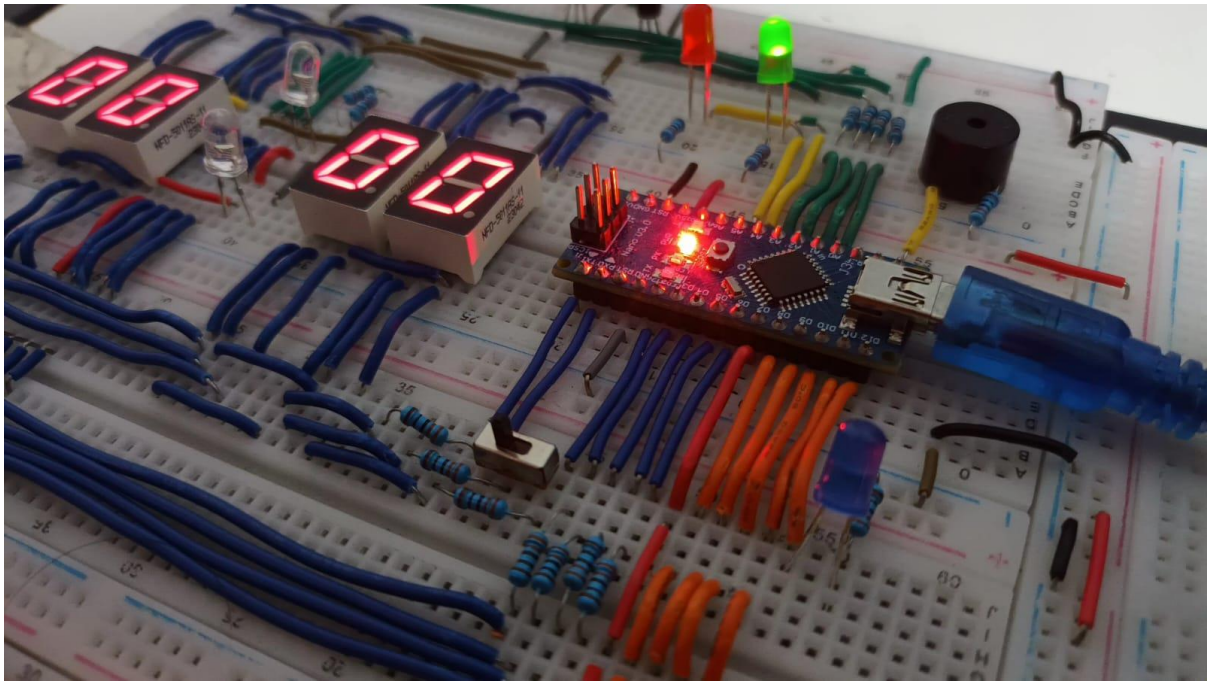
Proyecto 1 - Programación de Microcontroladores

Reloj digital



Universidad del Valle de Guatemala

Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica

Mario Alejandro Betancourt Franco, 23440

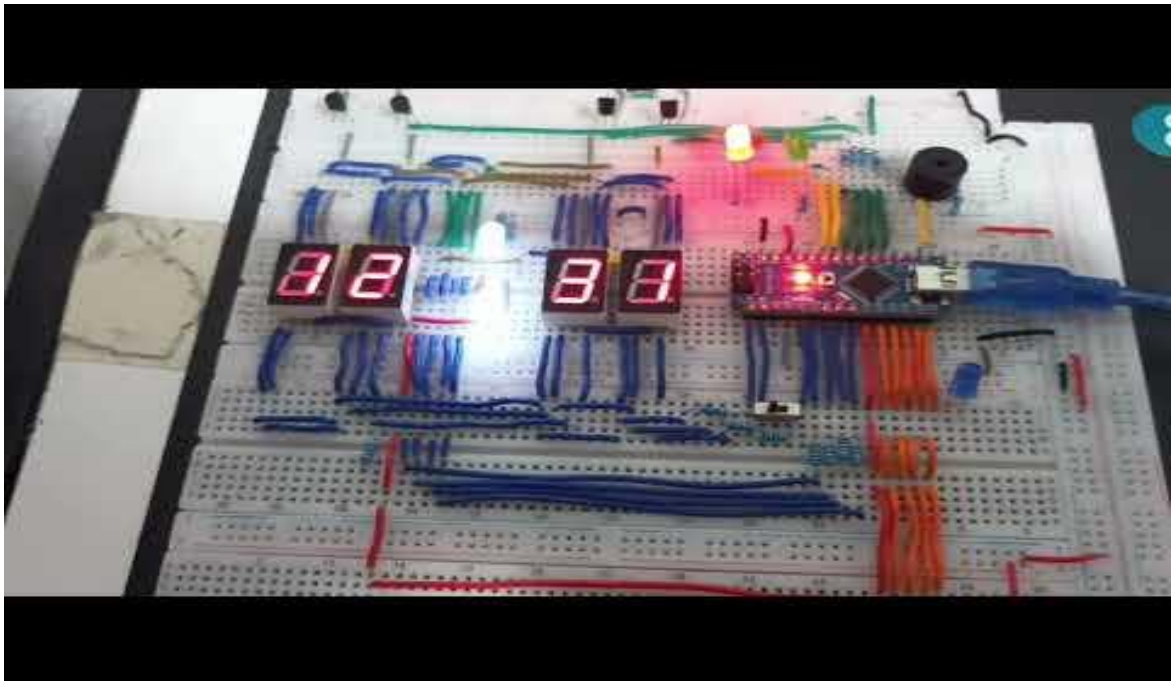
Tabla de contenidos

1. Introducción y Enlaces

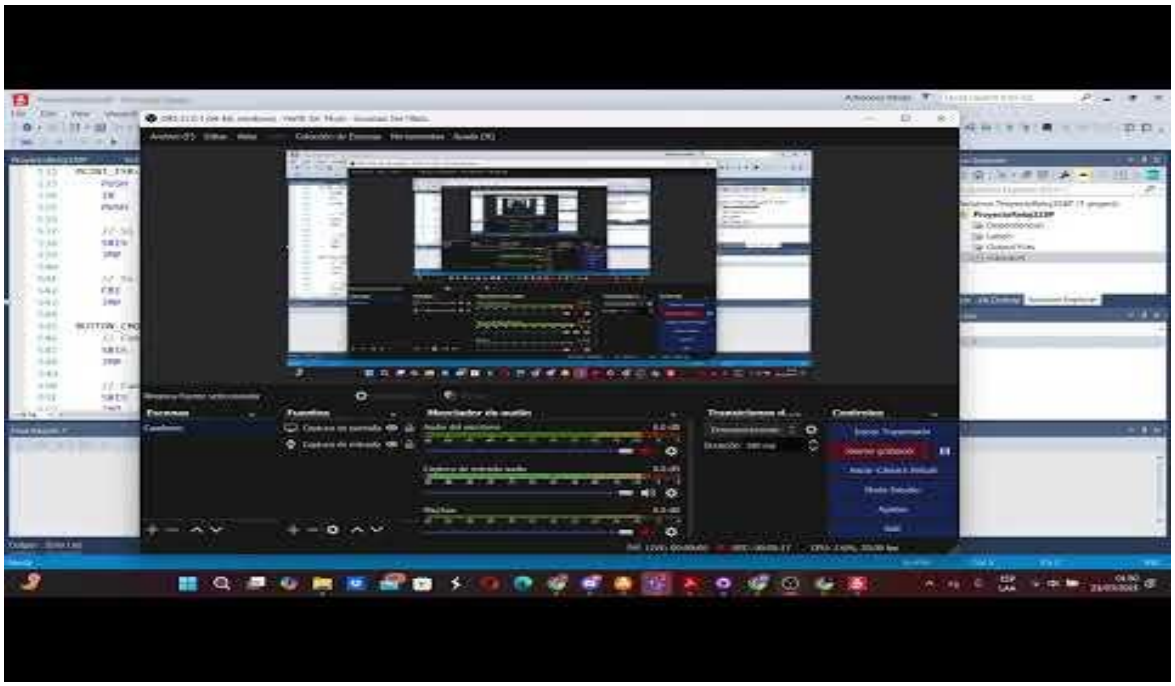
En este documento se presenta la documentación técnica del Proyecto 1 de Programación de Microcontroladores. Este consiste en la fabricación de un reloj digital usando el microcontrolador ATMega328P (En un Arduino Nano) programado en lenguaje ensamblador. El reloj dispone de cuatro displays de 7 segmentos multiplexados, tres LEDs indicadores de modo, dos LEDs indicadores de segundos y un buzzer piezoeléctrico. El reloj tiene funcionalidades para mostrar la hora y la fecha, así como para configurar una alarma dada la hora a la que debe activarse. Con cuatro botones el usuario puede cambiar de modo operacional y configurar la hora, fecha y la hora de la alarma.

Enlace a Github: https://github.com/mariobet23440/Proyecto1_PrograDeMicros

Enlace a Youtube Hardware: https://youtu.be/wzRZjsj6z4c



Enlace a Youtube Software: https://youtu.be/IdqRCAtvhnM



Disclaimer: No estoy seguro de si el video era sobre el Hardware o el Software ya que en la rúbrica y el documento no se especifica.

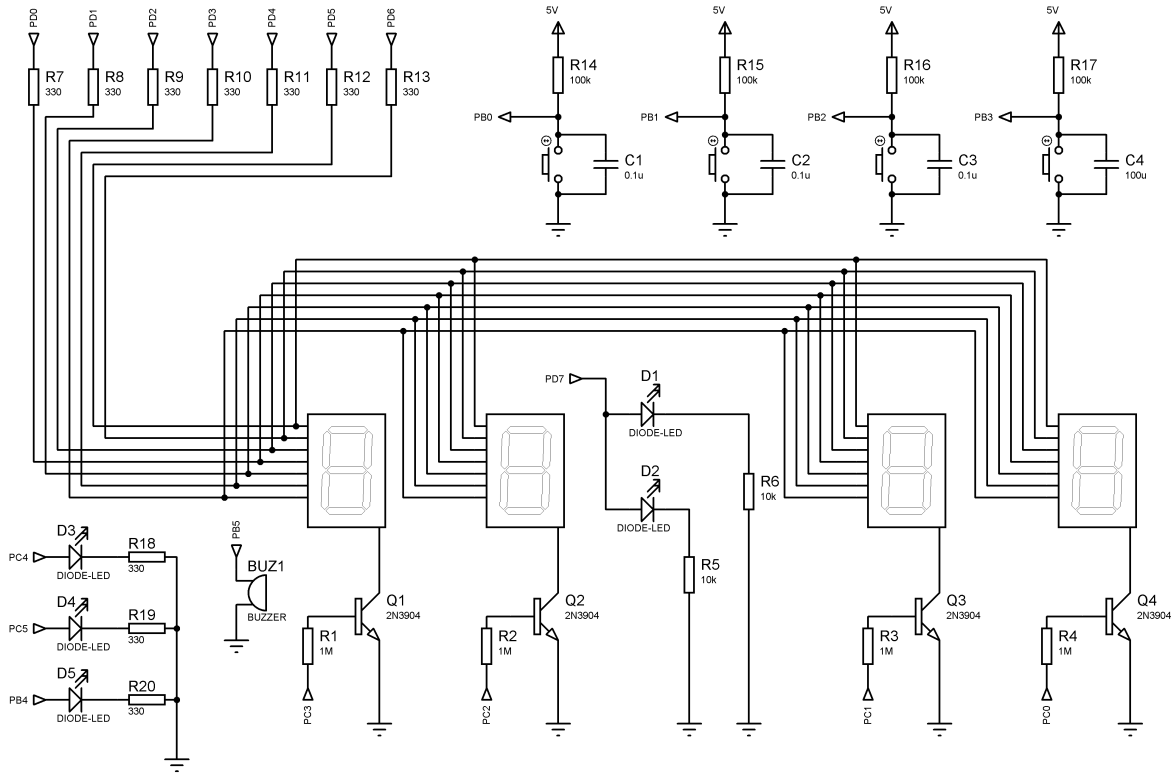
Instrucciones

Este proyecto se trabajará de forma **individual**. La fecha de entrega es la semana 17 – 21 de marzo durante su período de laboratorio. Deberá hacer un **trabajo escrito** (no se imprime, se sube en PDF a Canvas, antes del día de su presentación) donde explique todas sus configuraciones, esquemáticos, cálculos y código debidamente comentado. También deberá subir un **video** a YouTube explicando el funcionamiento de su proyecto.

Considere que este documento incluye una descripción detallada del programa implementado, descrito en términos de sus bloques funcionales. El video queda como un registro de que el proyecto funciona.

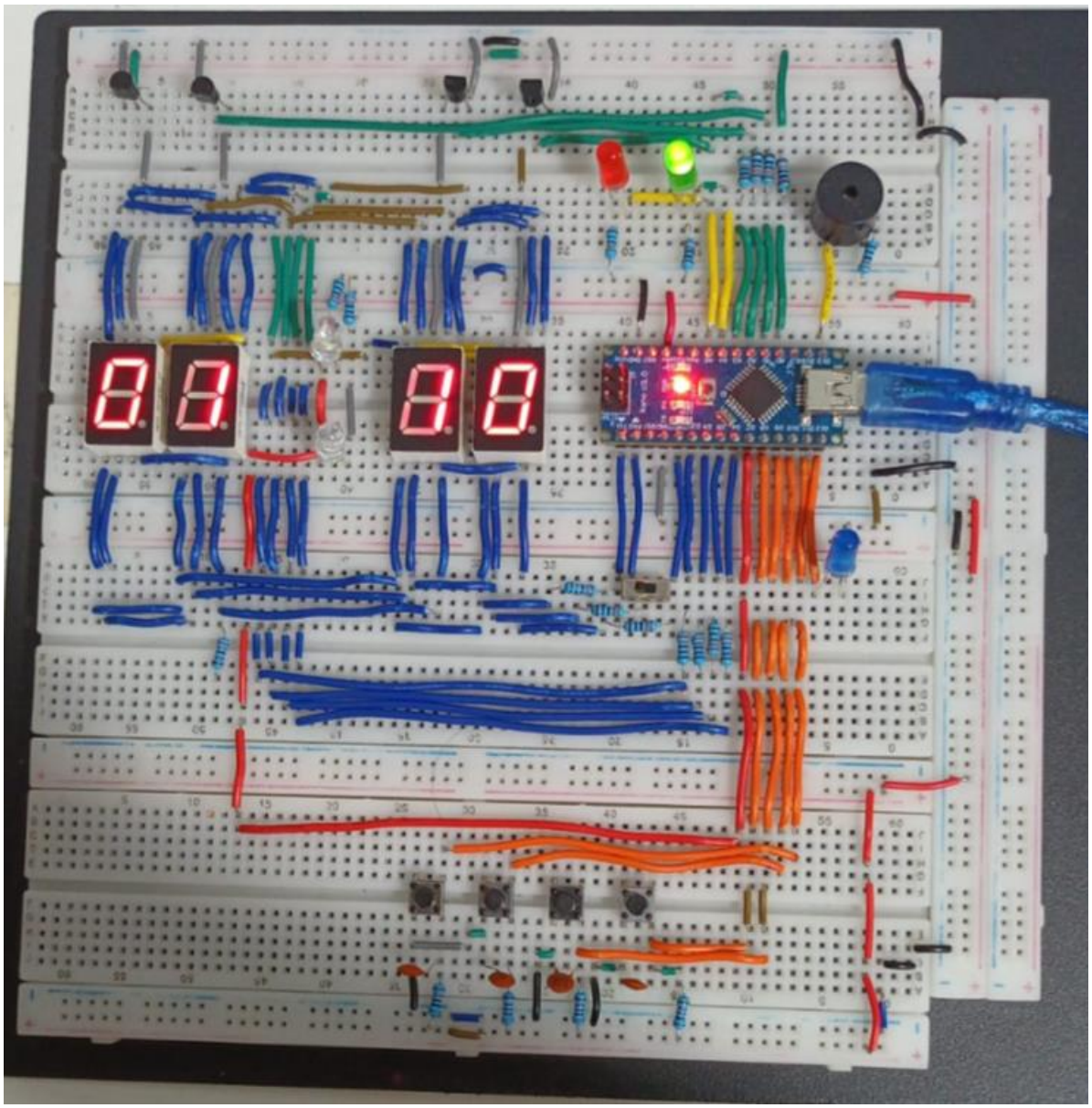
2. Esquemáticos

Figura 1 - Esquemático de Circuitos Conectados a Pines de Arduino Nano



3. Implementación de Circuito en Protoboard

Figura 2 – Circuito Implementado en Protoboard



En la imagen el LED verde es para el indicador de hora, el LED rojo es para el indicador de fecha y el LED azul es para el indicador de configuración de alarma.

4. Tablas de Displays de 7 Segmentos

A continuación, se muestra la tabla de los bits activos al mostrar cada número del 1 al 10 en los displays de 7 segmentos, así como la asignación de segmentos por cada bit de PORTD.

Tabla 1 – Correspondencia entre segmentos de displays y los pines que los controlan

Segmento controlado	Bit en PORTD
a	PD5
b	PD6
c	PD2
d	PD0
e	PD1
f	PD4
g	PD3

Tabla 2 – Segmentos encendidos por número mostrado en displays de 7 segmentos

Número	**Segmentos**	**Pines de PORTD**	**Binario (PORTD)**
0	A,b,c,d,e,f	Todos menos G (PD3)	1110111
1	B,c	6, 2	1000100
2	A,b,d,e,g	6,5,3,1,0	1101011
3	A,b,c,d,g	6,5,3,2,0	1101101
4	B,c,f,g	6,2,4,3	1011100
5	A, c,d, f, g	5, 2, 0, 4, 3	0111101
6	A,c,d,e,f,g	5,4,3,2,1,0	0111111
7	A,b,c	6,5,2	1100100
8	A,b,c,d,e,f,g	6,5,4,3,2,1,0	1111111
9	A,b,c,f,g	6,5,4,3,2	1111100

5. Configuración de Reloj de Sistema

Con la siguiente ecuación es posible determinar el valor máximo del intervalo de tiempo entre interrupciones de un temporizador, sea T_{max}, con un contador de n bits:

$$T_{max} = \frac{2^n \times prescaler}{f_{clk}}$$

Donde f_{clk} es la frecuencia de reloj del sistema. Al modificar el valor inicial de los contadores al reiniciar los timers, es posible reducir el intervalo de tiempo entre desbordamientos en un contador, siempre en periodos menores a T_{max}.

Debido a que TIMER1 se utilizará para la función del contador de minutos, es necesario que el periodo entre interrupciones por desbordamientos en este temporizador sea lo más cercano posible a 60 segundos. Por la relación de proporcionalidad entre variables, se observa que *el intervalo de tiempo incrementa conforme se aumenta el valor del prescaler y se disminuye la frecuencia de reloj del sistema*. Para hacer transiciones imperceptibles (Importante para el multiplexado de los displays), se escogerá una frecuencia de reloj de sistema de 1 MHz. Usando $n = 16$ y un prescaler de 1024 se obtiene

$$T_{max} = \frac{2^{16} \times 1024}{10^6 Hz} = 67.1s$$

Este es un valor aceptable. Por lo tanto, se supondrá una frecuencia de reloj de sistema de ***1 MHz*** en lo que resta de las memorias de cálculo de los temporizadores. El periodo entre desbordamientos máximos para TIMER0 y TIMER2 se calcula de forma similar:

$$T_{max} = \frac{2^8 \times 1024}{10^6 Hz} = 0.26\, s$$

Como se puede observar, este periodo es insuficiente para que por cada ciclo de desbordamiento se enciendan los leds indicadores de segundos (Los cuales deben mantenerse encendidos durante 500 ms). Sin embargo, estas limitantes de tiempo pueden resolverse usando las técnicas apropiadas, como se mostrará más adelante. Para mantener al intervalo de tiempo entre desbordamientos de TIMER1 lo más grande posible, se trabajará con una frecuencia de 1MHz, y con prescalers de 1024 para los tres temporizadores.

Tabla 3 – Configuraciones de Reloj de Sistema

Frecuencia de reloj de sistema	1 MHz
Factor de division de reloj	16
Bits Activos en el Registro Clock Prescale (CLKPR)	CLKPS2

6. Configuraciones de Timers

En este apartado se presenta una memoria de cálculo para los valores iniciales y los prescalers de TIMER0, TIMER1 y TIMER2. Los tres temporizadores *se utilizan en modo normal* y se reinician con llamadas que se pueden emplear a lo largo del código.

Tabla 4 – Uso de Timers en el proyecto

Temporizador	Delay entre Overflows	Función que cumple en el Proyecto
TIMER0	0.20 ms	Multiplexado
TIMER1	60s	Contar minutos
TIMER2	125 ms	LEDs Indicadores de Segundos. Parpadeo de displays durante cambio de contadores.

6.1. Prescaler de TIMER0 y TIMER1

Como se mostró en la sección anterior, al utilizar una frecuencia de reloj de 1 MHz, el temporizador 1 puede contar hasta 67 segundos, tiempo suficiente para realizar un contador de minutos. Este valor se obtuvo al considerar un prescaler de 1024. Considerando que el TIMER0 y el TIMER1 comparten prescaler en la arquitectura del ATMega328P, se utiliza este valor para los prescalers de ambos temporizadores.

6.2. Valor inicial de TIMER0

Usando un prescaler de 1024, se calcula el valor inicial del TIMER0 para ser operado en modo normal con la siguiente ecuación:

$$TCNT0 = 256 - \frac{f_{clk} \times t_{deseado}}{prescaler}$$

Sustituyendo valores:

$$TCNT0 = 256 - \frac{10^6 Hz \times (0.20 \times 10^{-3} s)}{1024} = 236$$

$$TCNT0 = 256 - \frac{10^6 Hz \times (5 \times 10^{-3} s)}{1024} = 251$$

6.3. Valor inicial de TIMER1

Usando un prescaler de 1024, se calcula el valor inicial del TIMER0 para ser operado en modo normal con la siguiente ecuación:

$$TCNT1 = 65536 - \frac{f_{clk} \times t_{deseado}}{prescaler}$$

Sustituyendo valores:

$$TCNT1 = 65{,}536 - \frac{10^6 Hz \times 60s}{1024} = 6942$$

6.4. Prescaler de TIMER2

Para calcular el prescaler necesario para el TIMER2 se utiliza la siguiente fórmula:

$$prescaler \geq \frac{t_{delay} \times f_{clk}}{2^n}$$

$$prescaler \geq \frac{(0.125\ s) \times (10^6\ Hz)}{2^8}$$

$$prescaler \geq 488$$

Es posible configurar el TIMER2 para que sus desbordamientos ocurran cada 0.125 segundos usando un prescaler de 1024.

6.5. Valor Inicial del TIMER2

Se calcula el valor inicial del timer 2 usando la siguiente fórmula

$$TCNT2 = 256 - \frac{f_{clk} \times t_{deseado}}{prescaler}$$

Sustituyendo valores

$$TCNT2 = 256 - \frac{10^6 Hz \times (125 \times 10^{-3} s)}{1024} = 133$$

6.6. Resumen de Configuraciones

Tabla 3 - Configuración de Temporizadores

Temporizador	Valor Inicial	Prescaler	Delay entre Overflows
TIMER0	236	1024	0.20 ms
TIMER1	6942	1024	60 s
TIMER2	133	1024	125 ms

6.7. Contadores como Divisores de Frecuencia

Si bien la elección de los delays de TIMER0 y TIMER1 se justifican en la necesidad de transiciones rápidas para el multiplexado y el conteo precisos de minutos para que el reloj sea funcional, aún no se ha mencionado por qué el delay del TIMER2 es de 125 ms si está destinado a ser utilizado para un indicador de segundos. Como se pudo observar anteriormente, con una frecuencia de reloj de 1 MHz es imposible lograr delays de 500 ms con el TIMER0 y el TIMER2. Entonces, ¿Cómo se puede realizar este delay dadas las limitaciones de los TIMERs?

En el transcurso del desarrollo del proyecto se han encontrado dos formas distintas de realizar la función del indicador de segundos. La primera fue utilizar un registro para almacenar un contador decimal que después de incrementar cierto número de veces (Usando un delay de 100 ms se hacen 5 conteos) utiliza un XOR sobre el bit 7 de PORTD. Si bien esto fue funcional inicialmente, conforme se agregaron más funciones dentro del programa se observó que la frecuencia eventualmente se volvía inestable y el manejo de los LEDs se volvía complicado a nivel algorítmico.

Recordando lo aprendido sobre lógica secuencial se encontró una solución distinta. En lugar de mantener el contador y el cambio del MSB de PORTD en funciones separadas, es posible utilizar un bit máscara de un registro que funge como un contador. Si se toma un registro contador cuyo valor incrementa por cada periodo de desbordamiento de TIMER2, el LSB debe cambiar de valor entre dos desbordamientos, volviendo a su valor original después de dos ciclos. El bit inmediatamente superior al LSB incrementa después del primer ciclo de oscilación. Ocurre lo mismo en el bit superior y así sucesivamente. En otras palabras, al tomar los bits de un registro contador como salidas lógicas, se construye una señal de reloj con una frecuencia igual al cociente entre la frecuencia de la “señal de entrada” (La frecuencia entre desbordamientos) y una potencia de 2. En palabras más simples, el contador funciona como un *divisor de frecuencias*.

La razón de la elección de un delay de 125 ms para TIMER2 es que 125 resulta de dividir 500 entre 4. Si el bit 0 del contador cambia cada 125 ms, el bit 1 debe cambiar cada 250 ms y el bit 2 cambia de valor cada 500 ms. Por lo tanto, fuera de la rutina de interrupción de TIMER2 se usa el bit 2 de un contador que cambia cada 125 ms para

obtener una señal de reloj con un periodo de 1 segundo. Este contador se denominó ***T2COSC*** (Timer 2 Custom Oscilator)

7. Modos Operativos y Máquinas de Estado Finito

Para permitirle al usuario el cambio de los contadores en los modos de indicador de horas, fecha y en el modo para establecer una alarma, se utilizaron 9 estados distintos, cada uno identificado por un valor del registro de propósito general STATE.

Figura 3 - Máquina de Estados Finitos

Modo Operacional	Estado	Descripción
MostrarHora	S0	El reloj muestra la hora. Al presionar cualquiera de los botones de incremento y decremento no pasa nada.
CambiarMins	S1	Con dos botones se incrementa/decrementa el contador de minutos. El reloj opera en el modo indicador de horas. El LED del indicador de horas está encendido. Los displays que muestran los minutos parpadean cada 250 ms.
CambiarHoras	S2	Con dos botones se incrementa/decrementa el contador de horas. El reloj opera en el modo indicador de horas. Los displays que muestran las horas parpadean cada 250 ms.
MostrarFecha	S3	El reloj muestra la fecha. Al presionar cualquiera de los botones de incremento y decremento no pasa nada.
CambiarDias	S4	Con dos botones se incrementa/decrementa el contador de días. El reloj opera en el modo indicador de fecha. El LED del indicador de fecha está encendido. Los displays que muestran el día parpadean cada 250 ms
CambiarMeses	S5	Con dos botones se incrementa/decrementa el contador de meses. El reloj opera en el modo indicador de fecha. El LED del indicador de fecha está encendido. Los displays que muestran los meses parpadean cada 250 ms.
ModoAlarma	S6	Modo para configurar alarma. Los displays no cambian con la hora.
AlarmaMinutos	S7	Permite modificar el valor de los minutos en el modo alarma.
AlarmaHoras	S8	Permite modificar el valor de las horas en el modo alarma.

Como se puede observar en la tabla 5, los estados están codificados por combinaciones de bits *one-hot,* para facilitar la elaboración de comparaciones sucesivas (Sentencias switch-case).

Tabla 5 - Encodings de estados (One Hot)

Estado	Encoding Binario	Bit encendido	Encoding Hexadecimal
S0	0000 0000	0	0X00
S1	0000 0001	1	0X01
S2	0000 0010	2	0X02
S3	0000 0100	3	0X04
S4	0000 1000	4	0X08
S5	0001 0000	5	0X10
S6	0010 0000	6	0X20
S7	0100 0000	7	0X40
S8	1000 0000	8	0X80

Tabla 6 - Tabla de Transiciones de Estados

Estado Actual	PB0	PB1	Estado Siguiente
MostrarHora (S0)	1	X	MostrarFecha (S3)
MostrarHora (S0)	X	1	CambiarMinutos (S1)
CambiarMinutos (S1)	1	X	MostrarHora (S0)
CambiarMinutos (S1)	X	1	CambiarHora (S2)
CambiarHora (S2)	X	X	MostrarHora (S0)
MostrarFecha (S3)	1	X	ModoAlarma (S6)
MostrarFecha (S3)	X	1	CambiarDias (S4)
CambiarDias (S4)	1	X	MostrarFecha (S3)
CambiarDias (S4)	X	1	CambiarMeses (S5)
CambiarMeses (S5)	X	X	MostrarFecha (S3)
ModoAlarma (S6)	1	X	MostrarHora (S0)
ModoAlarma (S6)	X	1	AlarmaMinutos (S7)
AlarmaMinutos (S7)	1	X	ModoAlarma (S6)
AlarmaMinutos (S7)	X	1	AlarmaHoras (S8)
AlarmaHoras (S8)	X	X	ModoAlarma (S6)

Tabla 7 - Tabla de Transiciones de Estados con Encodings Hexadecimales

Estado Actual	PB0	PB1	Estado Siguiente
0X00	1	X	0X08
0X00	X	1	0X01
0X01	1	X	0X00
0X01	X	1	0X02
0X02	X	X	0X00
0X04	1	X	0X20
0X04	X	1	0X08
0X08	1	X	0X04
0X08	X	1	0X10
0X10	X	X	0X08
0X20	1	X	0X00
0X20	X	1	0X40
0X40	1	X	0X20
0X40	X	1	0X80
0X80	X	X	0X20

8. Resumen del Código - Diagramas de Flujo

A continuación, se presenta un breve resumen de las funciones realizadas dentro del código en lenguaje ensamblador. Para facilitar la comprensión del flujo del programa, se muestran los diagramas de flujo de las funciones principales.

8.1. Uso de Puertos del ATMega328P

Tabla 8 – Funciones que realiza cada pin en el Arduino Nano

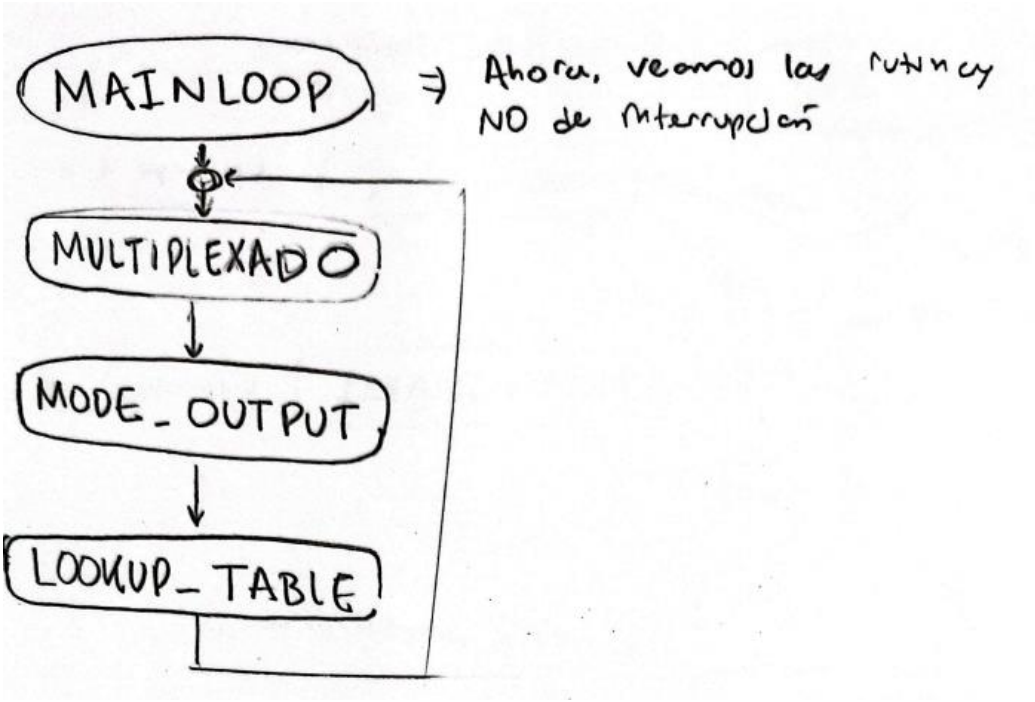
Pines de Puertos	Uso
PD0-PD6	Displays de 7 segmentos, señales de segmentos.
PD7	LEDs Intermitentes entre Displays (Indicador de Segundos).
PB0-PB3	Botones de entradas
PB4	LED indicador de alarma
PB5	Buzzer de Alarma
PC0-PC3	Señales a transistores de multiplexado.
PC4-PC5	LEDs de indicadores de modo hora y fecha.

8.2. Mainloop

Dentro del ciclo principal del programa se realizan 3 subrutinas no de interrupción:

1. MULTIPLEXADO: Encender el display activo según la señal de multiplexado (MUX_SIGNAL)
2. MODE_OUTPUT: Escoger la dirección en la tabla de consulta a la que debería apuntarse de acuerdo con el estado actual (STATE) de la máquina de estados finitos. La dirección se almacena en el registro OUT_PORTD.
3. LOOKUP_TABLE: Sacar las salidas a los displays de 7 segmentos de la tabla de consulta en la memoria de programa con OUT_PORTD.

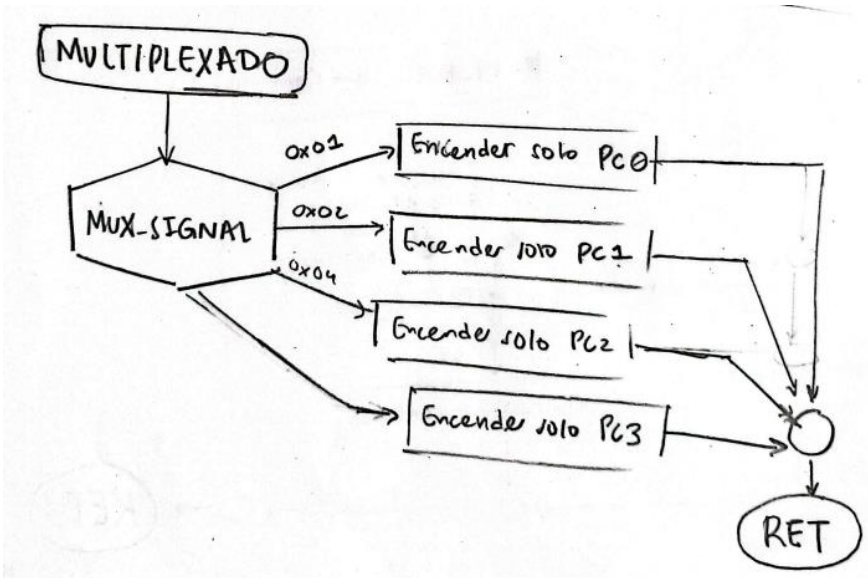
Debido a que estas tres son rutinas para mostrar los contadores, incluirlas dentro del ciclo principal no afecta el flujo del programa.



8.3. Multiplexado – Encender un Display a la vez

En la subrutina de Multiplexado se realiza una secuencia de comparaciones encadenadas ("*if*s encadenados") para determinar qué bit de PORTC (PC0-PC3 conectados a transistores de multiplexado) debe encenderse para encender el display indicado por la señal de multiplexado (MUX_SIGNAL). De acuerdo con el valor de MUX_SIGNAL, se enciende un único bit de PC0-PC3, para encender los segmentos de un único display a la vez.

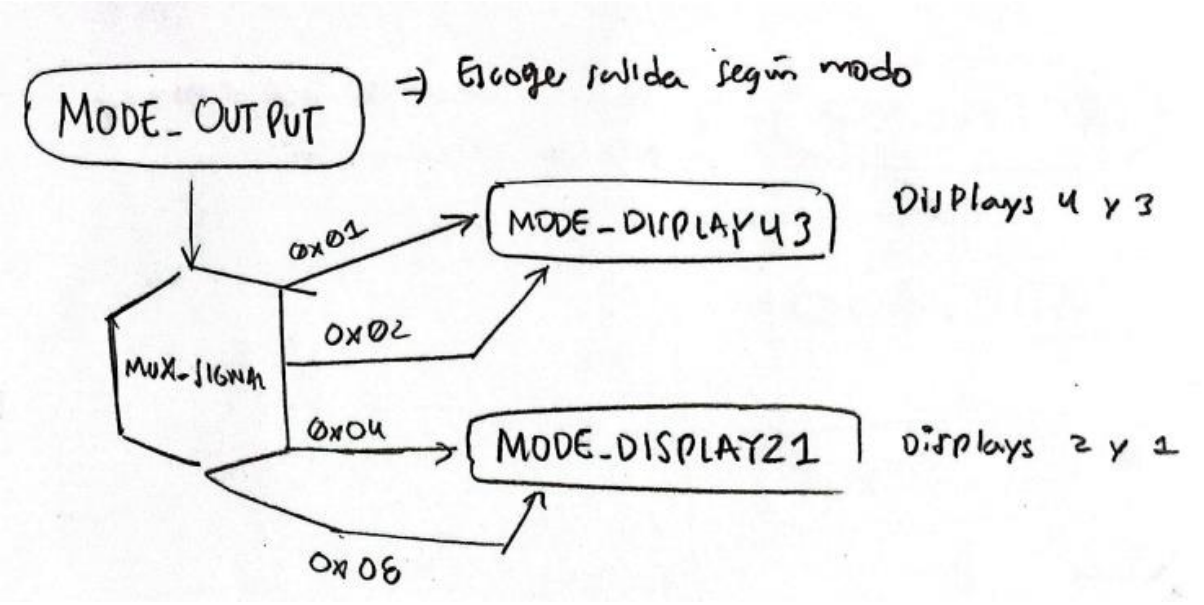
Figura 5 - Multiplexado



8.4. Selector de Salida a Displays de 7 Segmentos según Estado

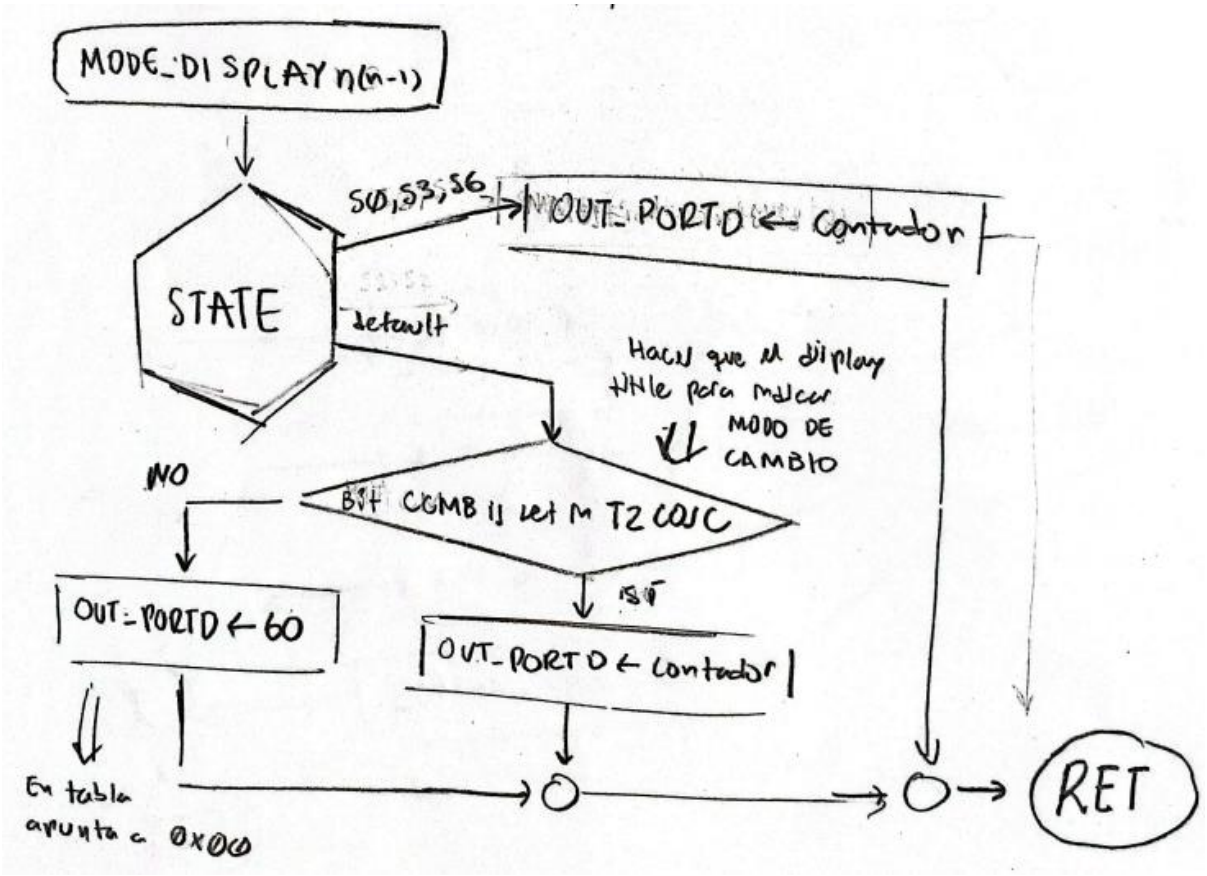
La rutina que le sigue a MULTIPLEXADO es MODE_OUTPUT, donde se determina la dirección a la que el puntero Z debería apuntar de acuerdo con el modo operacional actual del reloj. Cuando la subrutina inicia lo primero que hace es una secuencia de comparaciones encadenadas para determinar si los displays que se deben encender son los del contador "alto" (Horas y meses) o el contador "bajo" (Días y horas). Si MUX_SIGNAL es 0X01 o 0x02 se realiza un salto a una subrutina específica para los displays 4 y 3 (Horas y meses). Por el contrario, si MUX_SIGNAL es igual a 0x04 o 0x08 se realiza un salto a una subrutina especializada para los displays 1 y 2. Nótese que MUX_SIGNAL es una señal con codificación *one-hot*. Esta fue una elección de diseño para facilitar comparaciones como las de esta función.

Figura 6 – Selector de salida según modo



El esquema general de MODE_DISPLAY32 y MODE_DISPLAY21 es similar y el diagrama de flujo siguiente resume lo realizado en ambas subrutinas. Si el estado actual es S0, S3 o S6 (MostrarHora, MostrarFecha o ModoAlarma) al registro utilizado para controlar la dirección del puntero Z se le asigna el valor del contador correspondiente al modo operacional (MINUTE_COUNT, HOUR_COUNT, DAY_COUNT y MONTH_COUNT). En cualquier modo con prefijo "Cambio" (E.g. CambioMinutos, CambioHoras, etc.) se determina si el bit CCMB (Counter Change Mask Bit) del registro contador T2COSC está encendido. Si está encendido se muestra el contador correspondiente. Si no, a OUT_PORTD se le asigna un valor de 60, que apunta a una dirección que almacena 0x00. Esto permite que los displays de un contador titilen para indicar que se está cambiando su valor y muestren el contador mientras que están encendidos. Posteriormente se mostrará cómo se controla T2COSC.

Figura 7 - Determinación de dirección de puntero Z



8.5. Tabla de Consulta

La siguiente subrutina, LOOKUP_TABLE, extrae datos de una tabla de consulta utilizando el valor almacenado en OUT_PORTD. La tabla de consulta almacena las salidas a los displays de 7 segmentos y está organizada de una forma especial para mostrar números de dos dígitos.

En lugar de almacenar las secuencias de bits que muestran los números del 0 al 9, los datos almacenados en la tabla de consulta representan secuencias sucesivas de decenas y unidades de números de dos dígitos desde 00 hasta 60. Cualquier dirección con un índice par (Asumiendo que el primer índice es el índice cero) almacena las decenas de un número, mientras que las direcciones impares muestran las unidades. Por lo tanto, si el usuario desea mostrar un número arbitrario, debe apuntar a una dirección igual al doble de ese número para mostrar sus decenas, y sumarle uno a la dirección del puntero para mostrar las unidades. Por ejemplo, si se deseara mostrar las decenas del número 30, se debe apuntar a la dirección 60 de la tabla de consulta. Si se desea mostrar las unidades del mismo número, el puntero Z debe apuntar hacia la dirección 61.

Fragmento de Código 1 – Tabla en Program Memory

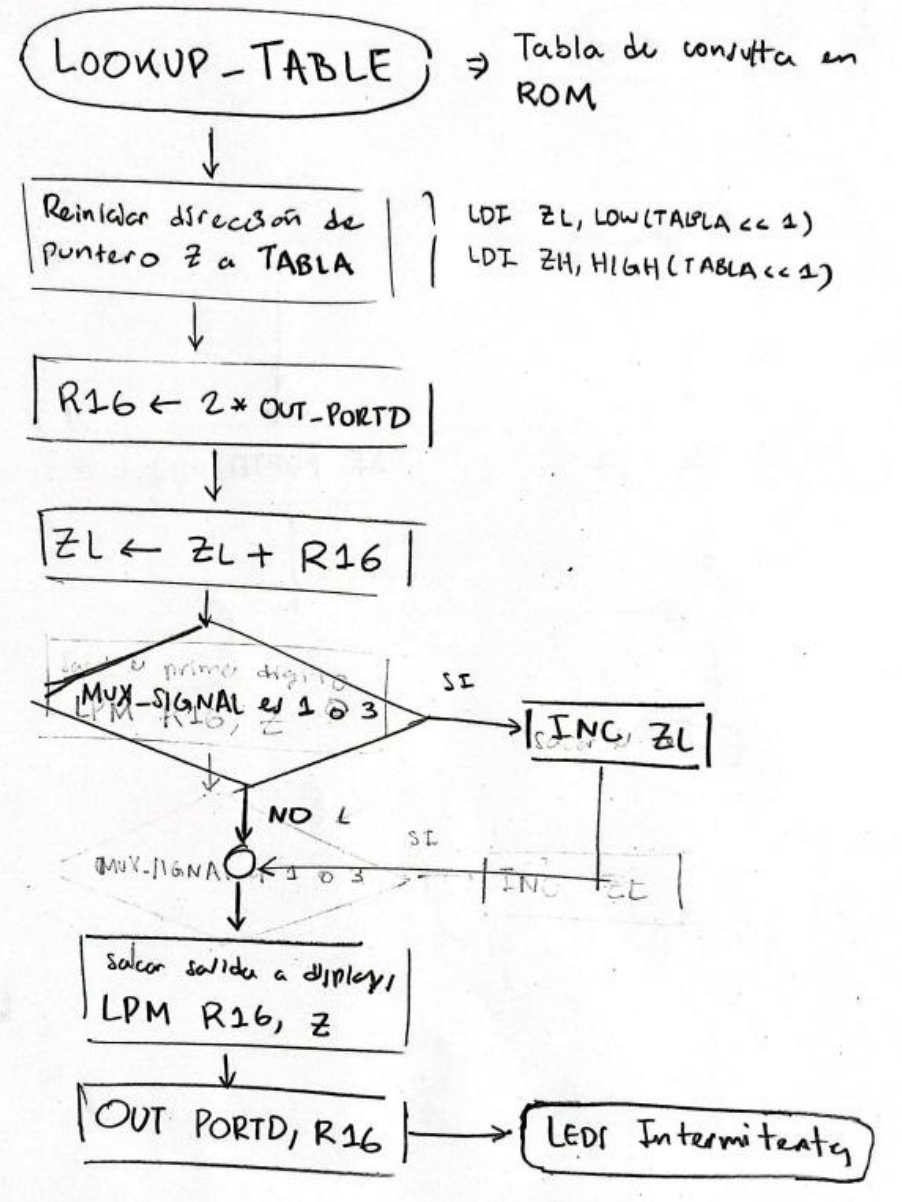
```
// ------------------------------------------------------------------
// | TABLAS                                                         |
// ------------------------------------------------------------------
// Definir la tabla en la memoria FLASH (Números del 1 al 10 en display de 7 segmentos)
.org    0x500
TABLA:
    .db T0, T0, T0, T1, T0, T2, T0, T3, T0, T4, T0, T5, T0, T6, T0, T7, T0, T8, T0, T9
    .db T1, T0, T1, T1, T1, T2, T1, T3, T1, T4, T1, T5, T1, T6, T1, T7, T1, T8, T1, T9
    .db T2, T0, T2, T1, T2, T2, T2, T3, T2, T4, T2, T5, T2, T6, T2, T7, T2, T8, T2, T9
    .db T3, T0, T3, T1, T3, T2, T3, T3, T3, T4, T3, T5, T3, T6, T3, T7, T3, T8, T3, T9
    .db T4, T0, T4, T1, T4, T2, T4, T3, T4, T4, T4, T5, T4, T6, T4, T7, T4, T8, T4, T9
    .db T5, T0, T5, T1, T5, T2, T5, T3, T5, T4, T5, T5, T5, T6, T5, T7, T5, T8, T5, T9
    .db 0x00, 0x00

// Observe que los últimos dos datos son para mostrar nada (OUT_PORTD debe ser 61)
```

Fragmento de código 2 – Constantes utilizadas para definir tabla de consulta

```
// Constantes de Lookup Table
.equ    T0 = 0b1110111
.equ    T1 = 0b1000100
.equ    T2 = 0b1101011
.equ    T3 = 0b1101101
.equ    T4 = 0b1011100
.equ    T5 = 0b0111101
.equ    T6 = 0b0111111
.equ    T7 = 0b1100100
.equ    T8 = 0b1111111
.equ    T9 = 0b1111100
```

El uso de una tabla de consulta no sólo evita comparaciones innecesarias, sino que también permite operar en intervalos de tiempo uniformes y evitar retrasos desiguales al mostrar números distintos. La única instancia donde el número de ciclos máquina utilizados cambia es cuando se muestran decenas y luego las unidades. Sin embargo, el número de líneas adicionales en este caso es pequeño y la diferencia de tiempos es insignificante.

Figura 8 – Tabla de consulta

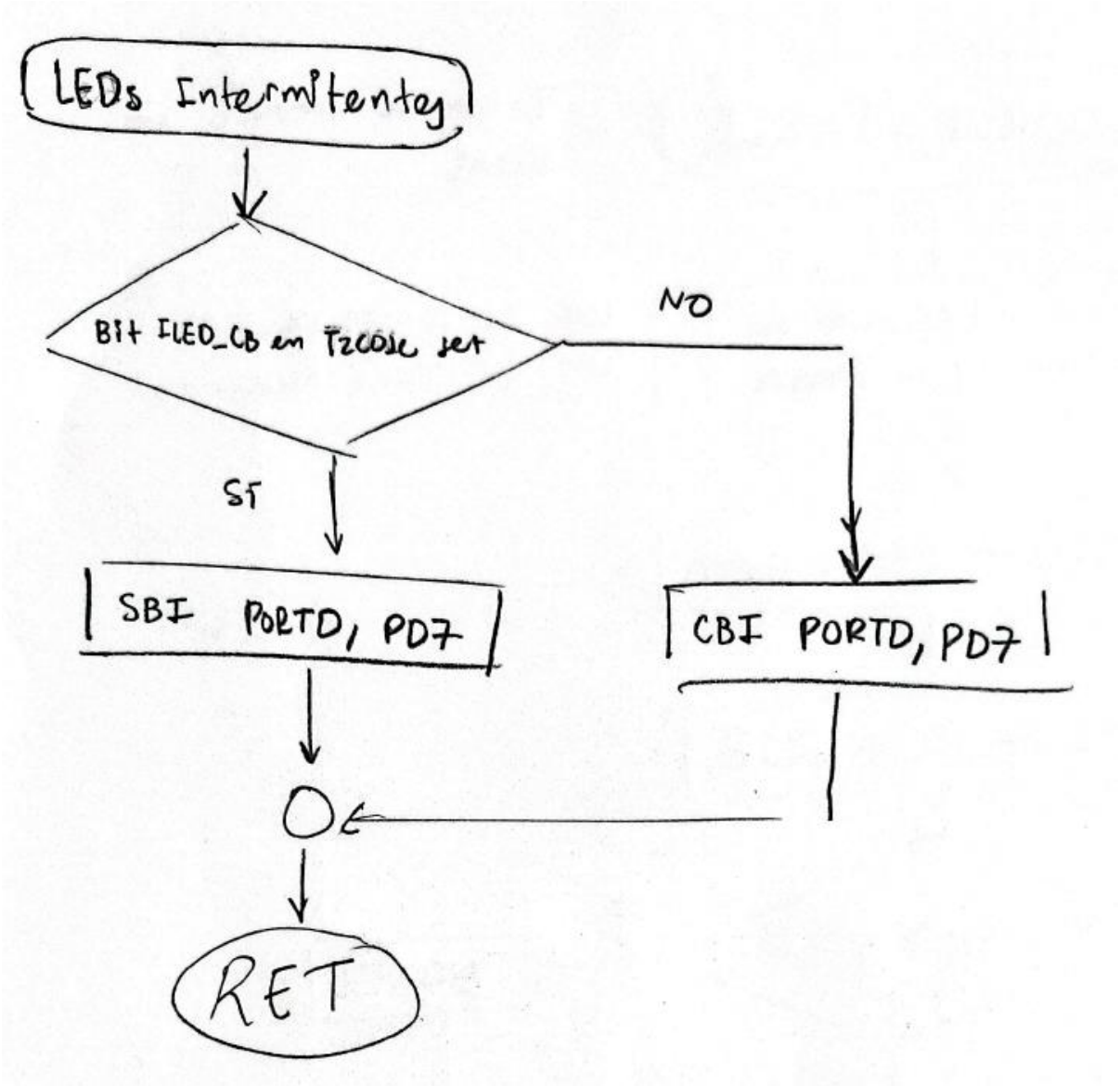


Inmediatamente después de terminar la rutina de la tabla de consulta, se hace un salto a la rutina de activación/desactivación de los LEDs intermitentes.

8.6. LEDs Intermitentes

Para realizar un indicador de segundos, se conectaron dos LEDs blancos entre los dos pares de displays de 7 segmentos. Estos LEDs parpadean en un periodo de 1 segundo, manteniéndose apagados durante 500 ms y encendidos en la otra mitad. Para determinar si los LEDs deben encenderse o apagarse, se utiliza un bit del registro contador T2COSC. Como se explicó anteriormente, cualquier bit de este registro puede ser utilizado como un generador de señal de reloj, con una frecuencia menor a la frecuencia de desbordamientos de TIMER2. Si el bit en cuestión, denominado ILED_CB (Intermittent LEDs Control Bit) está apagado, el bit PD7 de PORTD se apaga. Si el bit está encendido, el bit PD7 se enciende. Esto se hace después de sacar el dato extraído de la memoria de programa en PORTD para no sobrescribir el bit PD7 y alterar la oscilación de la señal.

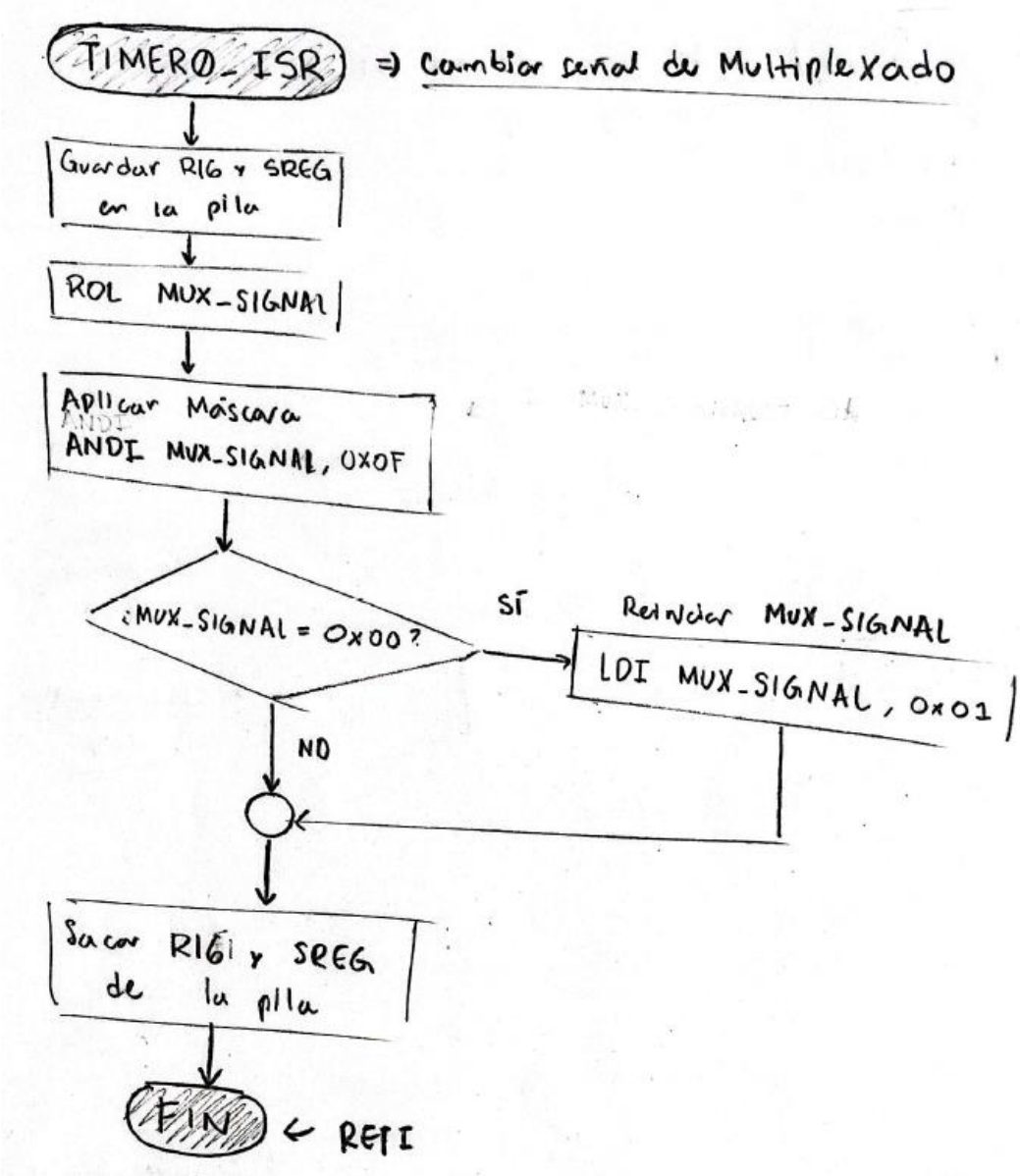
Figura 9 – LEDs intermitentes



8.7. Rotación de la señal de multiplexado (TIMER0_ISR)

En la rutina de interrupción de TIMER0 se realiza una rotación lógica a la izquierda (ROL) de los contenidos almacenados en MUX_SIGNAL. Esta señal se inicializa en 0X01, por lo que en cada ejecución de la instrucción ROL MUX_SIGNAL, el único bit encendido se mueve un bit a la izquierda. Esto se hace hasta que el bit encendido pasa al segundo byte del contador. Para verificar si esto ha ocurrido, se aplica una máscara con 0X0F para eliminar los contenidos del byte alto de MUX_SIGNAL. Si el bit de la señal se pasa, después de aplicar la máscara el contenido de MUX_SIGNAL será igual a cero y la bandera Z del SREG se encenderá. Si esto ocurre se reinicia el valor de MUX_SIGNAL a 0x01.

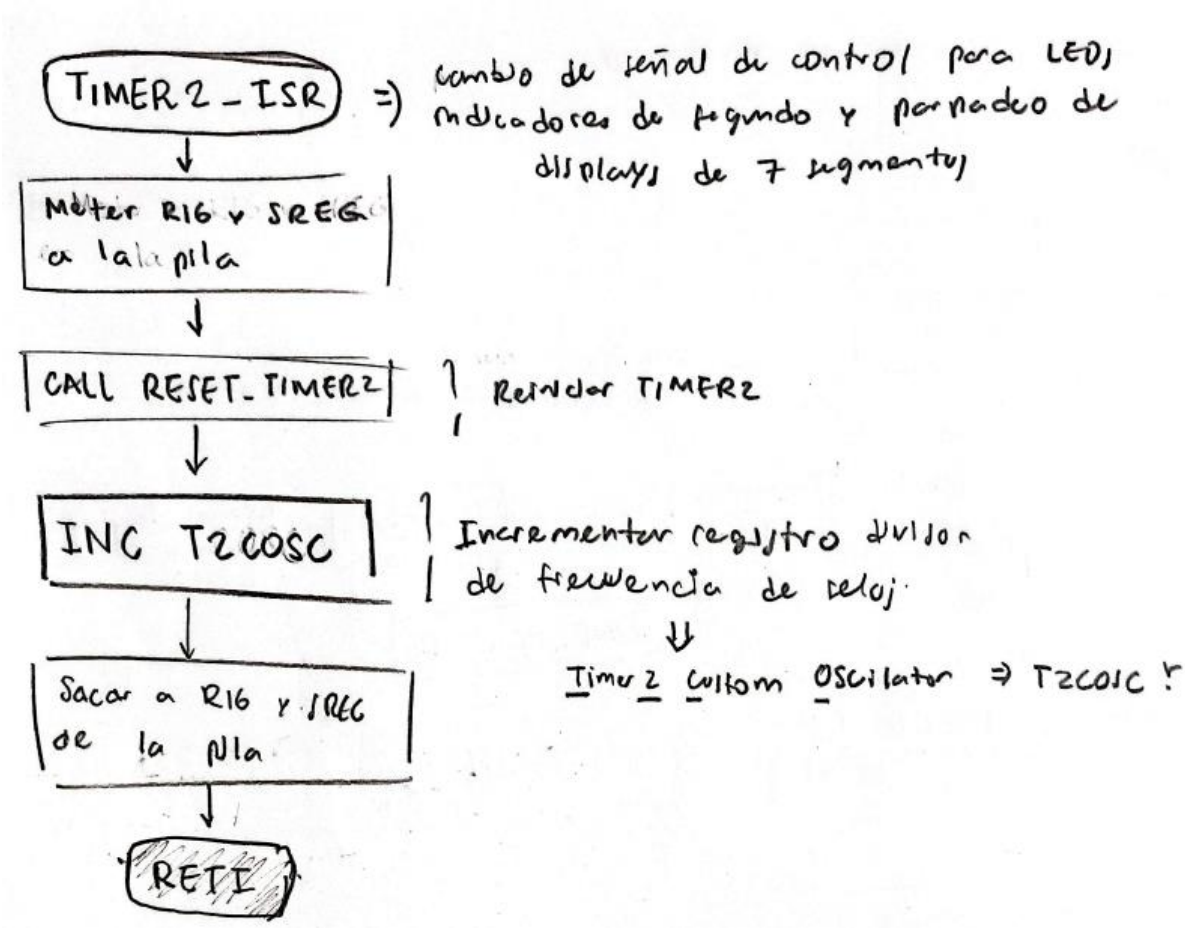
Figura 10 – Rutina de Interrupción de TIMER0



8.8. Señales Intermitentes (TIMER2_ISR)

En la rutina de interrupción de TIMER2 lo único que se hace es incrementar el registro T2COSC. Esto muestra lo conveniente que es utilizar contadores como divisores de frecuencia.

Figura 11 – Rutina de Interrupción de TIMER2

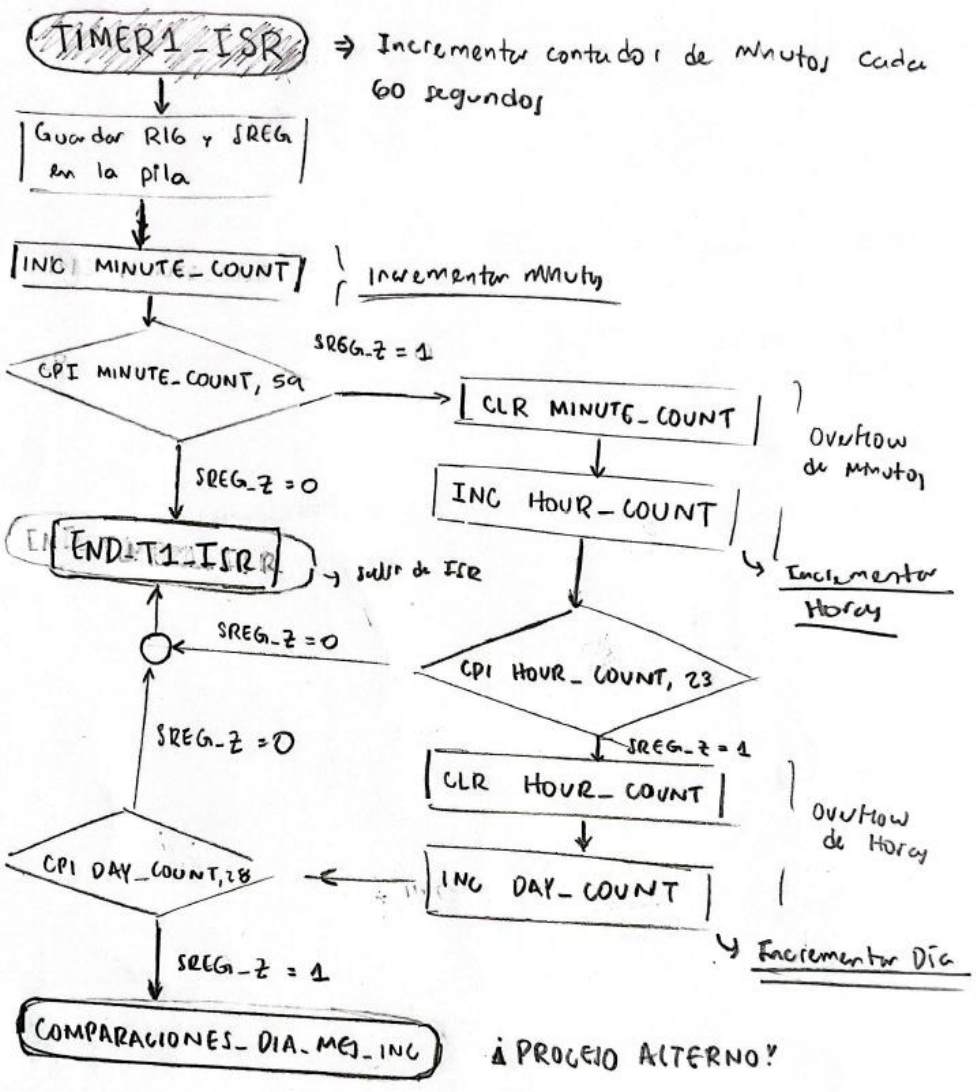


8.9. Incrementar Minutos automáticamente (TIMER1_ISR)

En la rutina de interrupción de TIMER1 lo primero que se hace es incrementar el contador de minutos MINUTE_COUNT. Posterior a ello se determina si es necesario hacer un Overflow comparando su valor con el valor inmediatamente anterior al valor máximo (60). Si el contador no es igual a este número, se hace un salto al final de la subrutina. Si es igual a 59, el contador de minutos se reinicia (CLR MINUTE_COUNT) y se incrementa en uno el contador de horas HOUR_COUNT.

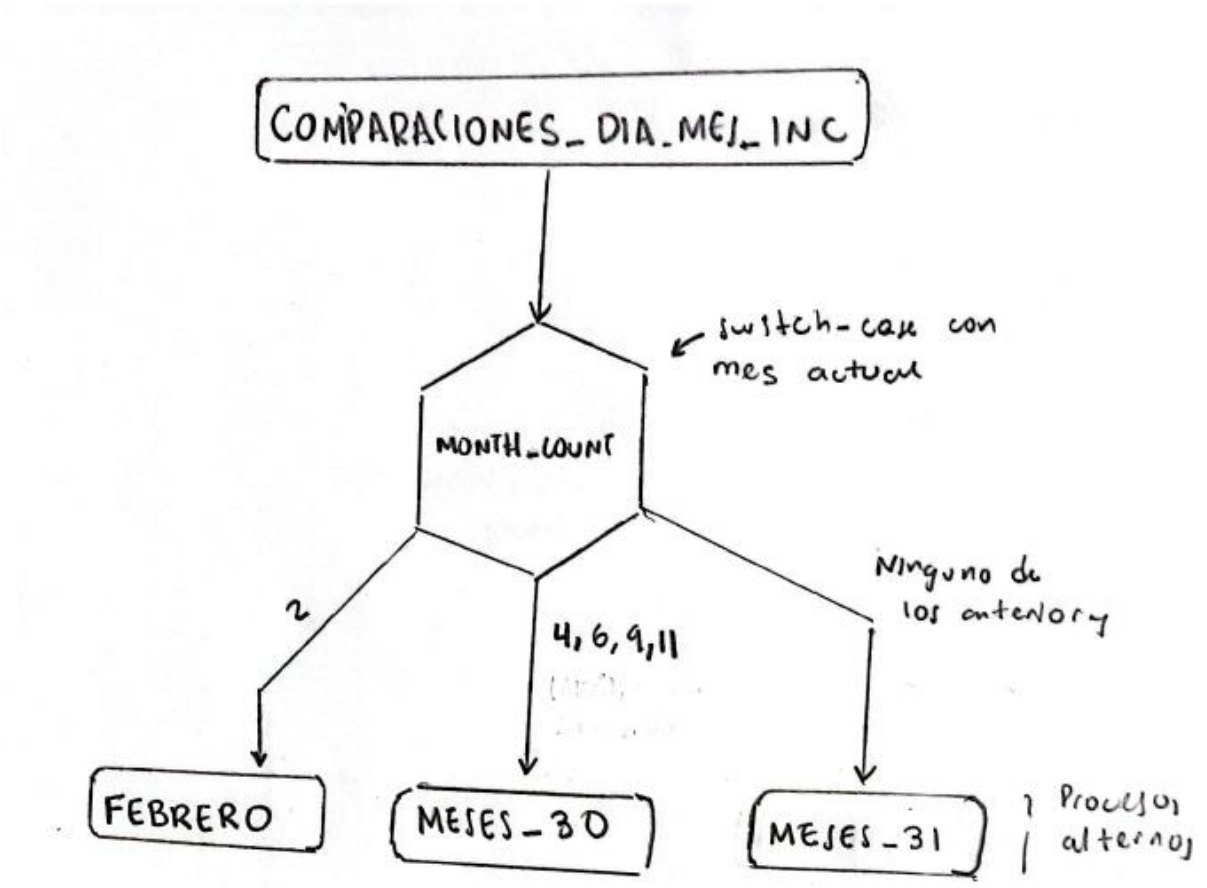
Si el contador de horas es igual a 23, el programa realiza un overflow y reinicia el contador HOUR_COUNT a 0 e incrementa en uno el contador de días DAY_COUNT. Nótese la similitud entre los procedimientos de incrementar y verificar si es necesario hacer overflows. Este esquema de ***incremento con overflow*** es una constante en todo el código y se discutirá con más detalle posteriormente.

Figura 12 – Incrementar minutos, horas y días



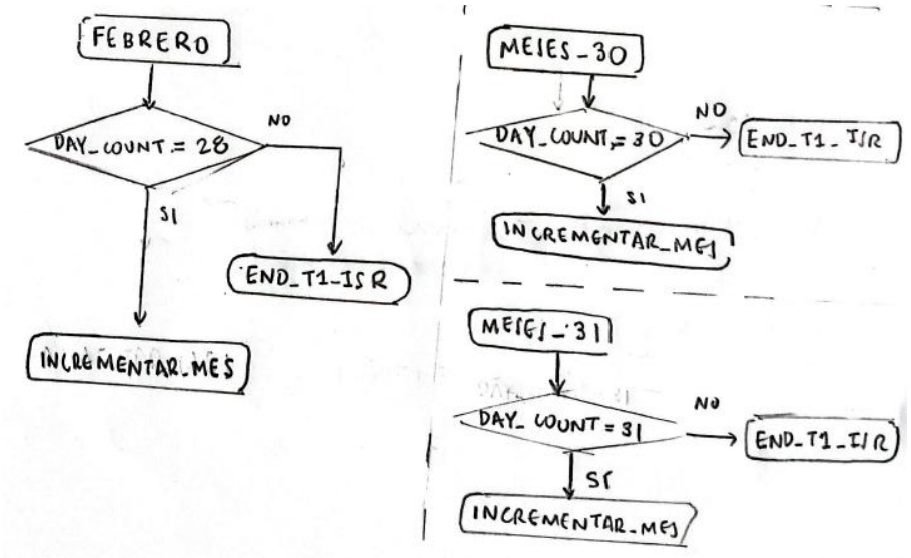
Para evitar hacer comparaciones innecesarias, se determina si el contador de días no es igual a 28, ya que este es el número mínimo en el que un mes termina (Febrero). De lo contrario, se realiza un salto a una subrutina especializada para determinar cómo proceder de acuerdo al mes actual (COMPARACIONES_DIA_MES_INC). Como se puede observar en la figura 13, la subrutina de comparación consiste en una secuencia de comparaciones anidadas (Un intento de un switch case) para determinar si el mes indicado por MONTH_COUNT es un mes de 30 días, de 31 días o es febrero (de 28 días).

Figura 13 – Comparaciones de contador de días de acuerdo al mes actual



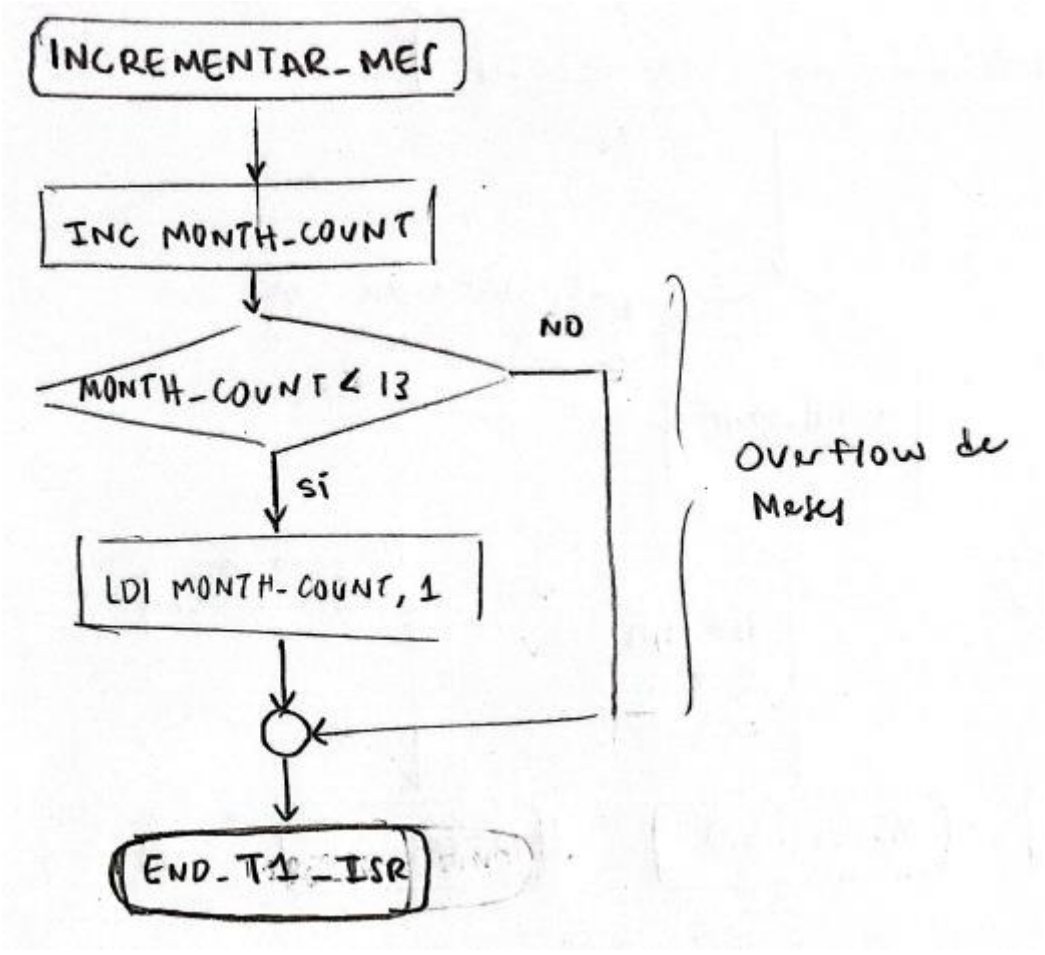
En las tres rutinas FEBRERO, MESES_30 y MESES_31 se compara la cantidad de días a la cantidad de días del mes en cuestión. Si el contador de días es igual a la cantidad de días en el mes, se incrementa el contador de meses MONTH_COUNT en uno. De lo contrario se salta directamente al final de la subrutina.

Figura 14 – Comparar el día con la cantidad de días en meses de 28, 30 y 31 días



Finalmente, si es necesario, se realiza un incremento del contador de meses MONTH_COUNT y se determina si es necesario hacer un overflow si su valor excede 12.

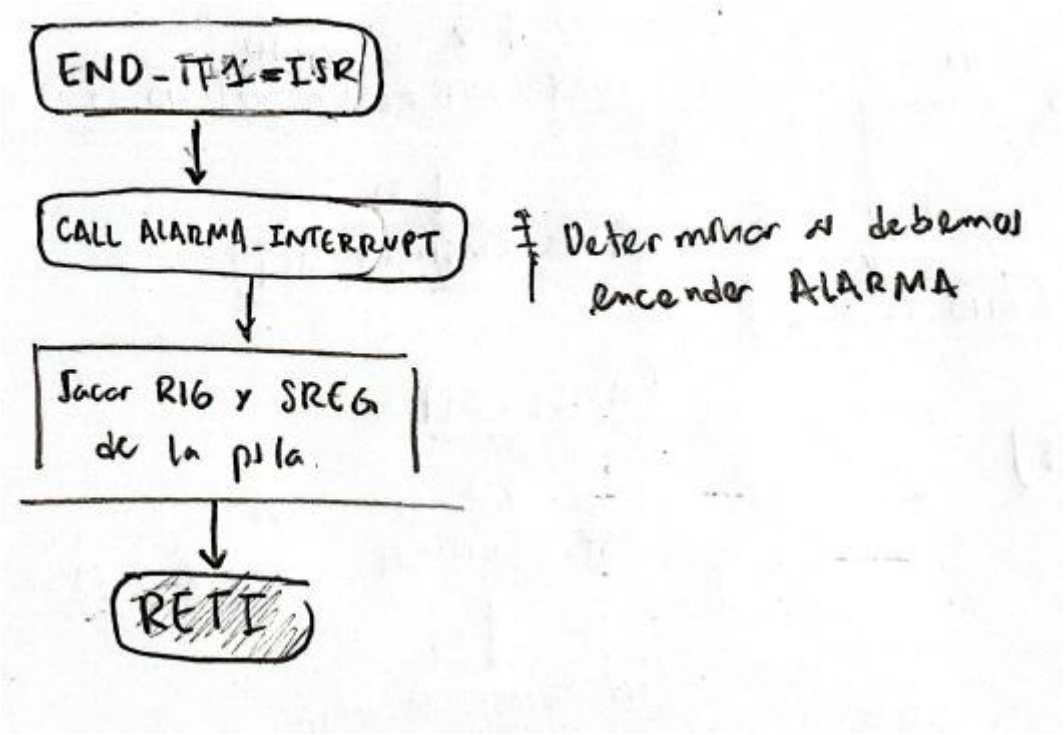
Figura 15 – Incremento de mes y overflow



Como se puede observar, la secuencia de incrementos se hace de forma anidada. Si un contador hace overflow se incrementa el contador siguiente y así sucesivamente hasta llegar al final. Por obvias razones, cuando el contador de orden más alto (En este caso MONTH_COUNT) hace overflow, todos los contadores se reincian a su valor inicial.

Finalmente, es importante mencionar que al final de la subrutina se realiza una llamada a una subrutina para determinar si es necesario encender la alarma. Esta llamada se colocó al final de la rutina de interrupción de TIMER1_ISR ya que es un bloque de código al que se salta en caso de no realizarse un overflow y al terminar el código. En todos los incrementos de contadores se puede saltar a esta subrutina.

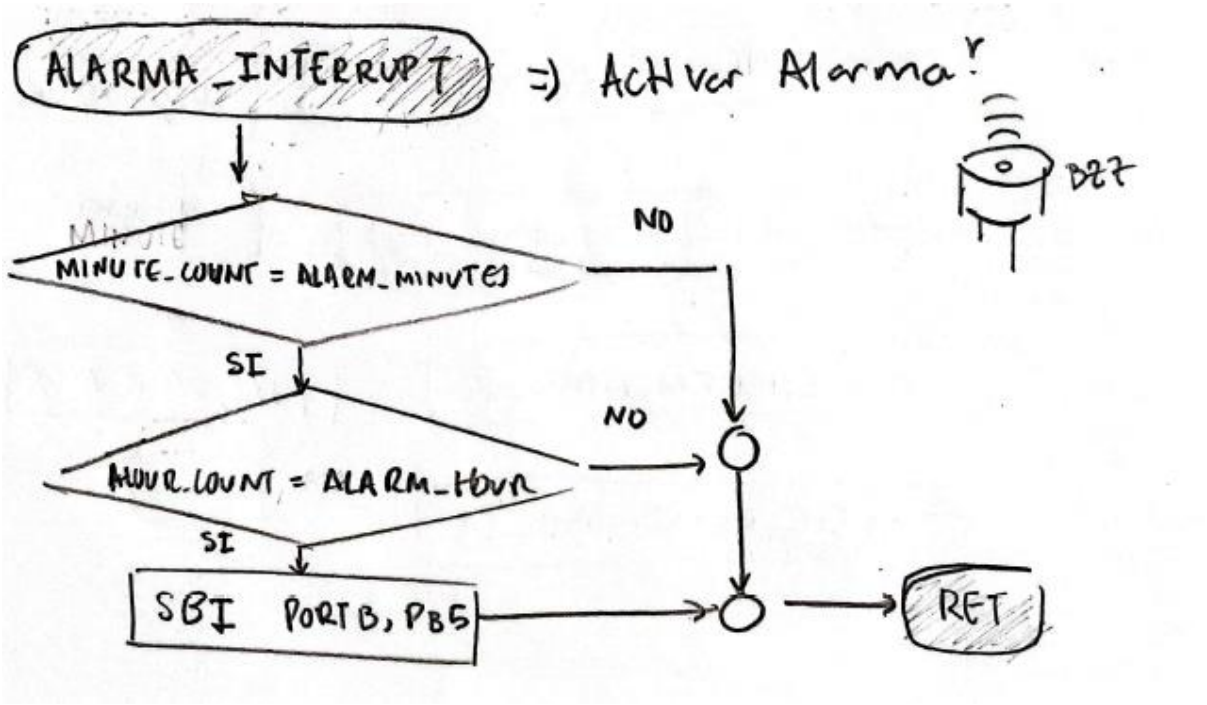
Figura 16 – Final de TIMER1_ISR



8.10. Activar Alarma (ALARMA_INTERRUPT)

Al final de la rutina de interrupción de TIMER1 se hace una llamada a esta subrutina para determinar si es necesario encender la alarma. La subrutina consiste de dos comparaciones de igualdad anidadas, donde se determina si el contador de minutos es igual a la cantidad de minutos configurada en la alarma (ALARM_MINUTES) y posteriormente si el contador de horas HOUR_COUNT es igual a la cantidad de horas configuradas en la alarma (ALARM_HOURS). Si son iguales, al final de TIMER1_ISR se encenderá el bit PB5 y hará sonar el buzzer conectado a ese pin.

Figura 17 – Encendido de Alarma

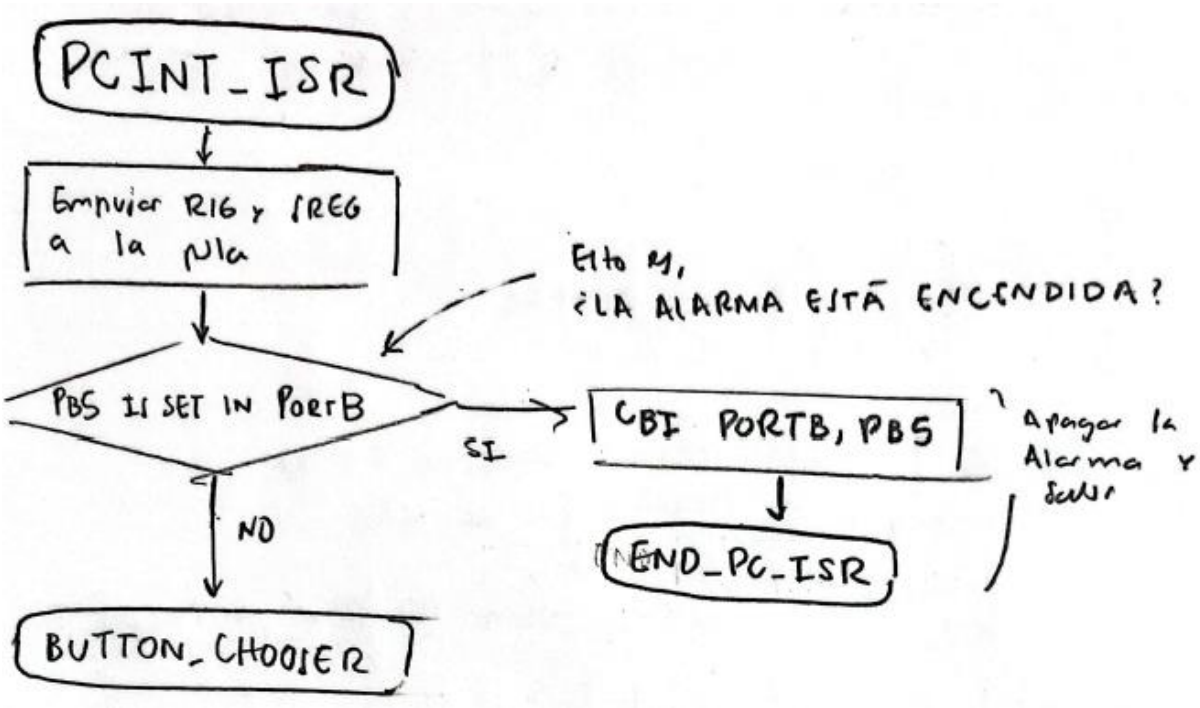


La llamada para activar la alarma sólo se realiza dentro de la rutina de interrupción de TIMER1_ISR. Por lo tanto, la alarma sólo se activa cuando los contadores son iguales ***después de que haya transcurrido un minuto***. Si el usuario configura la hora del indicador de horas para que sea igual a la hora configurada de la alarma, la alarma no sonará.

8.11. Inicio de Rutina de Interrupción por Pin-Change (PCINT_ISR)

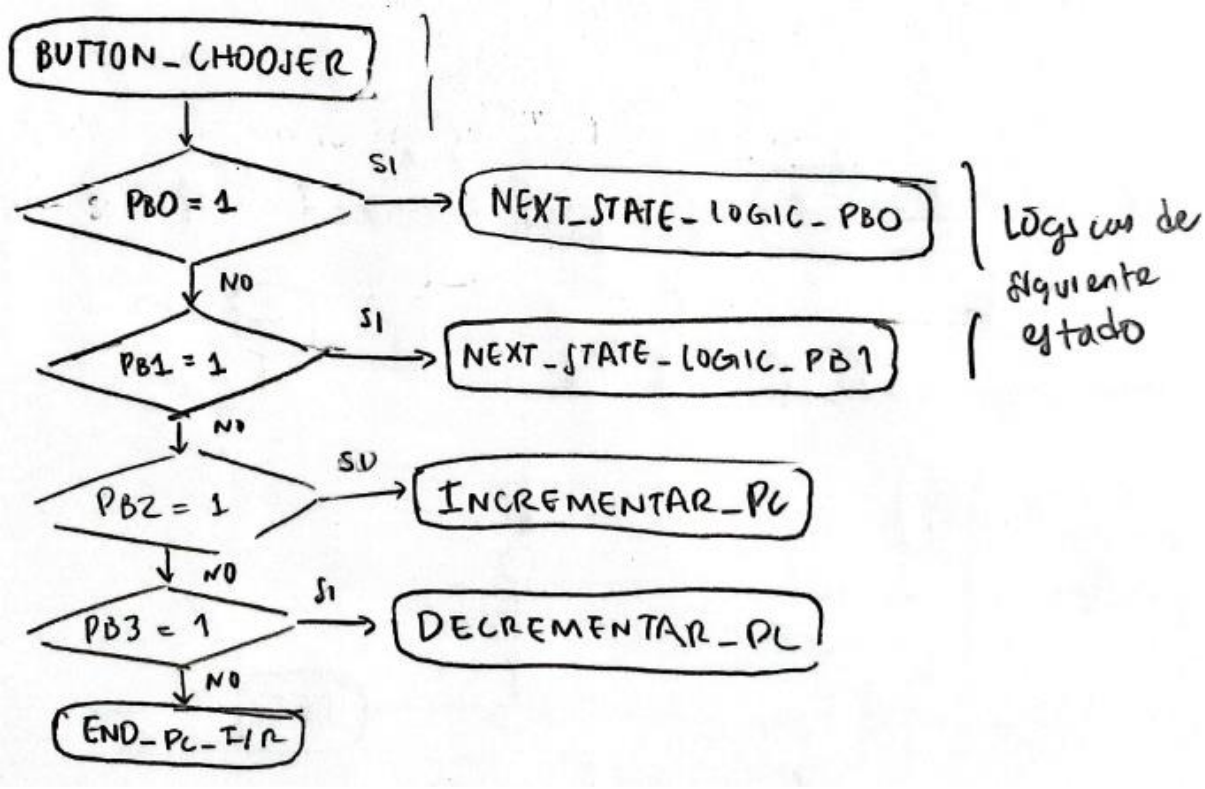
Si bien las interrupciones por cambio de pin en los pines PB0-PB3 se ramifican por cada botón conectado al Arduino, el inicio de la interrupción también cumple con una función especial. *Si la alarma está sonando, al presionar cualquier botón del reloj la alarma se apagará*. Esto es práctico, especialmente cuando el reloj se utiliza como un despertador (A la gente no le gusta que su alarma sea difícil de desactivar, más cuando hay varios botones de por medio).

Figura 18 – Inicio de PCINT_ISR



Después de haber realizado esta comparación, el programa salta a una secuencia de comparaciones anidadas para identificar cuál de los botones fue presionado. El uso de condicionales anidadas obliga a dar prioridad a un botón sobre otro. En caso se presionen varios botones al momento de realizar una interrupción, sólo se realizará la función asociada al botón con la comparación más alta.

Figura 19 – Identificación de botón presionado



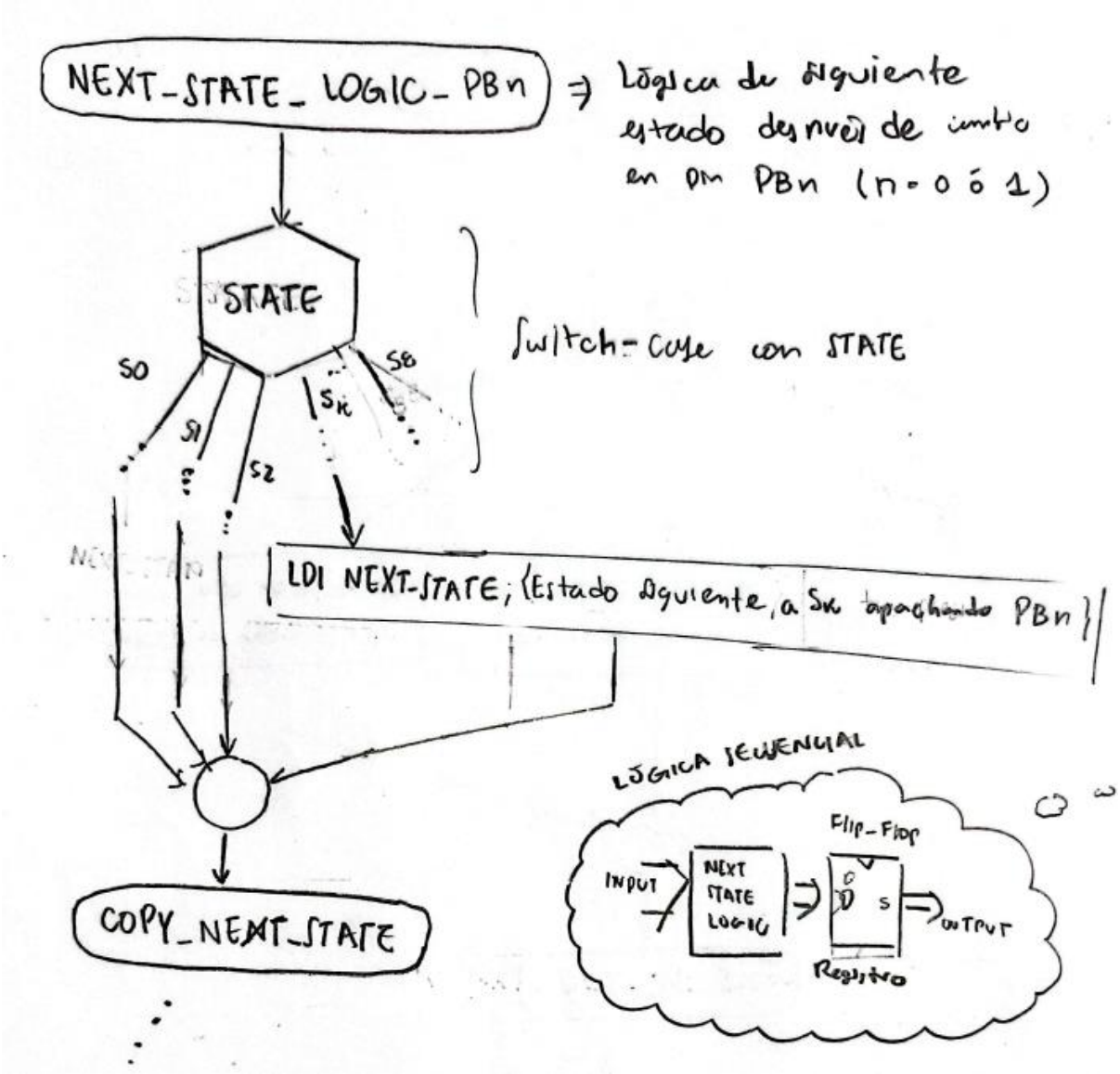
8.12. Lógica del Siguiente Estado (PB0 y PB1)

Si los botones presionados fueron aquellos conectados a los pines PB0 y PB1, el programa determina el siguiente modo operacional de acuerdo con el diagrama de transiciones mostrado anteriormente (Sección 4). Estas funciones se asemejan funcionalmente al primer bloque de una Máquina de Moore, donde a partir de una entrada lógica se determina el estado siguiente de una máquina de estados finitos.

Los botones en PB0 y PB1 tienen rutinas *separadas* para determinar el siguiente modo operacional. Al presionar PB0 se transiciona de un modo indicador (MostrarHora, MostrarFecha, ModoAlarma) al siguiente (*"Al siguiente modo indicador"*). En estos modos no es posible realizar cambios sobre los contadores. Además, si en el estado es posible cambiar alguno de los contadores, al presionar PB0 se regresa al modo indicador. Recuerde que cuando es posible cambiar uno de los contadores, esto se identificará haciendo los displays intermitentes. Por lo tanto, si alguno de los pares de displays parpadea, presionar PB0 hará que dejen de parpadear y *no se puedan editar*. Todos los *modos indicadores* funciona como un "Modo de candado" donde no se puede editar los valores de los contadores aún si se presionan los botones de incremento o decremento.

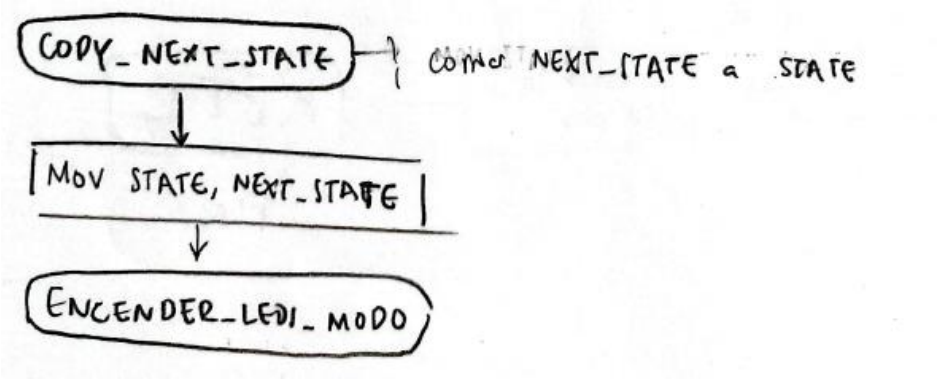
Al presionar PB1 se transiciona de un modo indicador a un *modo de cambio de contadores*. En estos modos el usuario puede incrementar o decrementar el valor del contador con los botones PB3 y PB4. Al presionar una vez el botón PB1 el usuario podrá modificar el contador bajo (Minutos y días). Al presionar otra vez, el usuario podrá modificar el contador alto (Horas y meses). Si vuelve a presionar otra vez, la máquina de estados finitos regresará de nuevo al modo indicador. En todos los casos el contador mostrado en el reloj seguirá siendo el mismo. Por ejemplo, si se encuentra en el modo indicador de horas y cambia a CambiarMinutos o CambiarHoras, el reloj seguirá mostrando la hora. Los displays del contador que se puede cambiar comenzarán a parpadear para indicar que son editables.

Figura 20 - Lógicas del Siguiente Estado



Después de determinar el siguiente estado, se copia este estado en el estado actual y se procede a encender el LED indicador de modo.

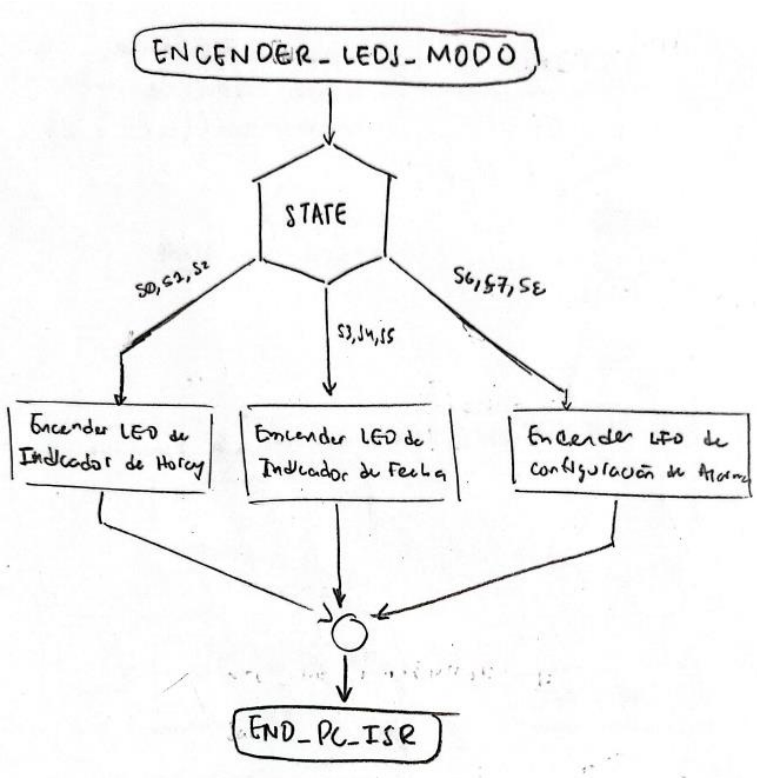
Figura 21 – Copiar estado



8.13. Encender LEDs Indicadores de Modo

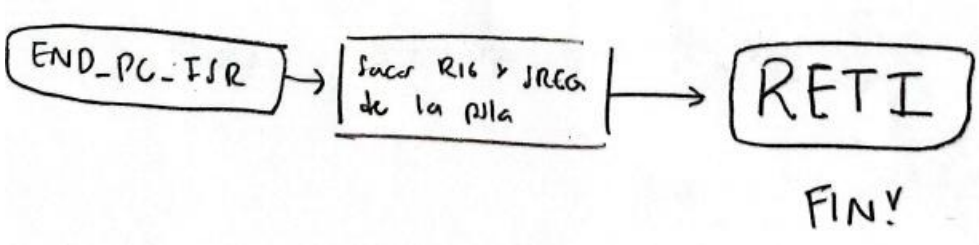
Después de haber determinado el estado, se realiza una serie de comparaciones sucesivas para determinar cuál de los tres LEDs indicadores de modo (Hora, Fecha, Alarma) debe encenderse. Posteriormente se procede a encender el LED en cuestión.

Figura 22 – LEDs Indicadores de Modo



Finalmente se muestra el final de la rutina de interrupción por Pin-Change.

Figura 23 – Fin de rutina de Interrupción por Cambio de Pin



8.14. Incrementar y Decrementar Contadores (PB3 y PB4)

Si el autómata se encuentra en un estado “Cambio” el usuario puede incrementar o decrementar el valor del contador cuyos displays parpadean usando los botones PB3 y PB4. En ambas las rutinas de *incremento y decremento* se inicia con una serie de comparaciones sucesivas para determinar la acción siguiente de acuerdo al valor de STATE. En la entrega final del proyecto, las rutinas de incrementar minutos, horas, días y meses son exactamente iguales a las implementadas para el TIMER1. Por la misma razón se reutilizaron en esta parte del código. Se crearon rutinas nuevas para incrementar los contadores de minutos y horas a los que se configura una alarma.

Figura 24 – Incremento con botón

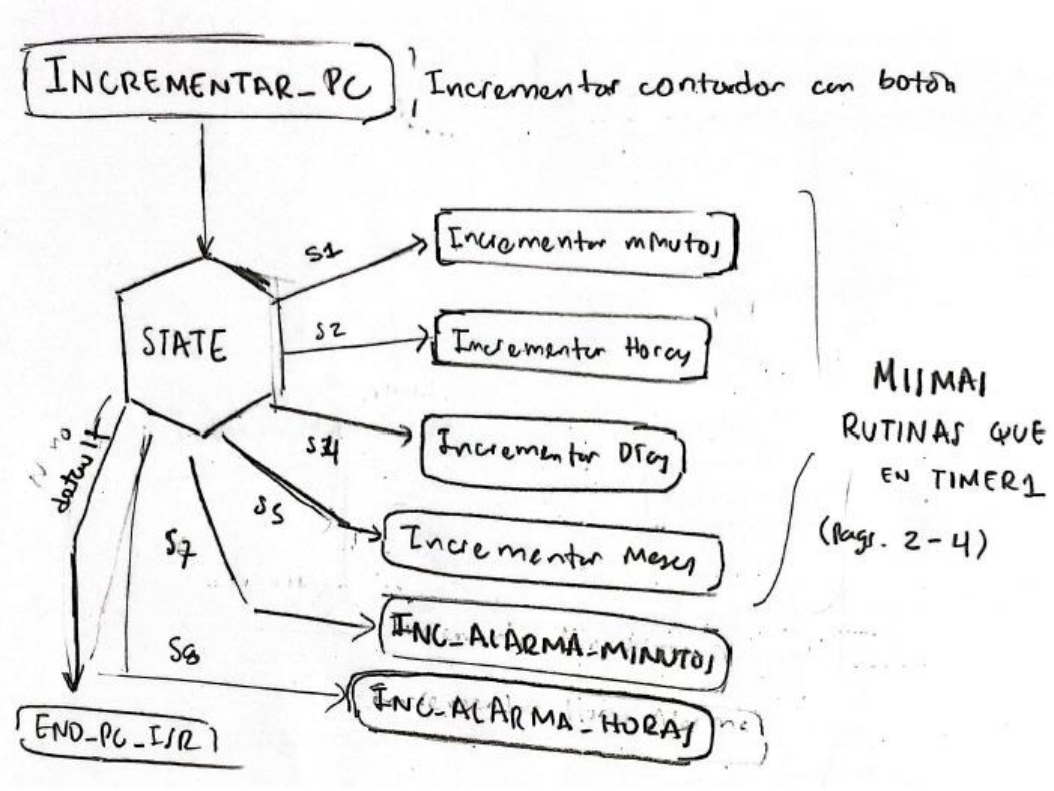


Figura 25 – Incrementar minutos de alarma

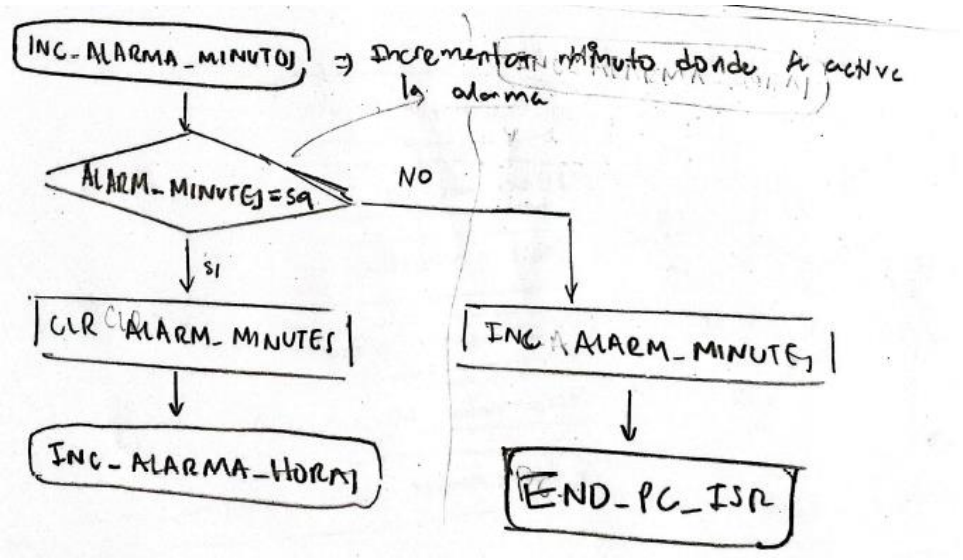


Figura 25 – Incrementar horas de alarma

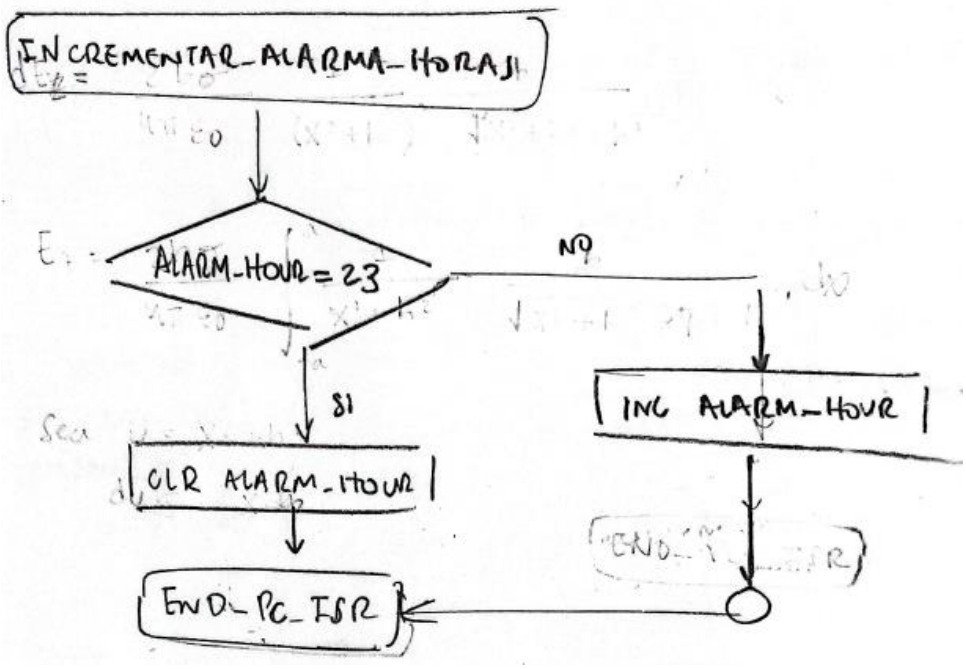
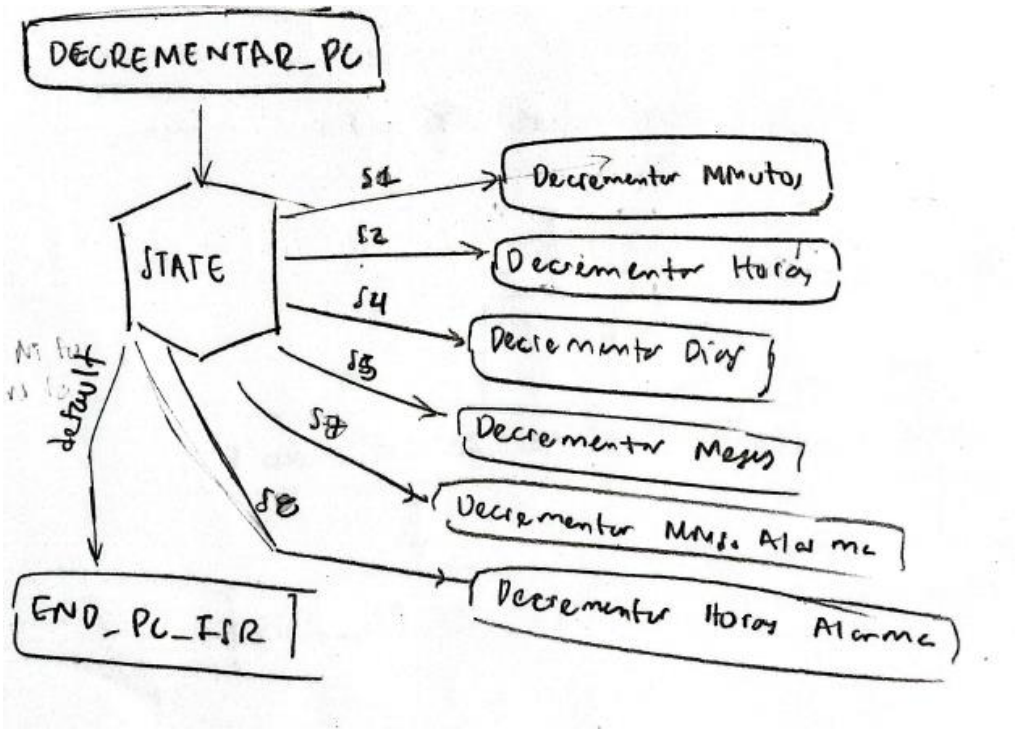


Figura 26 – Decremento con Botones



Estructuralmente las funciones de decremento son exactamente iguales a las de incremento con una lógica cambiada para permitir underflows. Para evitar hacer una discusión repetitiva se mostrará el esquema general seguido para realizar estas subrutinas.

8.15. Esquema General de Rutinas de Incremento y Decremento

Finalmente se presentan dos diagramas de flujo que representan a varios de los procedimientos realizados en el código. Estos proporcionan un procedimiento general para realizar operaciones de ***incremento con overflow*** y ***decremento con underflow***. Esto permitirá resumir aún mejor las partes restantes del código y evitar procedimientos repetitivos.

Figura 27 – Incremento con Overflow

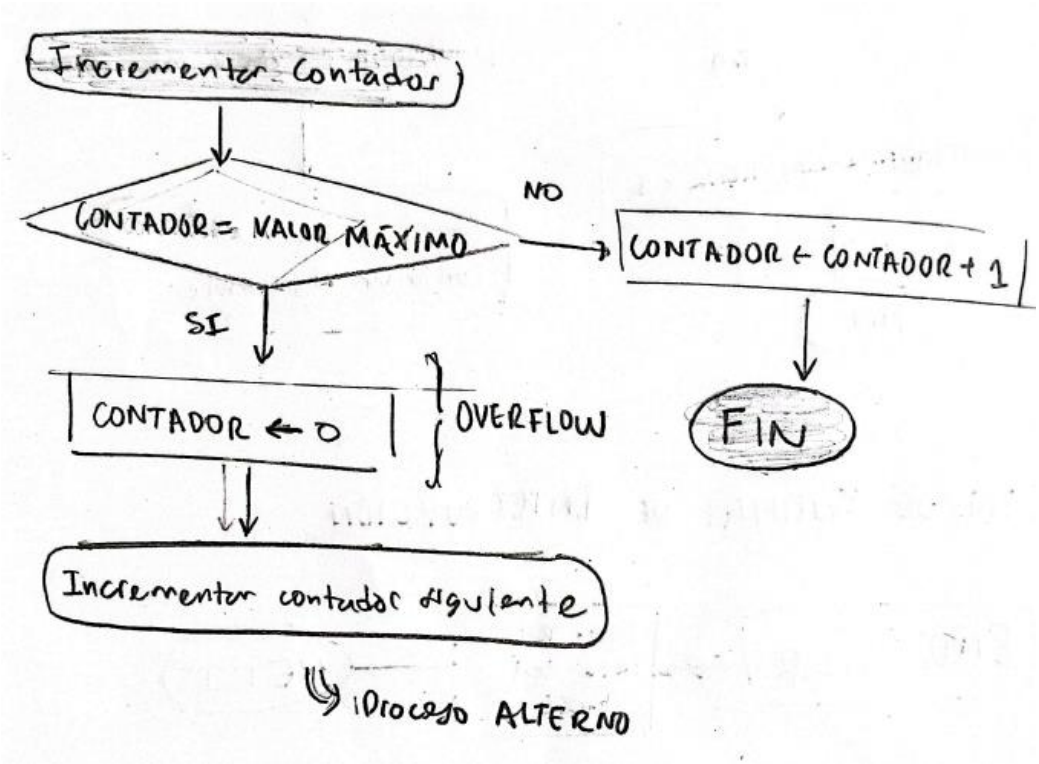
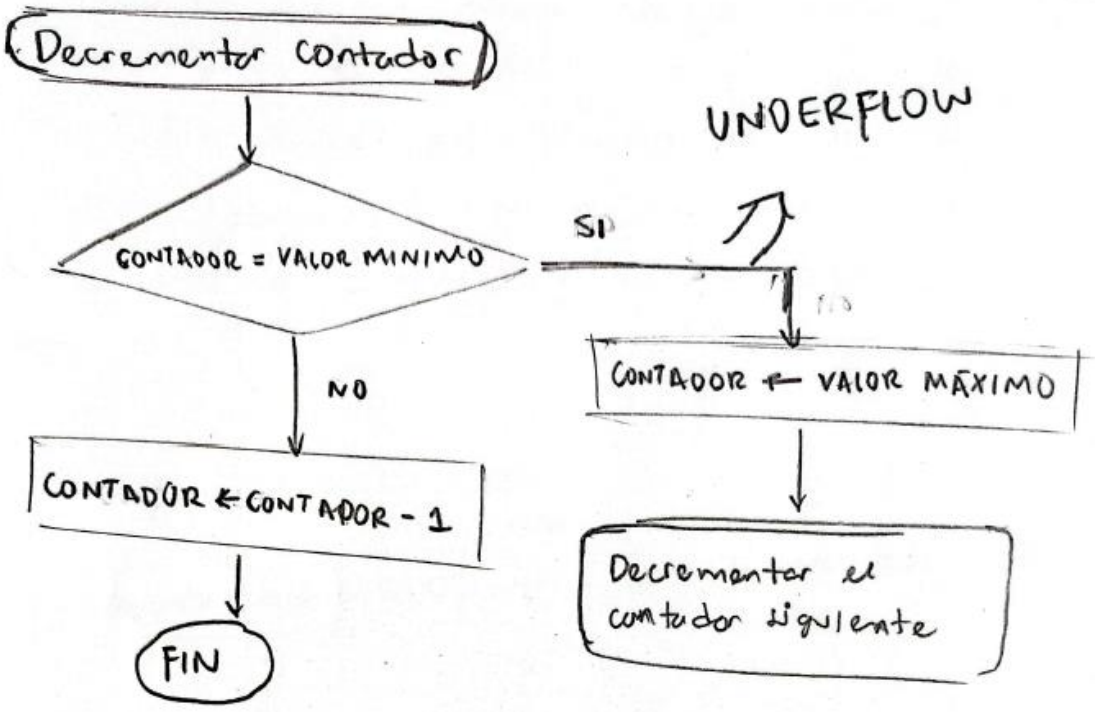


Figura 28 – Decremento con Underflow



Como se puede observar, en ambas funciones se compara el valor del contador con un valor extremo (Un máximo o un mínimo) y en base a esta comparación se determina si es necesario realizar un overflow o un underflow. Un overflow o un underflow implican el cambio del contador siguiente. Haga un breve vistazo a las funciones implementadas anteriormente y podrá identificar estos patrones en algunas secciones (Particularmente en el diagrama de flujo de TIMER1_ISR).

9. Recomendaciones y Aprendizajes

Una observación importante acerca de este programa es que *es funcional,* pero podría ocupar menos líneas de código si se utilizara un esquema más generalizado. Por cada modo operacional se definieron rutinas de incremento y decremento de contadores, las cuales podrían haberse realizado de otra manera siguiendo un paradigma "procedimental". Estos programas se asemejan a los que uno hace cuando apenas aprende a programar y desconoce lo que son las funciones.

Personalmente considero que elaborar diagramas de flujo me ayudó mucho a esclarecer lo que mi programa hace. Además, me guio en el proceso de encontrar un procedimiento más general para realizar incrementos con overflows y decrementos con underflows. Considero que con la realización de estas rutinas como funciones que admiten entradas y producen salidas el programa podría ocupar muchas menos líneas de código.

Con lo que he aprendido sobre programación de microcontroladores en clase y por mi cuenta he concluido que, si bien el lenguaje ensamblador puede ser tedioso y usa instrucciones sencillas, ofrece algunas virtudes que con otros lenguajes de más alto nivel no se podrían encontrar. Posiblemente la más importante es el *nivel de control que ofrece al programador sobre el uso de la memoria del microcontrolador*. La programación en C implica un proceso adicional de transformación de lenguaje de alto nivel a lenguaje máquina, denominado *compilación*. Esta transformación agrega instrucciones necesarias para traducirlo a un lenguaje comprensible para el microcontrolador.

Me gusta ver esto como la comparativa que se hace entre los carros mecánicos y automáticos. Lo que usualmente se hace en un carro mecánico de forma manual, un carro automático lo hace por el piloto. Ambos pueden usarse y uno no necesariamente sustituye al otro. Sin embargo, muchos prefieren a los carros mecánicos por el *control que les ofrece sobre las velocidades del vehículo. Que no sepamos usar un carro no necesariamente lo convierte en un mal carro.*

En conclusión, la programación en assembler no necesariamente implica usar más líneas de código. Con el enfoque correcto, podría ser mucho más eficiente. La

ineficiencia de un algoritmo rara vez es un problema del lenguaje utilizado, y generalmente se adjudica a las capacidades intelectuales del programador. En otras palabras, tengo que ponerme las pilas.

10. Referencias Bibliográficas

[1] Mazidi, Naimi y Naimi (2017) *The AVR Microcontroller and Embedded Systems Using Assembly and C: Using Arduino Uno and Atmel Studio*. NicerLand. Primera Edición.

Muy buen libro, personalmente le entiendo más que el manual de AVR.