciones de TO y Key wake ups. Por último siempre que se esté en modo configuración se llamará a la subrutina MODO_CONFIG que realiza el control de la configuración del usuario por el teclado.

El último caso es que se esté en modo Libre, en este caso se llama la subrutina modo libre que actualiza la LCD, apaga los display, desactiva las interrupciones TO y Key wake ups y actualiza las banderas de cambio de modo.

Este main tiene un pequeño inconveniente y es que en modo configuración y libre se borran las variables de manera recurrente en cada ciclo, pero esto no presenta ningún problema para el programa.

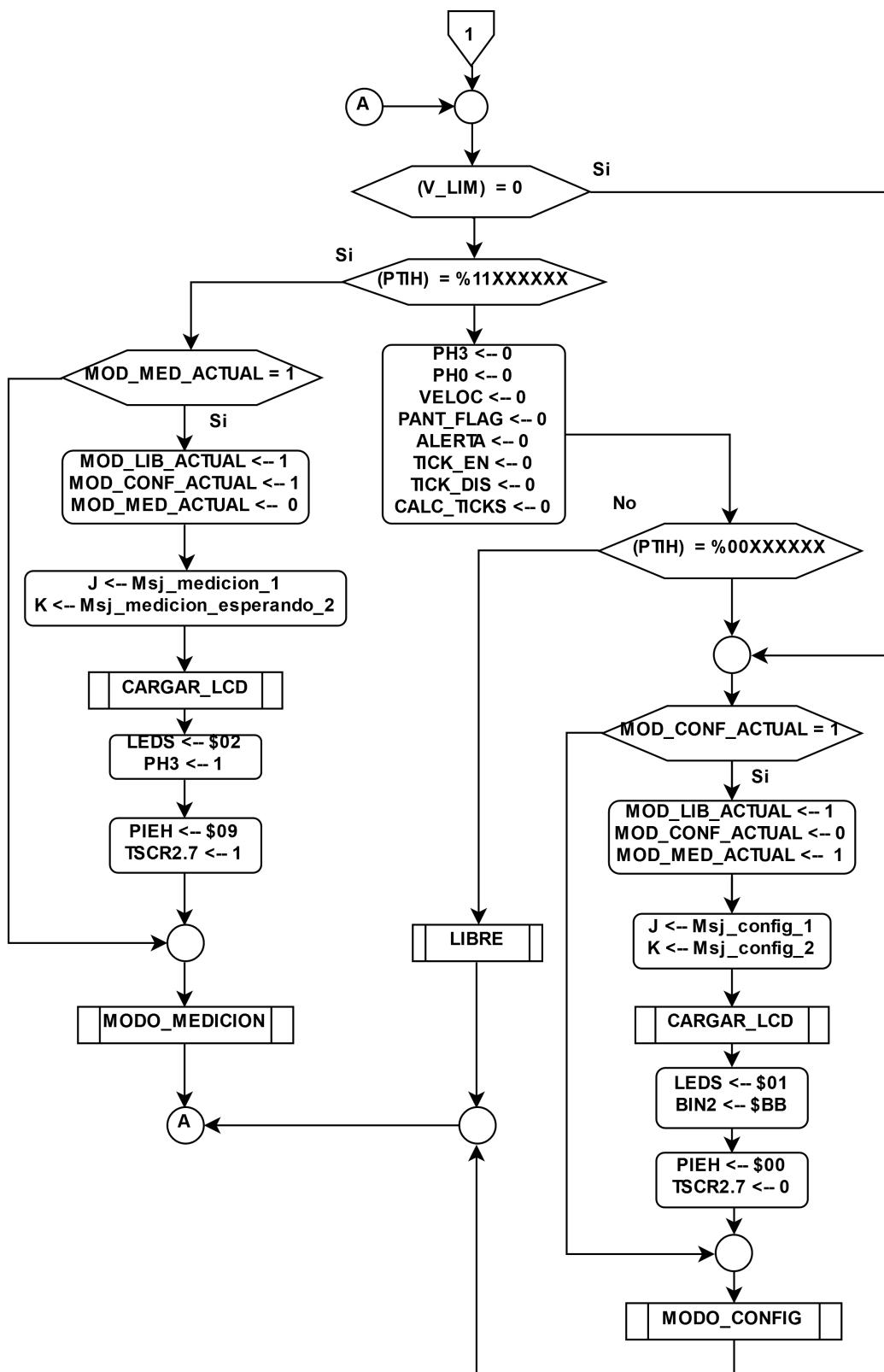


Figura 2: Programa principal.

2.3. Subrutina MODO_CONFIG

En la figura [3] se presenta el diagrama de flujo de esta subrutina.

2.3.1. Descripción:

Esta subrutina se encarga de hacer el control de todo el modo de configuración de la máquina, administrando la velocidad límite que ingresa el usuario por el teclado y verificando que sea válida.

2.3.2. Parámetros de entrada:

- ARRAY_OK: Este parámetro de entrada es pasado por memoria.

2.3.3. Parámetros de salida:

- BIN1: Este parámetro de salida es devuelto por memoria.
- V_LIM: Este parámetro de salida es devuelto por memoria.

2.3.4. Explicación detallada:

Esta subrutina se encarga de revisar si la bandera ARRAY_OK ya está en uno, esto significa que ya el usuario ingresó un valor válido en el teclado de Velocidad límite, cuando esto ocurre se llama a la subrutina BCD_BIN para que devuelva el valor en V_LIM de la velocidad ingresada por el usuario en binario. Esta subrutina verifica que dicha velocidad esté dentro del rango válido que es de 45 a 90 KM/H. En caso de que V_LIM no sea válido se pone en 0 para indicar que es un valor inválido. Si V_LIM es válido se guarda en BIN1 para que sea desplegado en la pantalla de 7 segmentos,

En caso de que el usuario no haya ingresado aún un valor de V_LIM se llama a la subrutina TAREA_TECLADO que realiza todo el procedimiento necesario para obtener las teclas pulsadas por el usuario. Además después de esto se guarda V_LIM en BIN1 y se retorna.

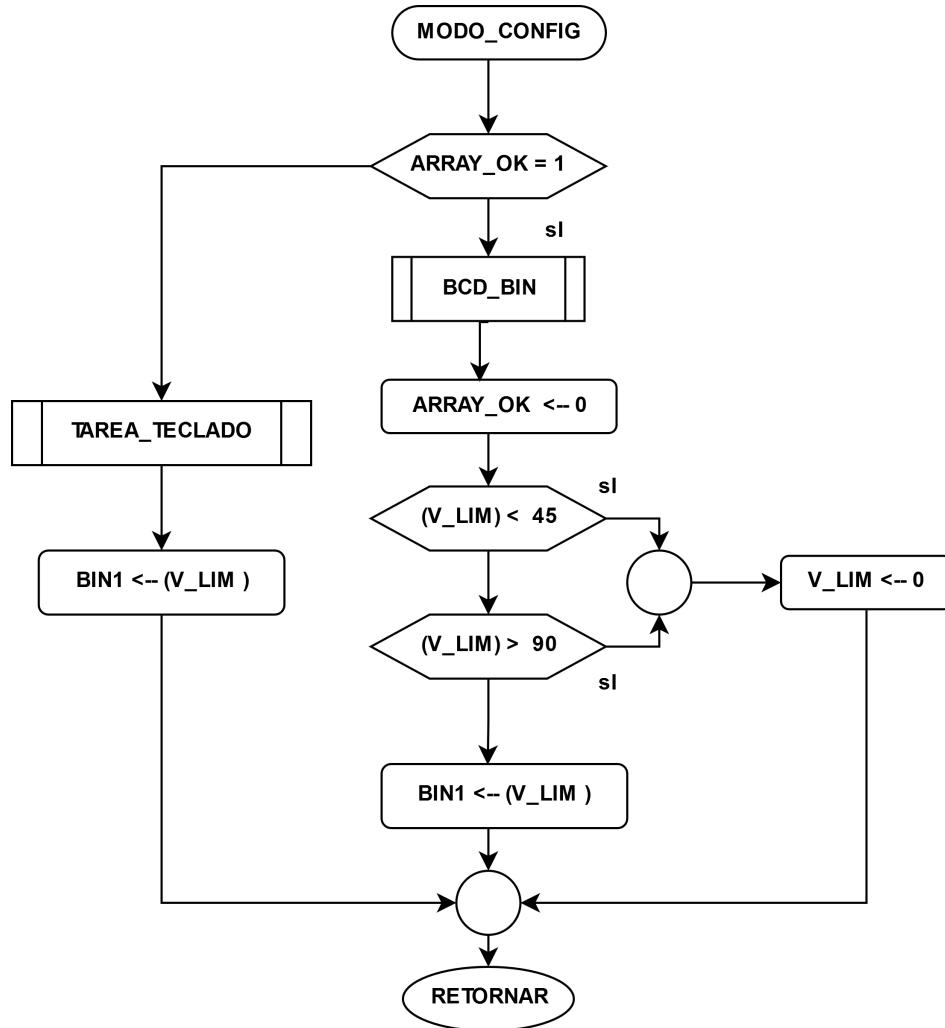


Figura 3: Subrutina MODO_CONFIG

2.4. Subrutina BCD_BIN

En la figura [4] se presenta el diagrama de flujos de esta subrutina.

2.4.1. Descripcion:

Esta subrutina se encarga de pasar valores de BCD a binario. El valor de velocidad binario es guardado en V_LIM.

2.4.2. Parámetros de entrada:

- NUM_ARRAY: Tabla tipo byte con los número ingresados en el teclado en código BCD. Obtenido por memoria.

2.4.3. Parámetros de salida:

- V_LIM: Variable tipo byte con la velocidad límite en binario. Transferido por memoria.

2.4.4. Explicación detallada:

Esta subrutina toma los valores de NUM_ARRAY y los convierte de BCD a binario, multiplicando por 10 el número de las decenas y sumándolo a las unidades. En caso de que NUM_ARRAY solo tenga un número válido, la subrutina se da cuenta y vuelve a llenar NUM_ARRAY con los valores \$FF, borrando este valor ya que no es válido. En caso de que el valor sea válido esta subrutina guarda en V_LIM la velocidad límite.

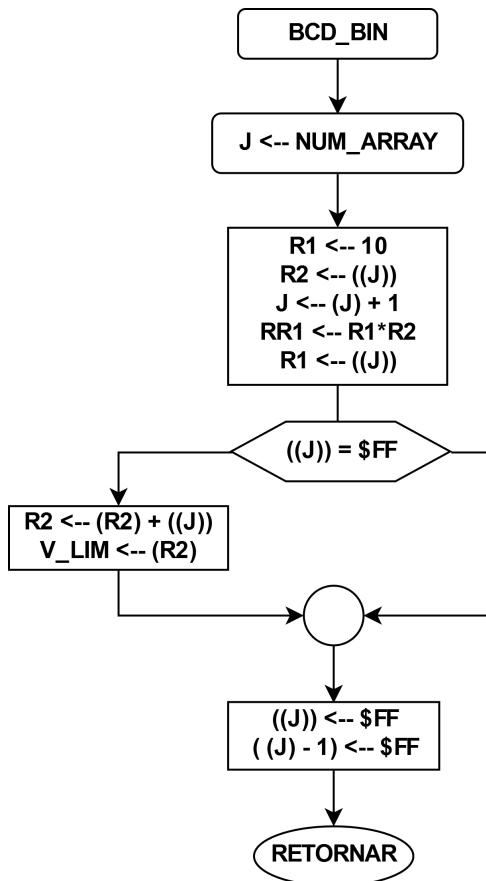


Figura 4: Subrutina BCD_BIN

2.5. Subrutina TAREA_TECLADO

En la figura [5] se presenta el diagrama de flujos de esta subrutina.

2.5.1. Descripción:

Esta subrutina realiza el control del teclado matricial de tamaño 3x4. Las teclas son validadas hasta que se suelta el botón de la misma. Se hace control de rebotes.

2.5.2. Parámetros de entrada:

- CONT_REB: Variable ingresada por memoria.

2.5.3. Parámetros de salida:

- NUM_ARRAY: Array devuelto por memoria, tamaño 2.
- ARRAY_OK: Bandera devuelta por memoria, tamaño 2.

2.5.4. Explicación detallada:

Esta subrutina debe ser llamada de manera recurrente para que funcione. En primer lugar se tiene que CONT_REB es 0 porque es el estado inicial que tiene esta variable, por lo que se llama a la subrutina MUX_TECLADO que realiza la lectura de una tecla y la devuelve por la variable TECLA, si el valor de TECLA es \$FF significa que no está presionada ninguna tecla del teclado, en caso de que sí se encuentre una tecla presionada se verifica con la bandera TCL_LEIDA si ya se leyó esta tecla anteriormente, como esto no ha pasado se pone la tecla leída TCL_IN, se pone en 1 la bandera TCL_LEIDA y se carga CONT_REB en 10.

Contador de rebotes es decrementado por la subrutina RTI una vez cada 1ms, por lo que deben pasar 10ms para que TAREA_TECLADO vuelva a encontrar CONT_REB en 0 y haga algo. En este momento se llama nuevamente a MUX_TECLADO y se obtiene la tecla presionada en TECLA, como TCL_LEIDA ya está en 1 se hace la pregunta si la tecla leída actualmente es igual a la tecla leída hace 10ms, si es así se pone la bandera TCL_LISTA en 1 y se retorna. En caso contrario se reinician las variables TECLA, TCL_IN y las banderas TCL_LEIDA y TCL_LISTA, para comenzar el ciclo nuevamente.

Si se vuelve a entrar a la subrutina y TCL_LISTA está en 1, se espera a que se suelte la tecla (verificando que TECLA sea \$FF), cuando esto ocurre significa que TCL_IN tiene una tecla válida y se llama a la subrutina FORMAR_ARRAY que se encarga de procesar dicha tecla. Por último se reinician las banderas y se vuelve a esperar la lectura de otra tecla.

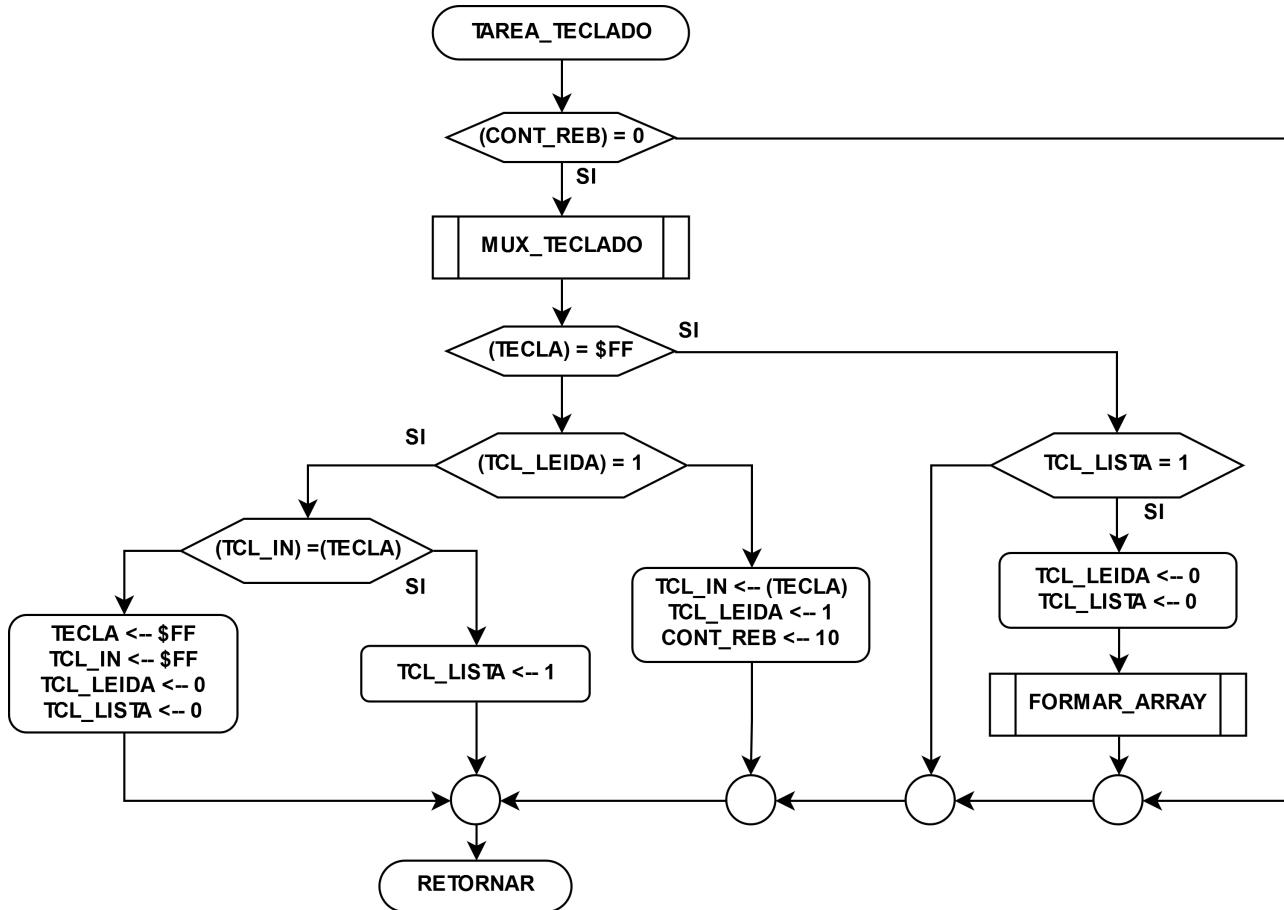


Figura 5: Subrutina TAREA_TECLADO

2.6. Subrutina MUX_TECLADO

En la figura [6] se presenta el diagrama de flujos de esta subrutina.

2.6.1. Descripción:

Esta subrutina se encarga de encontrar si se presionó una tecla en el teclado matricial de la dragon 12. Devuelve la tecla en la variable TECLA en BCD, en caso de que no se encuentre tecla presionada se devuelve un \$FF en la misma, indicando un valor inválido.

2.6.2. Parámetros de entrada:

- Puerto A: 3 pines del puerto A, comenzando del LSB.

2.6.3. Parámetros de salida:

- TECLA: Parámetro devuelto por memoria.

2.6.4. Explicación detallada:

Esta subrutina carga en el inicio TECLA en \$FF, patron en 1 y el acumulador A en \$EF. Seguidamente se escribe en los 4 bits de la parte alta del puerto el patrón \$EF, esto con el fin de escribir un 0 en la parte alta de PORTA. Seguidamente se procede a esperar en un loop de tamaño 10, con el fin de esperar un tiempo mientras se escribe el puerto A. Ahora se procede a buscar un 0 en alguna de las filas del teclado (3 bits menos significativos de puerto A), si se encuentra un cero significa que se encontró una tecla presionada por lo que se pasa a una lógica que calcula cuál tecla fue presionada a partir de la variable Patron, se guarda la tecla en TECLA accediendo la tabla TECLAS y se retorna.

Los patrones escritos en el puerto a son \$EF, \$DF,\$BF, \$7F , si en ninguno se encuentra un 0, se retorna devolviendo en TECLA \$FF.

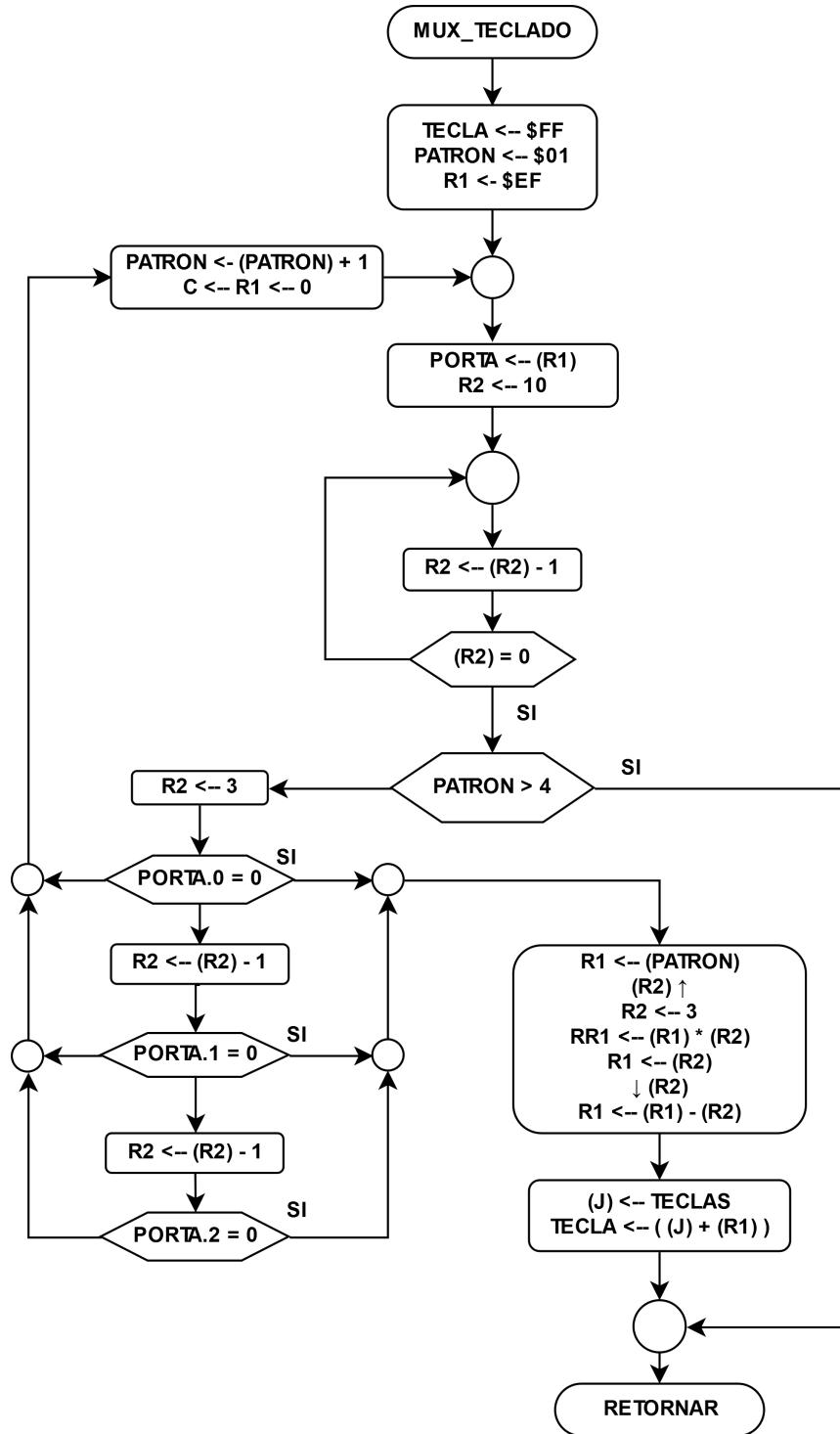


Figura 6: Subrutina MUX_TECLADO

2.7. Subrutina FORMAR_ARRAY

En la figura [7] se presenta el diagrama de flujos de esta subrutina.

2.7.1. Descripción:

Esta subrutina se encarga de realizar la lógica de control sobre las teclas presionadas con el fin de que se puedan borrar y validar.

2.7.2. Parámetros de entrada:

- MAX_TECLA: Parametro pasado por memoria.
- TECLA_IN: Parametro pasado por memoria.
- CONT_TCL: Parametro pasado por memoria.

2.7.3. Parámetros de salida:

- ARRAY_OK: Bandera devuelta por memoria.
- NUM_ARRAY: Arreglo devuelto por memoria.

2.7.4. Explicación detallada:

Al entrar a esta subrutina lo primero que se pregunta es si la cantidad de teclas leídas ya es la máxima aceptada, de ser así solo acepta dos posibles valores de teclas, enter y borrar. Si se presiona enter se valida el arreglo por lo que se pone la bandera ARRAY_OK en 1 y se borra el contador de teclas CONT_TCL. En caso de presionar borrar se borra la última tecla ingresada y se disminuye el CONT_TCL. Si el array aún no está lleno se valida nuevamente si la tecla presionada es enter, de ser así se verifica si CONT_TCL es 0, en este caso se ignora la tecla enter porque no se ha ingresado ningún dato válido al teclado, de haber al menos una tecla esta es validada con ARRAY_OK. Si la tecla no era un enter se verifica si es la tecla de borrar, si es así se valida que haya al menos una tecla ingresada para ser borrada, si no hay teclas ingresadas se ignora el botón de borrar. Si no es un enter ni un borrar significa que es un número por lo que se almacena en NUM_ARRAY y si aumenta CONT_TCL.

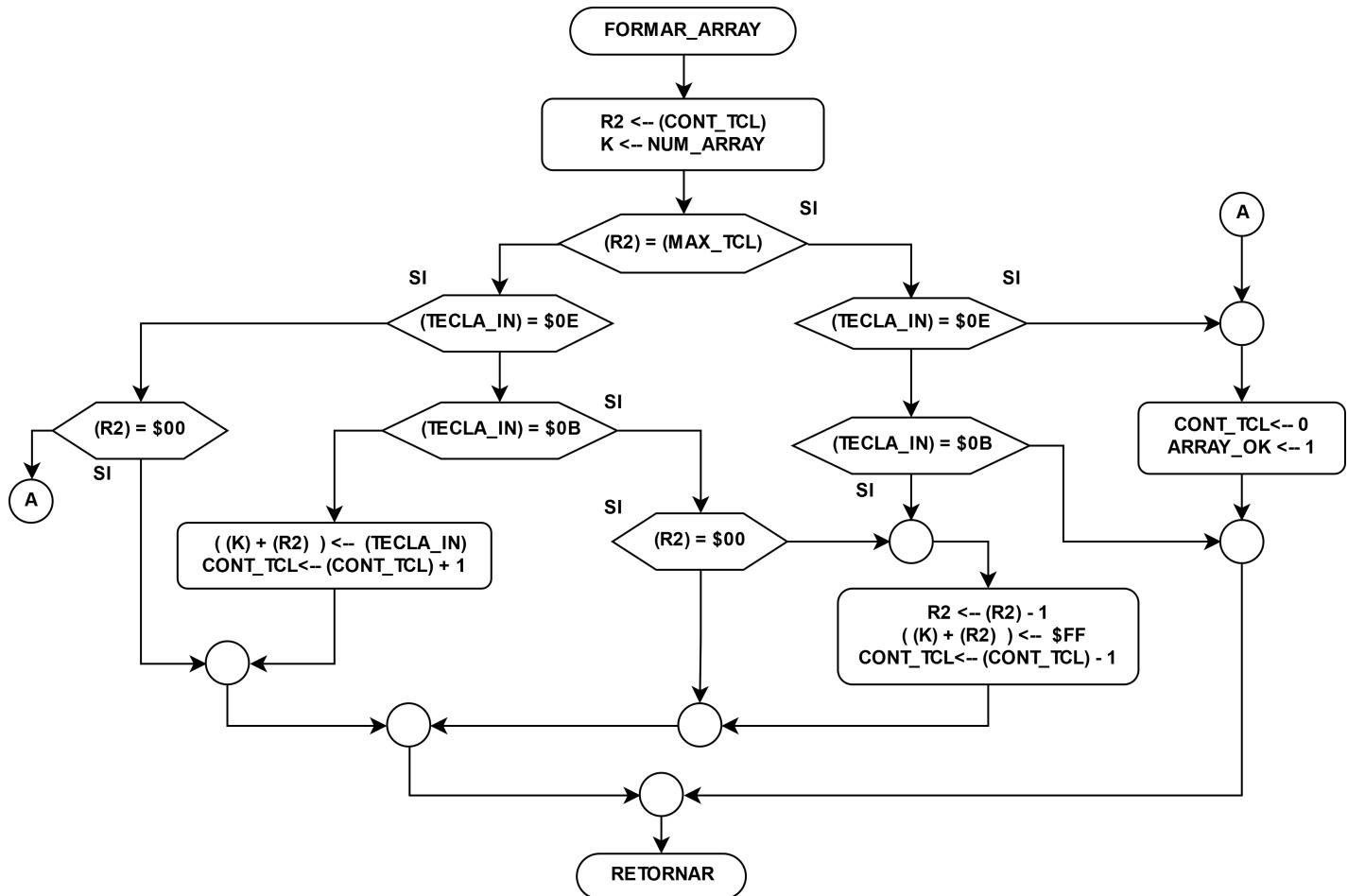


Figura 7: Subrutina FORMAR_ARRAY

2.8. Subrutina LIBRE

En la figura [8] se presenta el diagrama de flujos de esta subrutina.

2.8.1. Descripcion:

Esta subrutina se encarga de inicializar las variables y la LCD en el modo libre.

2.8.2. Parámetros de entrada:

- MOD_LIB_ACTUAL: Bandera recibida por memoria.

2.8.3. Parámetros de salida:

- MOD_CONF_ACTUAL: Bandera transmitida por memoria.
- MOD_MED_ACTUAL: Bandera transmitida por memoria.
- LEDS: Variable transmitida por memoria.

- BIN1: Variable transmitida por memoria
- BIN2: Variable transmitida por memoria

2.8.4. Explicación detallada:

Esta subrutina revisa la bandera MOD_LIB_ACTUAL, si es 1 significa que se viene de un cambio de modo por lo que se actualizan las demás banderas de cambio de modo. Seguidamente se envía el mensaje modo libre a la LCD, se actualizaz el LED de este modo, se desactivan las interrupciones de key wake ups y TO y por último se apagan los display de 7 segmentos. Si MOD_LIB_ACTUAL es 0 la subrutina no hace nada.

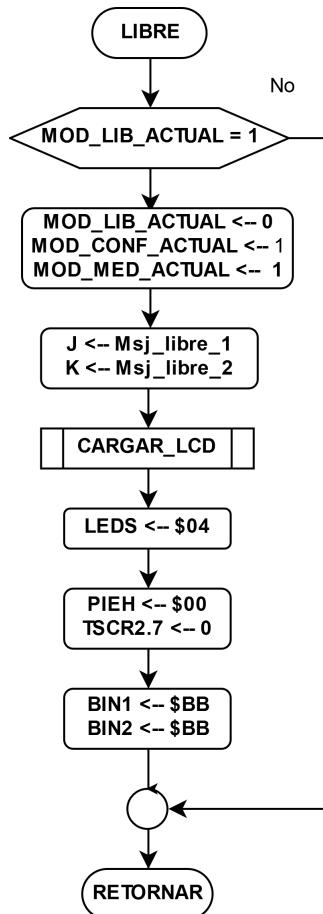


Figura 8: Subrutina FORMAR_ARRAY

2.9. Subrutina MODO_MEDICION

En la figura [9] se presenta el diagrama de flujos de esta subrutina.

2.9.1. Descripción:

Esta subrutina se encarga de controlar el modo medición llamando a PANT_CTRL cuando sea necesario.

2.9.2. Parámetros de entrada:

- VELOC. Parámetro tipo byte pasado por memoria.

2.9.3. Parámetros de salida:

- BIN1. Parámetro tipo byte retorna por memoria.
- BIN2. Parámetro tipo byte retorna por memoria.

2.9.4. Explicación detallada:

Esta subrutina verifica si la velocidad actual es 0, si es así simplemente apaga los display de 7 segmentos porque significa que ningún carro ha pasado por el sensor ph3. En caso de que velocidad sea distinta de 0 significa que un carro vino de ph3 y pasó ph0, por lo que se debe llamar a la subrutina que controla la pantalla llamada PANT_CTRL.

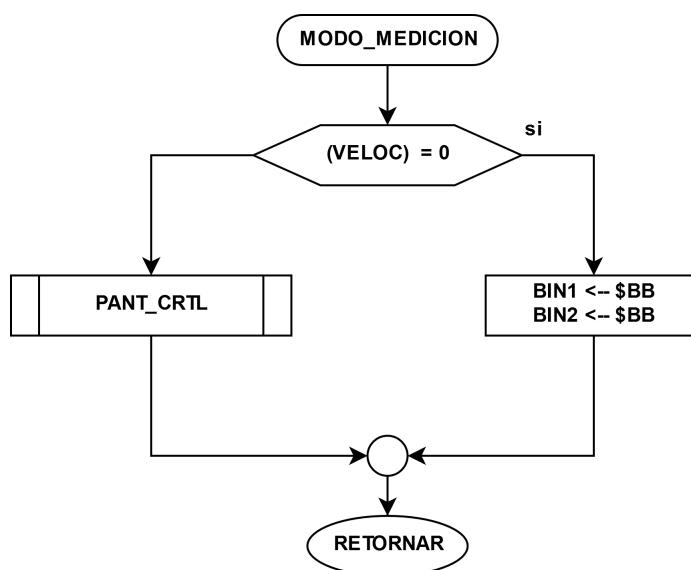


Figura 9: Subrutina MODO_MEDICION

2.10. Subrutina PANT_CTRL

En la figura [10] se presenta el diagrama de flujos de esta subrutina.

2.10.1. Descripción:

Esta subrutina se encarga de realizar todo el control de las pantallas cuando se presiona el botón PH0 después de haber presionado PH3.

2.10.2. Parámetros de entrada:

- VELOC: Parámetro ingresado por memoria.
- V_LIM: Parámetro ingresado por memoria.

- PANT_FLAG: Bandera ingresada por memoria.

2.10.3. Parámetros de salida:

- ALERTA: Bandera transmitida por memoria.
- TICK_EN: WORD transmitido por memoria.
- TICK_DIS: WORD transmitido por memoria.
- BIN1: Byte transmitido por memoria.
- BIN2: Byte transmitido por memoria.
- PH3_EN: Bandera transmitida por memoria.
- VELOC: Byte transmitido por memoria.

2.10.4. Explicación detallada:

Esta subrutina lo primero que hace es desactivar las interrupciones de los botones PH3 y PH0, seguidamente verifica que la velocidad almacenada en VELOC esté dentro del rango válido, si no está dentro del rango válido se pregunta si la velocidad es igual a \$AA, si no es así se carga TICK_DIS con el valor 91, que provoca que la pantalla se apague hasta dentro de 2 segundos (el cálculo de este valor se mostrará más adelante). Ahora en la siguiente iteración al preguntarse por PANT_FLH lo encuentra en 1 con lo que enciende la pantalla poniendo unos guiones en la misma, dentro de 2 segundos PANT_FLH será 0 por lo que la pantalla de 7 segmentos se apagará y reiniciará los valores a como estaban al inicio de entrar a la subrutina la primera vez.

Si la velocidad sí es válida, entonces se verifica si la velocidad es mayor que la velocidad máxima, de ser así se enciende el bit de ALERTA para que los leds se recorran en manera de alerta. La primer iteración se encuentra CALC_TICKS en 0 por lo que se cargan los valres de TICK_EN y TICK_DIS para que la pantalla se encienda cuando el carro esté a 100 metros de la pantalla y se apague cuando la pase. Este cálculo se mostrará más adelante en una memoria de cálculo. Finalmente se pone CALC_TICKS en 1 para que no se vuelvan a cargar los valores de TICK_EN y TICK_DIS.

Una vez que se cargaron dichos valores, la subrutina no hará nada hasta que PANT_FLH se ponga en 1, en este momento se pondrá en los display de 7 segmentos la velocidad a la que va el carro y la velocidad máxima permitida. Cuando PANT_FLH se ponga en 0 va a ser porque ya el carro pasó la pantalla y se apagarán los displays y se reiniciarán todos los parámetros necesarios y la LCD para esperar que vuelva a pasar otro auto.

2.10.5. Memoria de cálculo

El cálculo de TICK_DIS para 2 segundos se da teniendo en cuenta que cada tick en tiempo es de aproximadamente

$$2 = \left(\frac{\text{Preescalador}}{\text{Bus}_\text{clk}} \right) * 65535 * \text{TICKS_DIS}$$

$$\text{TICKS_DIS} = \frac{2}{\left(\frac{8}{24M} \right) * 65535} = 91$$

Por lo que el valor a cargar en TICK_DIS es 91.

Con respecto al cálculo requerido para que el conductor esté a 100 metros de distancia, tenemos la siguiente ecuación:

$$0,1 = \frac{VELOC}{3600} * TICK_EN * 0,021845$$

$$TICK_EN = \frac{16479}{VELOC}$$

y el cálculo de TICKS_DIS es simplemente multiplicar el lado izquierdo de la ecuación por dos

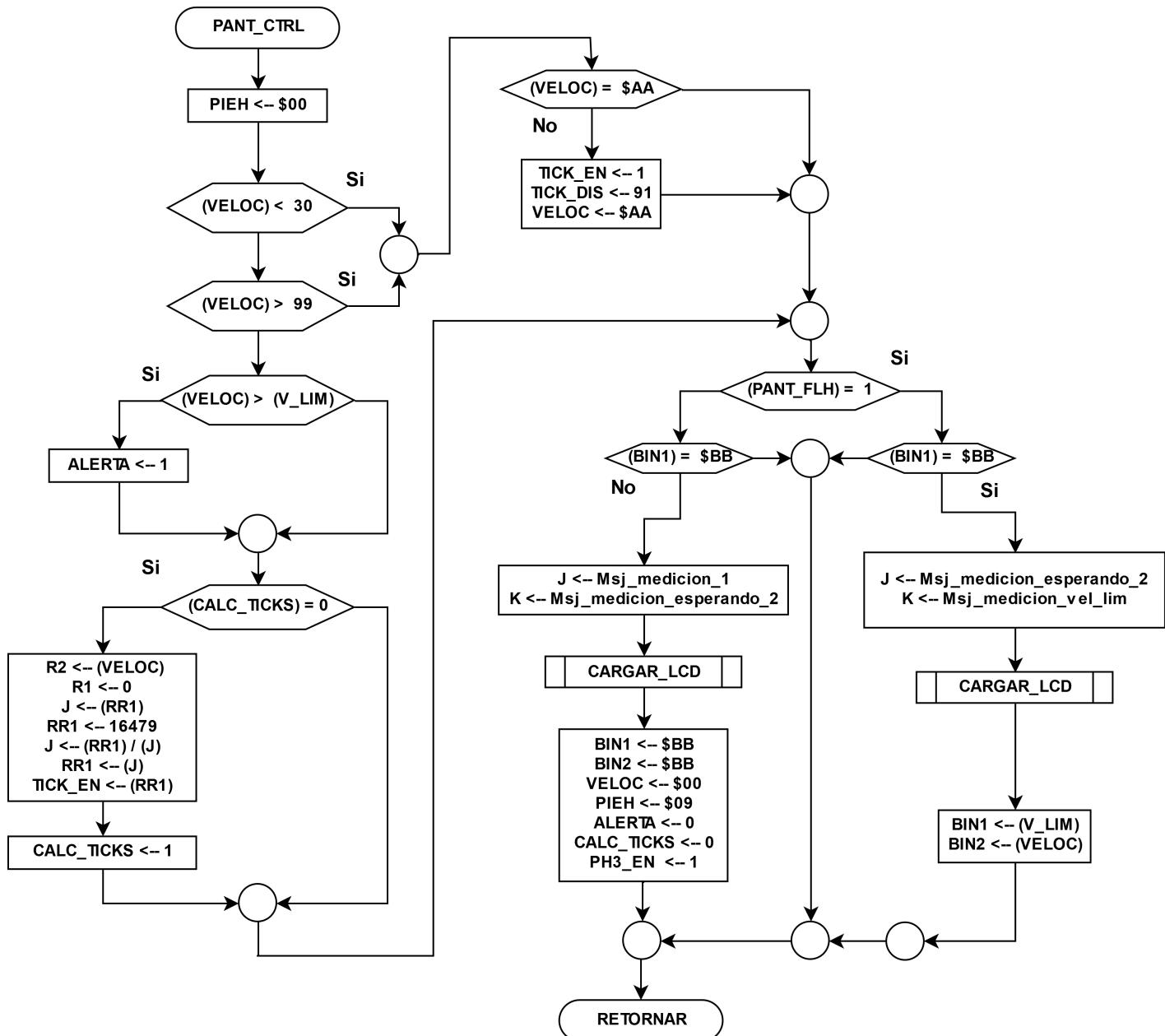


Figura 10: Subrutina PANT_CTRL

2.11. Subrutina CONV_BIN_BCD

En la figura [11] se presenta el diagrama de flujos de esta subrutina.

2.11.1. Descripción:

Esta subrutina se encarga de hacer la conversión total de Binario a BCD, llamando la subrutina BIN_BCD.

2.11.2. Parámetros de entrada:

- BIN1: Parámetro pasado por memoria.
- BIN2: Parámetro pasado por memoria.

2.11.3. Parámetros de salida:

- BCD1: Parámetro retorna por memoria.
- BCD2: Parámetro retorna por memoria.

2.11.4. Explicación detallada:

Esta subrutina toma los valores de BIN1 y BIN2, si dichos valores son \$AA o \$BB, se guardan directamente en BCD1 y BCD2 respectivamente. En caso de que tengan otro valor se llamará a la subrutina BIN_BCD para calcular el valor en BCD y almacenarlo en BCD1 y BCD2. En caso de que se presente un valor de BCD con un 0 a la izquierda, se copiará una \$B en la izquierda para que solo se encienda uno de los display.

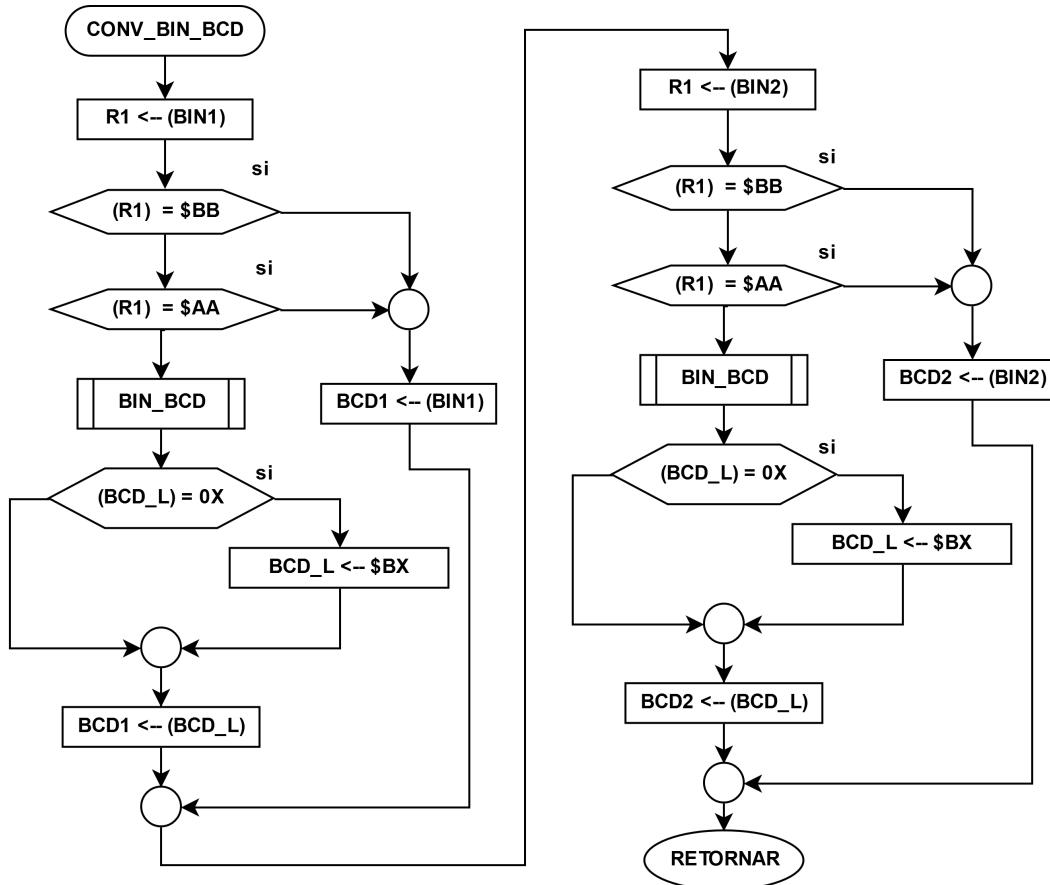


Figura 11: Subrutina CONV_BIN_BCD

2.12. Subrutina BIN_BCD

En la figura [12] se presenta el diagrama de flujos de esta subrutina.

2.12.1. Descripción:

Esta subrutina recibe un parámetro por el acumulador A, dicho número en binario es convertido a BCD y almacenado en BCD_L.

2.12.2. Parámetros de entrada:

- Acumulador A: recibe número en binario a ser convertido.

2.12.3. Parámetros de salida:

- BCD_L: Este parámetro es pasado por memoria, contiene el número resultado en BCD

2.12.4. Explicación detallada:

Esta subrutina toma el valor del acumulador A y mediante el uso de la pila siguiendo el algoritmo de conversión de binario a BCD, almacena el valor del número transformado en la variable BCD_L. Se

utiliza BCD_H como parámetro auxiliar.

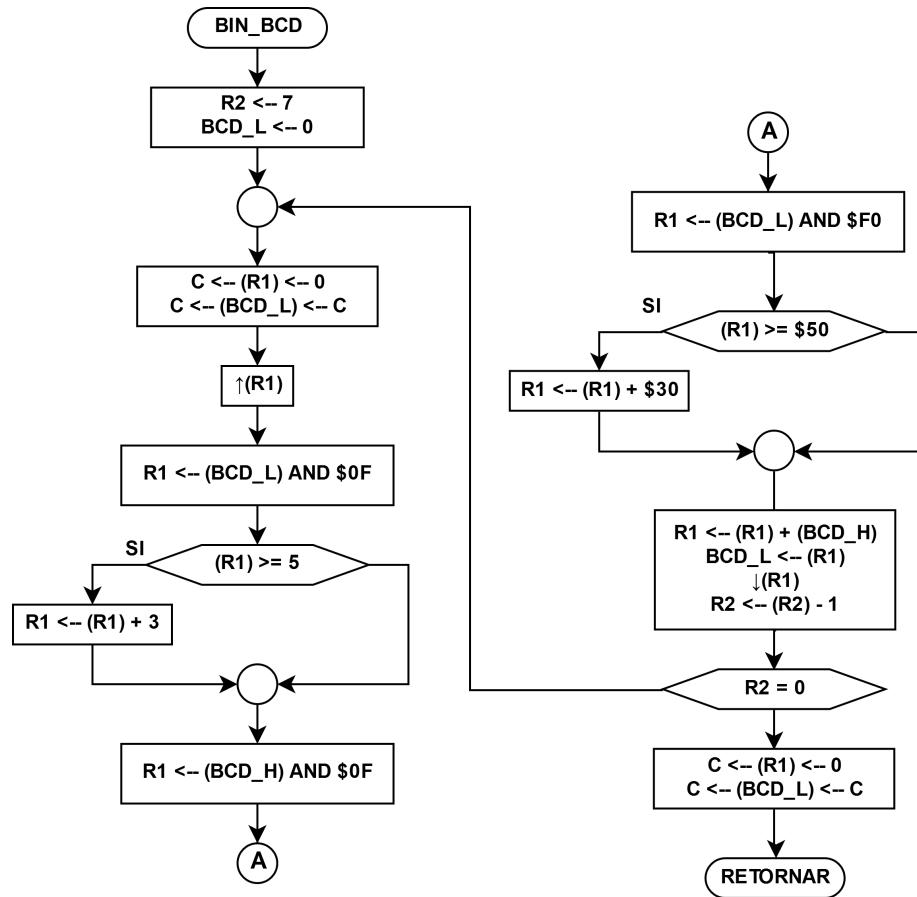


Figura 12: Subrutina BIN_BCD

2.13. Subrutina BCD_7SEG

En la figura [13] se presenta el diagrama de flujos de esta subrutina.

2.13.1. Descripcion:

Esta subrutina se encarga de tomar los valores de BCD1 y BCD2 y transformarlos en código de 7 segmentos para almacenarlos en DISP1, DISP2, DISP3, DISP4.

2.13.2. Parámetros de entrada:

- BCD1: Parámetro ingresado por memoria.
- BCD2: Parámetro ingresado por memoria.

2.13.3. Parámetros de salida:

- DISP1: Parámetro devuelto por memoria.

- DISP2: Parámetro devuelto por memoria.
- DISP3: Parámetro devuelto por memoria.
- DISP4: Parámetro devuelto por memoria.

2.13.4. Explicación detallada:

Esta subrutina se encarga de tomar el valor contenido en la variable BCD1, que contiene en los primeros 4 bits (comenzando de LSB) el primer número en BCD y en los últimos dos el segundo número. Estos números se codifican en 7 segmentos y son pasados a las variables de salida DISP1 y DISP2. Lo mismo ocurre con BCD2 pero almacena su resultado en DISP3 y DISP4.

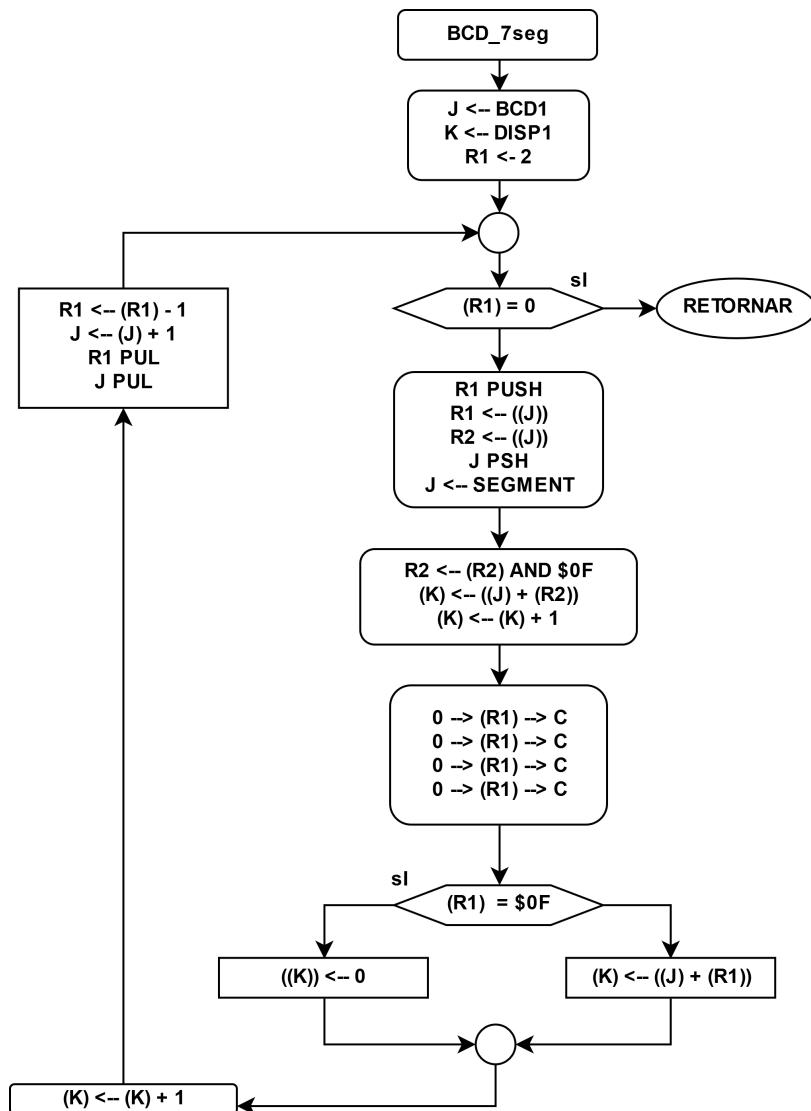


Figura 13: Subrutina BCD_7SEG

2.14. Subrutina PATRON_LEDS

En la figura [14] se presenta el diagrama de flujos de esta subrutina.

2.14.1. Descripción:

Esta subrutina controla el barrido de emergencia de los leds a partir de una bandera.

2.14.2. Parámetros de entrada:

- ALERTA: Bandera pasada por memoria.

2.14.3. Parámetros de salida:

- LEDS: Variable tipo byte devuelto por memoria.

2.14.4. Explicación detallada:

Esta subrutina no hará nada nunca a menos que el bit ALERTA esté en 1, en este momento cargará el bit más significativo de LEDS y retornará. En las siguientes iteraciones desplazará el led encendido hasta llegar a pb3, en este momento volverá a encender pb7 y apagará pb3. Esto se repite hasta que ALERTA sea 0, en este momento solo se dejarán encendidos los leds de modo.

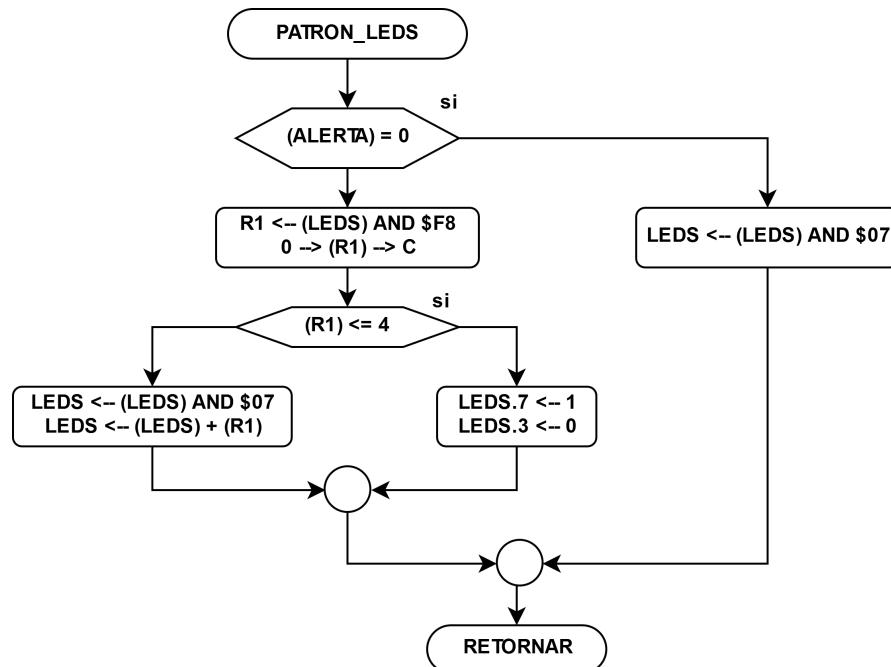


Figura 14: Subrutina PATRON_LEDS

2.15. Subrutina LCD

En la figura [15] se presenta el diagrama de flujos de esta subrutina.

2.15.1. Descripción:

Esta subrutina se encarga de inicializar la pantalla LCD con el fin de cargar mensajes en ella posteriormente.

2.15.2. Parámetros de entrada:

- IniDsp: Tabla tipo bye con comandos de inicialización. Pasada por memoria.

2.15.3. Parámetros de salida:

- No tiene

2.15.4. Explicación detallada:

Esta subrutina carga los valores de inicialización de la tabla iniDsp y manda uno a uno a la pantalla LCD, para esto utiliza la subrutina SEND mandando los distintos comandos. También se utiliza la subrutina DELAY con el fin de respetar la temporización. Los comandos específicamente configuran que el display esté encendido, el cursor apagado y sin parpadear. Se utiliza \$FF como símbolo para saber que se terminó el string. La etiqueta EOM es la que tiene este valor. Esta subrutina solo se ejecuta una vez, debido a que es de configuración.

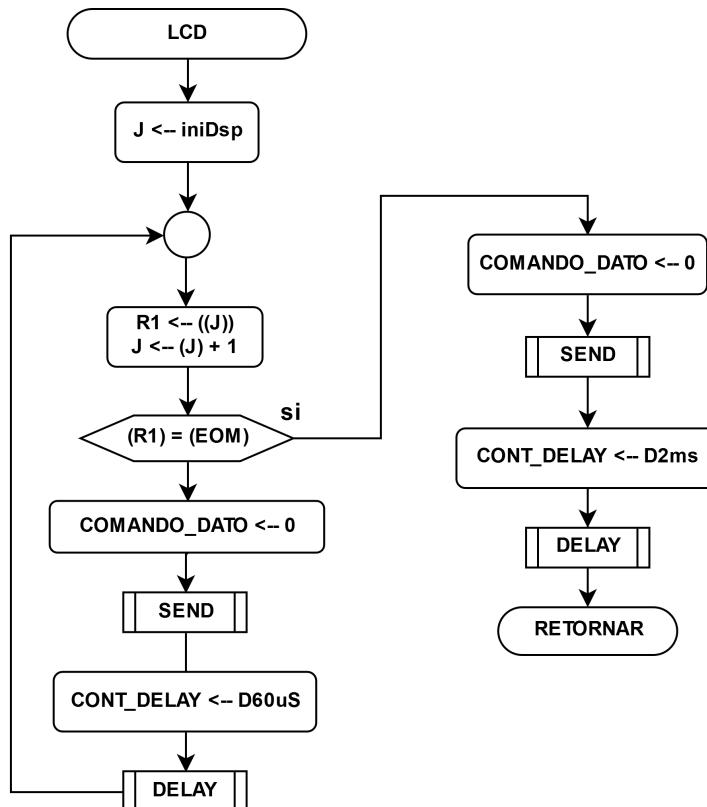


Figura 15: Subrutina LCD

2.16. Subrutina CARGAR_LCD

En la figura [16] se presenta el diagrama de flujos de esta subrutina.

2.16.1. Descripción:

Esta subrutina se encarga de cargar un mensaje en la LCD, recibe los mensajes como parámetros de entrada en los punteros X y Y.

2.16.2. Parámetros de entrada:

- Registro índice X: Recibe la dirección de inicio del mensaje que irá en la parte superior de la LCD.
- Registro índice Y: Recibe la dirección de inicio del mensaje que irá en la parte inferior de la LCD.

2.16.3. Parámetros de salida:

- No tiene

2.16.4. Explicación detallada:

Esta subrutina toma la dirección de los mensajes que se pasaron como parámetros de entrada por los registros índices X y Y. Caracter por carácter esta subrutina envía los datos del mensaje que irá en la parte superior de la LCD, y se detiene hasta que encuentra EOM. Seguidamente se envía byte por byte el mensaje que irá en la parte inferior de la LCD hasta que se vuelve a encontrar un EOM. En este momento la subrutina retorna.

Es importante recordar que esta subrutina depende de la interrupción OC4 por lo que se debe tener disponible. Si esta subrutina se usa en una interrupción, antes de llamarla se deben activar las interrupciones mascarables.

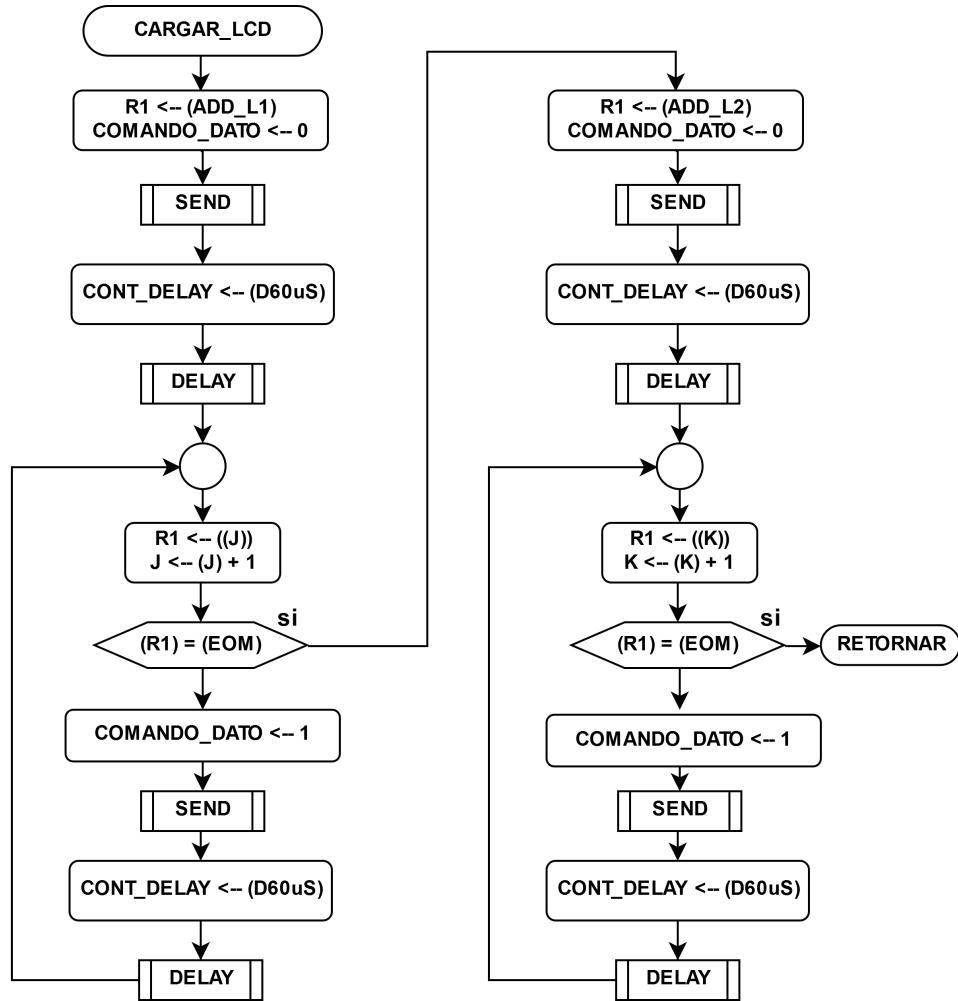


Figura 16: Subrutina CARGAR_LCD

2.17. Subrutina SEND

En la figura [17] se presenta el diagrama de flujos de esta subrutina.

2.17.1. Descripcion:

Esta subrutina tiene dos funciones, manda comandos o datos a la pantalla LCD. Si la bandera COMANDO_DATO es 0 se manda un comando, si es 1 se manda un dato.

2.17.2. Parámetros de entrada:

- COMANDO_DATO: Bandera que se pasa por memoria e indica con un 0 que se va a mandar un comando, con un 1 se va a mandar un dato.
 - Acumulador A: Se pasa el byte que será enviado a la pantalla LCD.

2.17.3. Parámetros de salida:

- No tiene

2.17.4. Explicación detallada:

Esta subrutina controla la temporización y el envío de un byte a la pantalla LCD. Debido a que un byte se debe enviar en nibbles la subrutina realiza este proceso dos veces. La bandera COMANDO_DATO controla el envío de un dato o un comando.

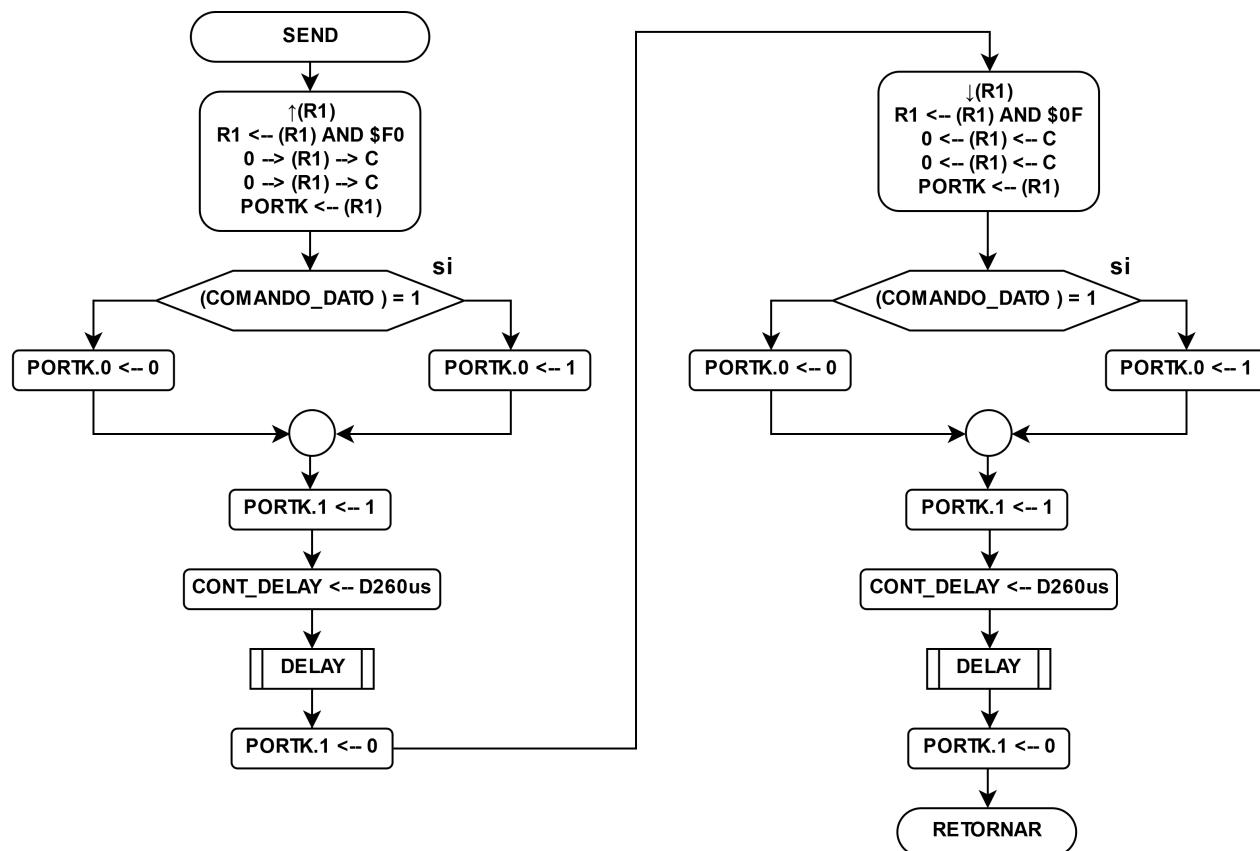


Figura 17: Subrutina SEND

2.18. Subrutina DELAY

En la figura [18] se presenta el diagrama de flujos de esta subrutina.

2.18.1. Descripción:

Se encarga de generar delays con la ayuda de la subrutina RTI.

2.18.2. Parámetros de entrada:

- CONT_DELAY: Variable tipo byte pasada por memoria.

2.18.3. Parámetros de salida:

- No tiene.

2.18.4. Explicación detallada:

Esta subrutina simplemente se encarga de esperar a que CONT_DELAY sea 0. CONT_DELAY es decrementada una vez cada 1ms por la interrupción RTI, por lo que con esto se realizan delays de distinto tamaño de tiempo, cargando dicha variable.

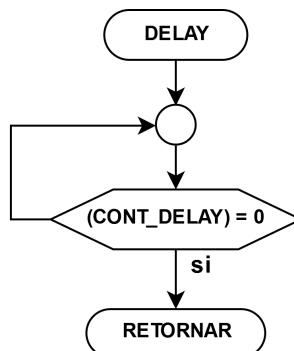


Figura 18: Subrutina DELAY

2.19. Subrutina de atención de interrupciones ATD_ISR

En la figura [19] se presenta el diagrama de flujos de esta subrutina.

2.19.1. Descripción:

Esta subrutina se encarga de tomar 6 valores leídos por el ATD y calcular un promedio que será guardado en la variable BRILLO.

2.19.2. Parámetros de entrada:

- ADRO1H: Registro de datos del periférico, tamaño de 16 bits.
- ADRO2H: Registro de datos del periférico, tamaño de 16 bits.
- ADRO3H: Registro de datos del periférico, tamaño de 16 bits.
- ADRO4H: Registro de datos del periférico, tamaño de 16 bits.
- ADRO5H: Registro de datos del periférico, tamaño de 16 bits.
- ADRO6H: Registro de datos del periférico, tamaño de 16 bits.

2.19.3. Parámetros de salida:

- POT: Parámetro de salida por memoria.
- BRILLO: Parámetro de salida por memoria.

2.19.4. Explicación detallada:

Esta subrutina toma los 6 valores de 8 bits de los registros de datos del ATD y los divide entre 6, con el fin de calcular el promedio. Dicho promedio se guarda en la variable POT. Seguidamente se procede a realizar la siguiente ecuación:

$$BRILLO = \frac{POT * 20}{256} * 5$$

Esta ecuación transforma un valor de 0 a 255 en un valor en escala de 0 a 100, en pasos de 5 en 5. Esto debido a que BRILLO se comporta de esta forma. La variable BRILLO se almacena con este valor.

2.19.5. Memoria de cálculo:

Se necesita que la frecuencia del ATD sea de 500KHz, por lo que se necesita un preescalador con el siguiente valor:

$$500KHz = \frac{Bus_clk}{2 * (Preescalador + 1)}$$

$$Preescalador = \frac{24M}{500KHz * 2} - 1 = 23$$

Esta fórmula fue tomada del material del curso. [1]

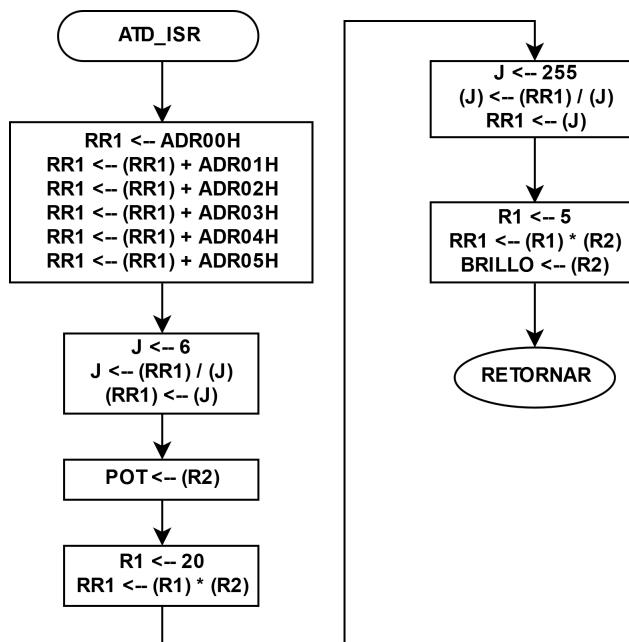


Figura 19: Subrutina de atención de interrupción ATD_ISR

2.20. Subrutina de atención de interrupciones CALCULAR

En la figura [20] se presenta el diagrama de flujos de esta subrutina. Se harán dos descripciones debido a que esta subrutina de atención de interrupción tiene dos funciones distintas dependiendo de qué botón se presionó.

2.20.1. Descripción PH3:

Esta subrutina se encarga de inicializar el paso de un carro por el primer sensor, y habilitar el funcionamiento del botón ph0.

2.20.2. Parámetros de entrada PH3:

- CONT_REB: Parámetro recibido por memoria.
- PH3_EN: Bit recibido por memoria

2.20.3. Parámetros de salida PH3:

- PH0_EN: Bit transmitido por memoria.
- TICK_VEL: Byte transmitido por memoria

2.20.4. Explicación detallada PH3:

Esta subrutina antes de empezar su funcionamiento verifica que CONT_REB sea 0, esto es para evitar pulsaciones por los rebotes mecánicos del botón. Seguidamente se revisa la bandera PH3_EN que es un habilitador para que este botón solo se pulse una vez por ciclo (donde un ciclo es todo el proceso mientras el carro toca el primer sensor y pasa la pantalla). Seguidamente se habilita el botón PH0 poniendo la bandera PH0_EN en 1 y se procede a cargar CONT_REB mientras que se borra TICK_VEL. Por último se borra la bandera de la interrupción y se activan las interrupciones mascarables con el fin de llamar la subrutina CARGAR_LCD. Dichas subrutina utiliza OC4 para hacer los delay, es por esto que se activan las interrupciones mascarables.

2.20.5. Descripción PH0:

Esta subrutina se encarga de realizar el cálculo de la velocidad del auto y almacenarla en VELOC.

2.20.6. Parámetros de entrada PH0:

- CONT_REB: Parámetro recibido por memoria.
- PH0_EN: Bit recibido por memoria
- TICK_VEL: Byte recibido por memoria

2.20.7. Parámetros de salida PH0:

- PH3_EN: Bit transmitido por memoria.
- VELOC: Byte transmitido por memoria.

2.20.8. Explicación detallada PH0:

Esta subrutina primero verifica que no sea un rebote la pulsación que se está leyendo en este momento, para esto revisa el byte CONT_REB. Seguidamente revisa que el botón esté activo leyendo el bit PH0_EN, de ser así se desactiva el botón PH0 y se carga el contador de rebotes. Seguidamente se realiza el cálculo de la velocidad a partir de la variable TICK_VEL, en la memoria de cálculo se especificará cómo se da este cálculo, se borra TICK_VEL y se verifica que la velocidad calculada no fuese mayor de 255, de ser así pone VELOC en \$FF si la velocidad era menor a 255 se guarda en VELOC y se retorna.

2.20.9. Memoria cálculo PH0:

Para el cálculo de la velocidad primero tenemos que:

$$\frac{1}{vel_por_tick} = \frac{1}{0,021845} = 46$$

Ya que tenemos esta aproximación, procedemos al cálculo de la velocidad de la siguiente forma:

$$\frac{Dist_entre_sensores}{vel_por_tick * TICK_VEL} * 3600 = \frac{46 * 3600 * 0,04}{TICK_VEL} = \frac{6624}{TICK_VEL}.$$

Por lo que de esta forma se calcula la velocidad.

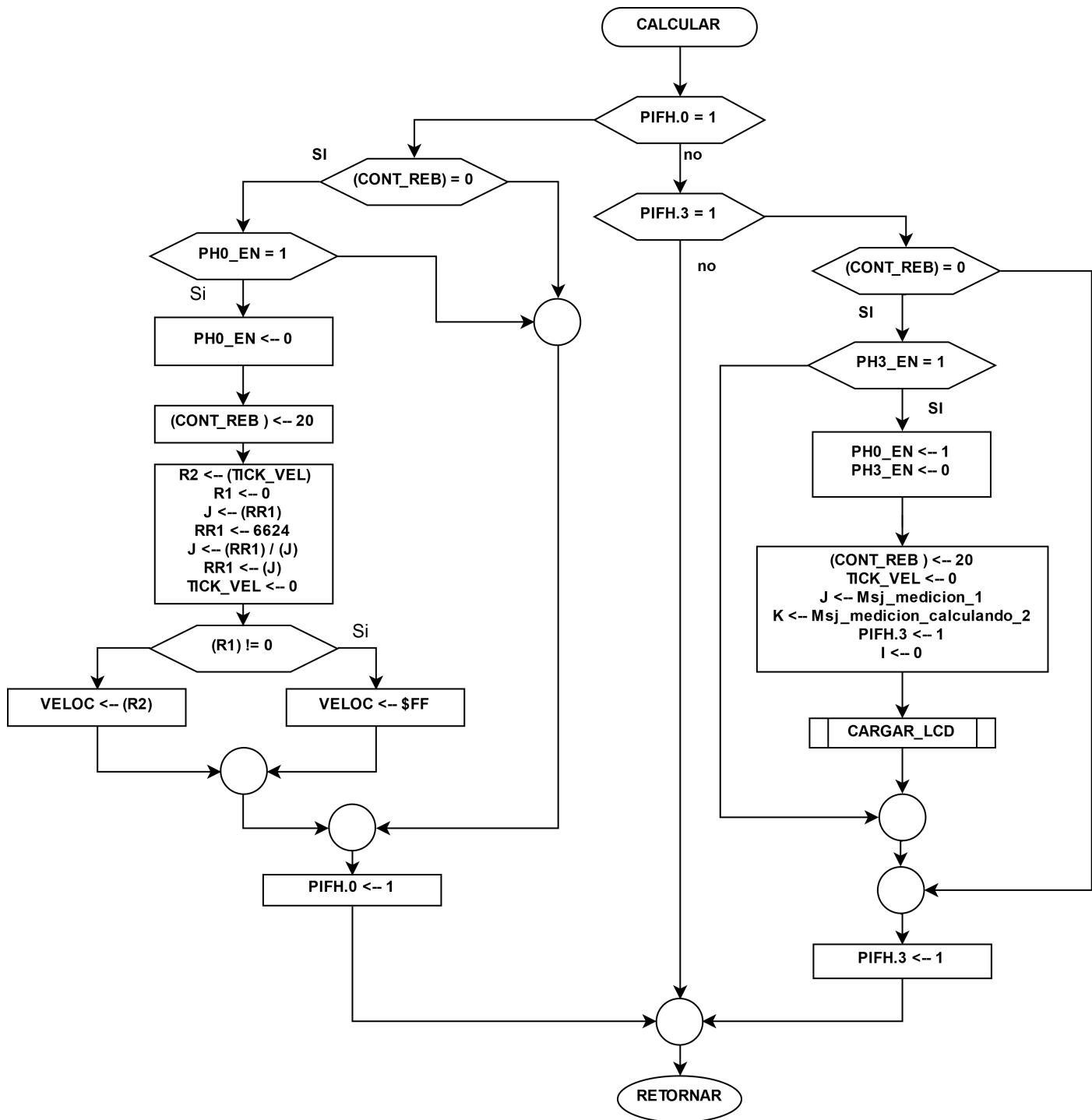


Figura 20: Subrutina de atención de interrupción CALCULAR

2.21. Subrutina de atención de interrupciones TCNT_ISR

En la figura [21] se presenta el diagrama de flujos de esta subrutina.

2.21.1. Descripción:

Esta subrutina se encarga de contar para calcular la velocidad y para apagar o encender pantallas.

2.21.2. Parámetros de entrada:

- PH0: Bandera pasada por memoria.
- TICK_EN: Word pasado por memoria.
- TICK_DIS: Word pasado por memoria.

2.21.3. Parámetros de salida:

- PANT_FLAG: Bandera transmitida por memoria.

2.21.4. Explicación detallada:

Esta subrutina lo primero que hace es preguntar por la bandera que activa PH0 la cuál es PH0_EN, esta bandera solo se levanta cuando se ha presionado PH3 por lo que es el momento en que se debe contar TICK_VEL, con cada interrupción esta variable se aumenta en una unidad. Dicha variable cuando se va a rebasar se mantiene en 255, ya que esto producirá un valor inválido de todas formas, y si este contador se rebasara podría generar errores.

La otra función de esta interrupción es decrementar los word TICK_EN y TICK_DIS. Cuando TICK_EN es 0 PANT_FLAG se pone en 1, cuando TICK_DIS es 0 PANT_FLAG se pone en 0.

2.21.5. Memoria de cálculo:

Para calcular el tiempo entre cada interrupción del TCNT tenemos la siguiente fórmula:

$$Tiempo_{entre_interrup} = 65535 * \frac{preescalador}{bus_clk} = 0,02184s$$

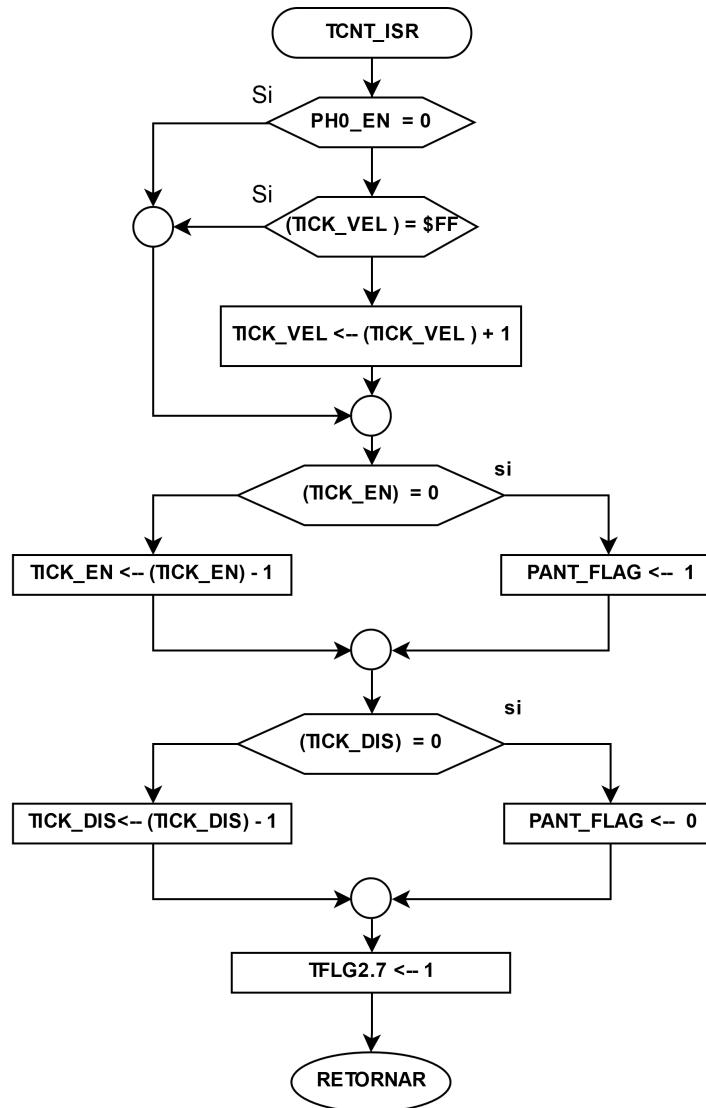


Figura 21: Subrutina de atención de interrupción TCNT_ISR

2.22. Subrutina de atención de interrupciones RTI_ISR

En la figura [22] se presenta el diagrama de flujos de esta subrutina.

2.22.1. Descripción:

Esta interrupción simplemente se encarga de decrementar CONT_REB para controlar los rebotes de los botones mecánicos.

2.22.2. Parámetros de entrada:

- CONT_REB: Parametro pasado por memoria, tipo byte

2.22.3. Parámetros de salida:

- CONT_REB: Parametro pasado por memoria, tipo byte

2.22.4. Explicación detallada:

Esta subrutina simplemente se encarga de decrementar el contador de rebotes si no es 0, con el fin de realizar el control de rebotes de todos los botones mecánicos de la tarjeta. Entre estos botones están los del teclado, que solo funcionan en modo configuración y también los de ph0 y ph3, que solo funcionan en modo medición. Debido a que los modos son excluyentes es posible utilizar solo una variable para contar los rebotes sin que se presenten problemas.

2.22.5. Memoria de cálculo:

Para calcular el tiempo entre cada interrupción de RTI tenemos la siguiente fórmula:

$$\text{Tiempo_entre_interrup} = \frac{(N + 1) * 2^{M+9}}{\text{OSC_CLK}} = \frac{(3 + 1) * 2^{2+9}}{8M} = 1,024\text{ms}.$$

Esta fórmula fue tomada del material del curso. [1]

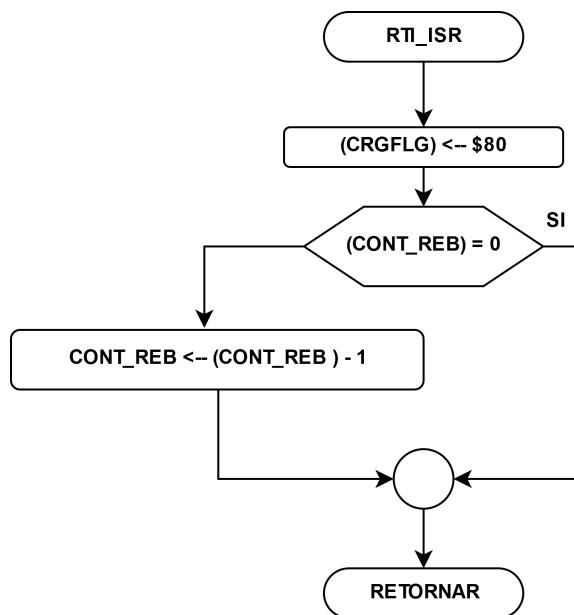


Figura 22: Subrutina de atención de interrupción RTI_ISR

2.23. Subrutina de atención de interrupciones OC4_ISR

En la figura [23] se presenta el diagrama de flujos de esta subrutina.

2.23.1. Descripción:

Esta subrutina se encarga de organizar la multiplexación de las pantallas, iniciar la conversión analógica a digital cada 200ms y de enviar datos nuevamente a los display cada 100ms.

2.23.2. Parámetros de entrada:

- BRILLO: Variable recibida por memoria.
- PORTB: Variable recibida por memoria.

2.23.3. Parámetros de salida:

- CONT_DELAY: Variable transmitida por memoria.

2.23.4. Explicación detallada:

Esta subrutina en primer lugar incrementa la variable CONT_7SEG hasta que llega a 5000, cuando llega a este valor significa que han pasado 100ms por lo que es momento de poner CONT_7SEG en 0, llamar a la subrutina CONV_BIN_BCD y BCD_7SEG para actualizar los valores a ser desplegados en la pantalla.

Seguidamente esta subrutina realiza toda la multiplexación de cada uno de los displays de 7 segmentos y los LEDS. Para esto utiliza la variable BRILLO con el fin de saber cuánto tiempo dejar encendido cada uno de los display, de esta forma se puede ajustar el brillo de alto a nulo de la pantalla.

También esta subrutina se encarga de decrementar CONT_DELAY que es utilizada para generar los delays respectivos en la pantalla de LCD. Por último se utiliza la variable CONT_200 para contar hasta 10000, cuando se llega a este valor significa que han pasado 200ms y es momento de iniciar un ciclo de conversión analógico a digital escribiendo el registro ATD0_CTL5, además se pone en 0 CONT_200 y se llama a la subrutina PATRON_LEDS con el fin de realizar o no el barrido de los LEDS de emergencia.

Lo último que se hace en esta subrutina es tomar TCNT sumarle 60 y guardarla en TC4 con el fin de que se genere la siguiente interrupción en 20us. Esto se realizó al final de la interrupción debido a que tiene una longitud considerablemente alta.

2.23.5. Memoria de cálculo:

La interrupción de OC4 necesita un periodo de interrupción de aproximadamente 20us. Para calcular el valor a cargar de TC4 se utiliza la siguiente fórmula:

$$20\mu = \frac{TC4 * \text{preescalador}}{\text{BUS_CLK} *}$$

Con un preescalador de 8 y un BUS_CLK de 24M tenemos que TC4 es:

$$TC4 = 60$$

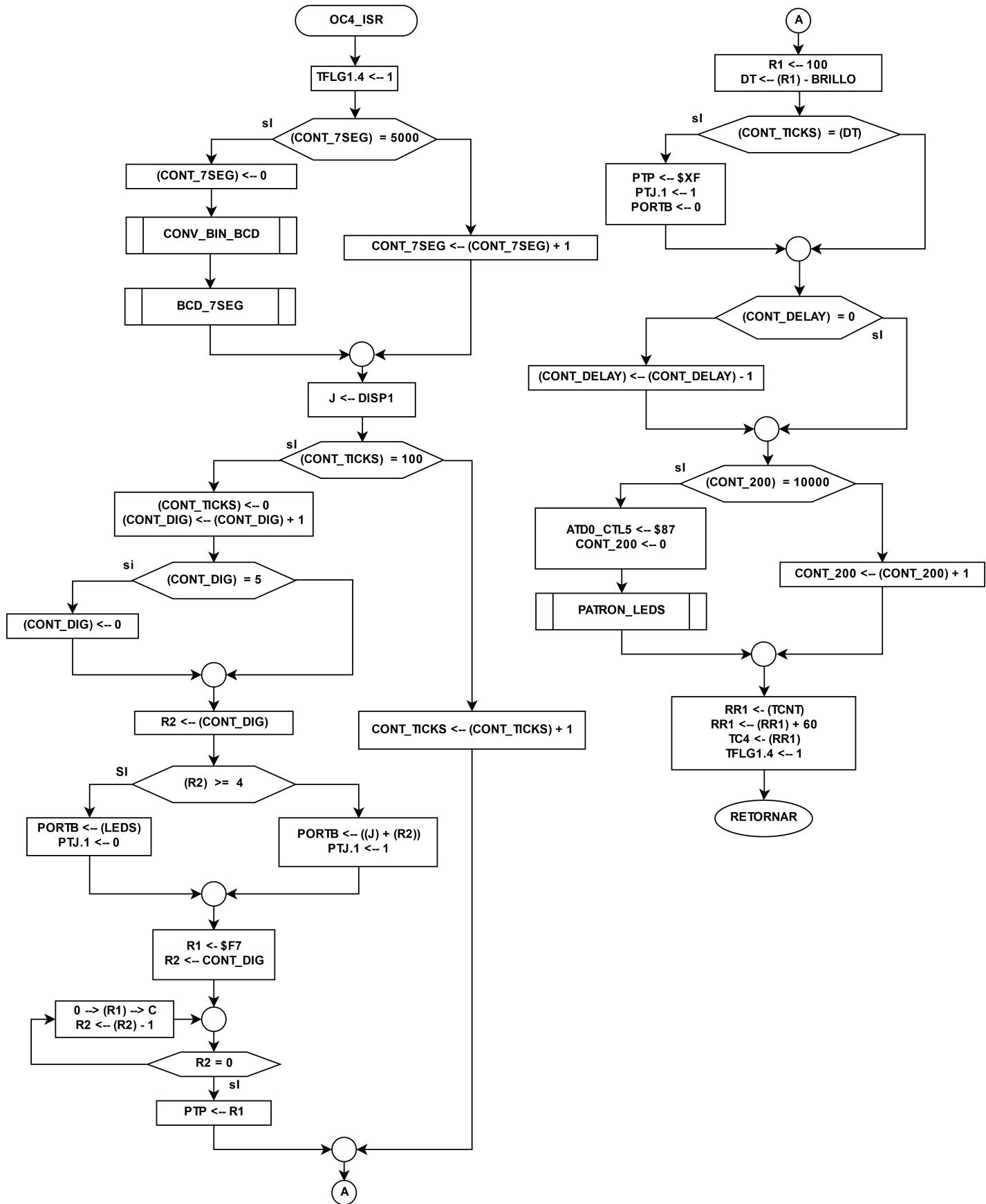


Figura 23: Subrutina de atención de interrupción OC4_ISR

3. Protocolo de pruebas realizado.

En esta sección se presenta el protocolo de pruebas realizado para este proyecto, con el fin de asegurar el correcto funcionamiento en distintos escenarios del mismo.

3.1. Prueba 1.

La primer prueba consiste en lo primero que se debe probar, y es el correcto funcionamiento general de la máquina calculando la velocidad del automóvil y el tiempo en que se debe encender la pantalla. Veamos la figura 24, en ella se muestra el tiempo entre pulsaciones de los botones PH0 y PH3 y la respectiva velocidad que debería marcar el producto. También se muestra el tiempo que debe tardar la pantalla en mostrar dicha velocidad y luego quitarla. En este prueba se hizo lo posible por medir todos los valores de esta tabla y el producto se comporta igual que lo esperado.

Tiempo (S)	Velocidad del auto Km/H	Tiempo de pantalla (S)
1,5	96	3,75
1,6	90	4
1,7	85	4,25
1,8	80	4,5
1,9	76	4,75
2	72	5
2,1	69	5,25
2,2	65	5,5
2,3	63	5,75
2,4	60	6
2,5	58	6,25
2,6	55	6,5
2,7	53	6,75
2,8	51	7
2,9	50	7,25
3	48	7,5
3,1	46	7,75
3,2	45	8
3,3	44	8,25
3,4	42	8,5
3,5	41	8,75
3,6	40	9
3,7	39	9,25
3,8	38	9,5
3,9	37	9,75
4	36	10
4,1	35	10,25
4,2	34	10,5
4,3	33	10,75
4,4	33	11
4,5	32	11,25
4,6	31	11,5
4,7	31	11,75
4,8	30	12
4,9	29	12,25

Figura 24: Tabla de tiempos esperados para el producto.

3.2. Prueba 2.

Se debe iniciar el sistema y que solo ingrese en modo configuración hasta que se asigne una velocidad máxima para los vehículos. En caso de ingresar una velocidad erronea la velocidad volverá nuevamente a ser 0, y debe configurarse un valor entre 45 y 90 para que se pueda utilizar en otros modos.

3.3. Prueba 3.

Se presionaron los botones ph0 y ph3 en todos los modos que no sean medición, no debería pasar nada al hacer esto.

3.4. Prueba 4.

Esta prueba consiste en presionar PH3 en modo medición una vez, y seguirlo presionando reiteradas veces. Luego presionar PH0 y verificar que la velocidad marcada por el producto sea la diferencia de tiempo entre la primer pulsación de PH3 y no las demás.

3.5. Prueba 5.

Presionar PH3 y seguidamente presionar PH0 dejando muy poco tiempo entre las pulsaciones de ambos, con el fin de verificar que la velocidad sea inválida.

3.6. Prueba 6.

Llevar la máquina en modo configuración y repetir el protocolo de pruebas de la tarea 4, con el fin de ver que se comporte correctamente.

3.7. Prueba 7.

Poner breakpoints en los lugares donde se actualiza la pantalla LCD con el fin de verificar que esto solo ocurra una única vez, según corresponda.

3.8. Prueba 8.

Verificar que al volver de modo Medición o Libre a configuración, aun se conserve la velocidad máxima almacenada anteriormente.

3.9. Prueba 9.

Presionar el botón PH3 y dejarlo por más de 8 segundos antes de presionar ph0, la velocidad debe ser inválida. Esta prueba busca validar el control de rebase de TICK_VEL.

3.10. Prueba 10.

Verificar la temporización de las luces de alarma, con el fin de que no sean ni más rápidas ni más lentas. Además de verificar que las luces de alarma solo se enciendan cuando la velocidad del conductor es mayor a la máxima programada.

3.11. Prueba 11.

Verificar que el CAD realiza los 20 pasos de brillo para la pantalla (en la medida de lo posible).

3.12. Prueba 12.

Ir a modo medición, presionar ph3 y cambiar de modo. Al volver a modo medición todo debería funcionar como siempre, en caso de que algo esté diferente significa que hay un error.

3.13. Prueba 13.

Ir a modo medición, presionar ph3, esperar y presionar ph0, cambiar de modo y volver a modo medición y verificar que todo funcione de manera correcta.

3.14. Prueba 14.

Ir a modo medición, presionar ph3, esperar y presionar ph0, esperar que se despliegue la velocidad en la pantalla y en este momento cambiar de modo. Al volver a modo medición la pantalla debería estar apagada y todo funcionando como es de costumbre

4. Resultados

Las 14 pruebas mencionadas en la sección anterior fueron realizadas al menos dos veces cada una para el sistema diseñado sin problemas, comprobando así que el diseño es robusto.

5. Conclusiones y recomendaciones

A continuación se presenta una recopilación de las principales conclusiones obtenidas en este proyecto, con el fin de resumir los principales resultados que se obtuvieron

- Se puede destacar la gran importancia de realizar un protocolo de pruebas, esto debido a que es posible encontrar y reparar errores a partir de pruebas robustas e ingeniosas.
- Es posible realizar un producto funcional en la DRAGON 12 con el fin de probar su funcionamiento.
- Aunque se tengan múltiples interrupciones en el programa, funciona de manera correcta. Esto se debe a que la arquitectura realizada estuvo bien pensada para evitar colisiones de datos.
- Se demuestra la utilidad de los diagramas de flujo para realizar codificaciones y depurar.
- Se demuestra la utilidad de lo aprendido en el curso realizando una implementación de un producto relativamente complejo, por ser en lenguaje ensamblador.
- Al haberse realizado la implementación en lenguaje ensamblador, se tiene un control total de lo que está ocurriendo, por lo que se tiene una capa de abstracción casi nula, lo cuál es conveniente para el aprendizaje.

Seguidamente presentamos algunas recomendaciones a seguir a la hora de realizar un proyecto de este tipo, con el fin de que no se cometan errores que pueden causar que el trabajo sea más complicado.

- Utilizar la arquitectura planteada por el profesor para el programa, con el fin de evitar problemas de posibles errores de arquitectura que pueden ser inducidos por la falta de experiencia del estudiante.
- Si se utilizan dos distintas interrupciones del módulo de timers, analizar si es posible que activar el bit de borrado rápido genere algún tipo de colisión.
- Documentar extensivamente cada una de las subrutinas creadas, con el fin de que depurar no se vuelva un trabajo imposible. Además de tener todos los diagramas hechos antes de realizar la codificación del programa.
- No encender y apagar las interrupciones en lapsos cortos de tiempo, debido a que esto puede generar problemas serios de temporización en el módulo de timers.
- Probar exhaustivamente cada una de las subrutinas antes de crear otra que dependa de la misma, porque esto puede generar que se dure demasiado tiempo intentando depurar por no saber dónde se encuentra el error en específico.
- Pensar bien a la hora de depurar dónde colocar los 2 breakpoints disponibles de la Dragon 12, y revisar en dónde se encuentra el PC y qué contiene la memoria en ese momento que se detuvo el procesador. Esto con el fin de lograr encontrar errores de manera más sencilla.
- Utilizar repositorios con el fin de guardar distintas versiones del código del proyecto, ya que puede ser que se tenga que recurrir a copias viejas para restaurar funcionamientos en caso de que se de una falla y no se sepa qué fue lo que se cambió.

Referencias

- [1] Delgado, Geovanny. *II PARTE. DISPOSITIVOS PERIFÉRICOS*. Escuela de ingeniería Eléctrica. UCR. 2019