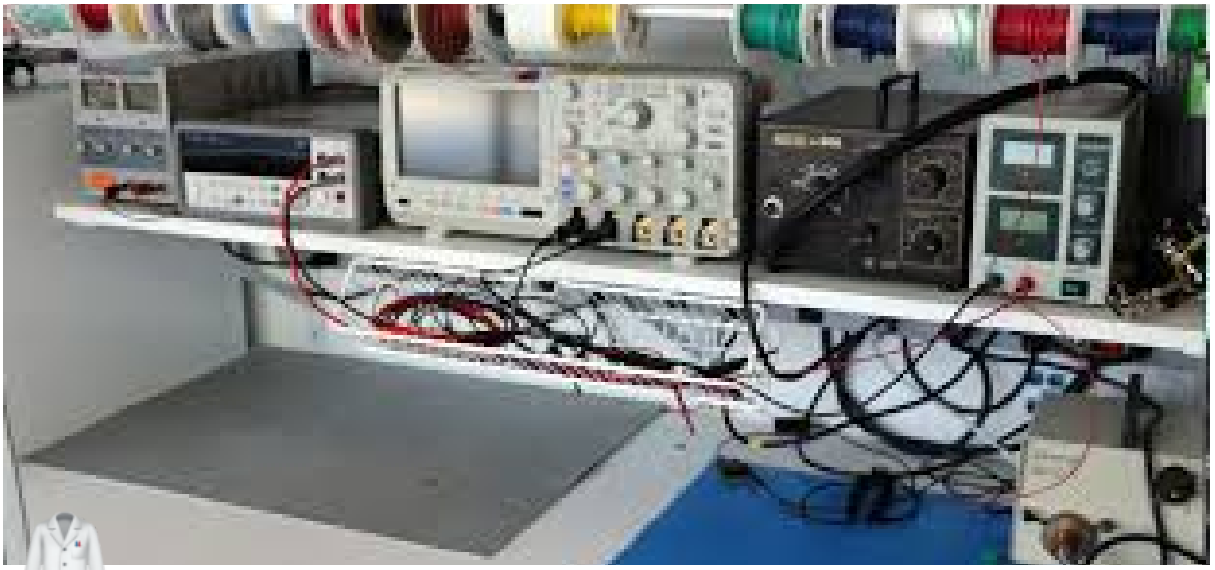


LABORATORIO 2

PARTE 1-1)

¿Por qué el programa llega a la etiqueta "fin" cuando no debería?

En el código actual, el problema es que el flujo de ejecución del programa principal llega a la etiqueta `fin` y entra en un bucle infinito en `loop2`. Esto ocurre debido a la forma en que está configurado el bucle `loop1` en el programa principal:

1. **Instrucción `brne loop1`:**

   La instrucción `brne` (Branch if Not Equal) verifica el flag de cero (Z) del registro de estado `SREG`. Si el flag Z está en 0 (es decir, el resultado de la última operación no fue igual a cero), el programa salta de vuelta a `loop1`. De lo contrario, continúa con la ejecución.

2. **Instrucción `ori r16, 0xFF`:**

   El `ORI` (OR immediate) realiza una operación `OR` entre `r16` y `0xFF`, lo que siempre da como resultado `0xFF`. Esta operación no afecta el flag Z, ya que el resultado nunca será 0.

   Como el flag Z no se establece correctamente en ningún momento dentro del bucle `loop1`, la condición `brne` no detecta el salto y, eventualmente, el flujo sale de `loop1`, llegando a `loop2` y luego a la etiqueta `fin`.

Resumen del problema:

- **Flag Z no está configurado correctamente**, por lo que el salto condicional `brne` no funciona como se espera.
- El programa continúa su ejecución más allá de `loop1`, llega a `loop2` y, finalmente, a `fin`.

ii) Modificar la rutina de interrupción del Timer0 para que no suceda más este problema.

El flujo de la interrupción se ha estructurado para realizar dos acciones clave:

1. **Encender un LED (PB4) cuando `r24` alcanza 20.**
2. **Apagar el LED y reiniciar `r24` cuando `r24` alcanza 125.**

Veamos paso a paso cómo está funcionando esta implementación:

Explicación del código:

1. **Contador de interrupciones (`r24`):**
   - Cada vez que ocurre una interrupción del Timer0, el registro `r24` se incrementa para contar el número de veces que la interrupción ha ocurrido. El contador tiene dos umbrales importantes:
     - **Cuando `r24` alcanza 20:** Se enciende un LED en el puerto PB4.
     - **Cuando `r24` alcanza 125:** Se apaga el LED en PB4 y se reinicia el contador `r24` a 0.
2. **Estructura de la interrupción (flujo condicional):**
   - **Primera parte:** Comprueba si `r24 == 20`. Si es igual a 20, enciende el LED en PB4.
   - **Segunda parte:** Comprueba si `r24 == 125`. Si es igual a 125, apaga el LED en PB4 y reinicia el contador `r24` a 0.
3. **Restaurar el estado de las banderas (`SREG`):**
   - Guardas el registro de estado `SREG` al inicio y lo restauras al final para preservar las banderas de estado.

LED QUE PARPADEA:

1)

Para hacer que el LED parpadee a una frecuencia de 1Hz (es decir, encender y apagar cada 1 segundo), debemos asegurarnos de que el Timer0 esté configurado correctamente para generar interrupciones con la suficiente frecuencia, y que el número de interrupciones acumuladas controle el parpadeo del LED.

Objetivo:

El LED debe **cambiar de estado cada segundo** (encenderse y apagarse), lo que equivale a una **frecuencia de parpadeo de 1Hz**. Esto significa que el LED debe encenderse durante **0.5 segundos** y apagarse durante **0.5 segundos**.

Enfoque:

1. **Calcular las interrupciones necesarias para 1 segundo:**
   - El Timer0 está configurado para generar una interrupción cada 8 ms (aproximadamente).
   - Podemos usar un contador (`r24`) para contar cuántas interrupciones han ocurrido. Cuando hayamos acumulado suficientes interrupciones para sumar 1 segundo, cambiamos el estado del LED.
2. **Cálculo del tiempo de interrupción:**Frecuencia del Timer=102416,000,000 =15625HzTiempo por interrupcio´n=15625125=0.008segundos(8ms)0.0081 =125interrupciones
   - **Frecuencia del reloj (Fclk):** 16 MHz.
   - **Prescaler:** 1024.
   - **Valor de `OCR0A`:** 124.

   Con esta configuración, la frecuencia del temporizador es:

   Frecuencia del Timer=16,000,0001024=15625 Hz\text{Frecuencia del Timer} = \frac{16,000,000}{1024} = 15625 \, \text{Hz}

   Esto significa que el Timer cuenta 15625 ciclos por segundo. Como se genera una interrupción cuando el contador alcanza 125, el tiempo entre interrupciones es:

   Tiempo por interrupcio´n=12515625=0.008 segundos (8 ms)\text{Tiempo por interrupción} = \frac{125}{15625} = 0.008 \, \text{segundos} \, (8 \, \text{ms})

   Entonces, necesitamos contar 125 interrupciones para obtener 1 segundo:

$$\frac{1}{0.008} = 125 \, \text{interrupciones}$$

CODIGO:

```
; Ejemplo_Interrupción_Timer0.asm modificado para agregar contador de segundos en el loop principal
; y estimar el porcentaje de tiempo ocupado por la interrupción
.ORG 0x0000
jmp     start           ; dirección de comienzo (vector de reset)
.ORG 0x001C
jmp     _tmr0_int ; salto atención a rutina de comparación A del timer 0
; ---------------------------------------------------------------------------
; Comienza el programa principal
start:
; Configuración de los puertos:
;     PB2 PB3 PB4 PB5 - son los LEDs del shield
ldi     r16,  0b00111101
out     DDRB,   r16             ; Configuro los 4 LEDs del shield como salidas
out     PORTB,  r16             ; Apago los LEDs
```

```
ldi     r16,    0b00000000
out     DDRC,   r16         ; 3 botones del shield como entradas
```

```
;---------------------------------------------------------------------------
; Configuración del Timer0 y su interrupción.
ldi     r16,  0b00000010
out     TCCR0A, r16             ; Modo CTC: cuenta hasta OCR0A y vuelve a cero
ldi     r16,  0b00000101
out     TCCR0B, r16             ; Prescaler = 1024
ldi     r16,  124
out     OCR0A,  r16             ; Divido el contador entre 125 (para interrupciones cada 8ms)
```

```
ldi     r16,  0b00000010
sts     TIMSK0, r16             ; Habilito la interrupción por comparación con
OCR0A
;-------------------------------------------------------------------------------
----
; Inicializo registros que voy a usar como contadores.
ldi     r24, 0x00           ; Contador para las interrupciones (125
interrupciones = 1 segundo)
ldi     r20, 0x00           ; Contador de segundos
ldi     r21,  30            ; Límite de 30 segundos
ldi     r22, 0x00           ; Registro que llevará el estado del LED
;-------------------------------------------------------------------------------
----
; Programa principal
comienzo:
sei                         ; Habilito las interrupciones globales (set
interrupt flag)
loop1:
; Reviso el contador de segundos (r20)
cpi     r20, 30             ; Comparo si han pasado 30 segundos
brne    skip_sound              ; Si no han pasado 30 segundos, no hago
nada
```

```
; Si han pasado 30 segundos, enciendo un LED o emito un son
ido
sbi     PINB,   3           ; Enciendo el LED en PB3 (se pu
ede cambiar por sonido)
ldi     r20,    0x00        ; Reinicio el contador de segun
dos
```

```
skip_sound:
nop                         ; No hacer nada (se puede añadir otro código
aquí)
rjmp    loop1               ; Loop infinito principal
;-------------------------------------------------------------------------------
----
; Rutina de interrupción del Timer0.
```

```
_tmr0_int:
in          r25,SREG                    ; Guardo SREG en r25 para
preservar estado de las banderas
inc         r24                         ; Incremento contador de
interrupciones

    cpi     r24,    125                 ; Comparo si han pasado
    125 interrupciones (1 segundo)
    brne    _tmr0_out                   ; Si no han pasado, sal
    ir de la rutina

    ldi     r24,    0x00                ; Reinicio el contador
    de interrupciones (r24)
    sbi     PINB,   2                   ; Hago un "toggle" en e
    l pin PB2 (LED)
    inc     r20                         ; Incremento el contado
    r de segundos

_tmr0_out:
out         SREG,   r25                 ; Restauro SREG desde r25 (con 1 ciclo
de reloj)
reti                                    ; Retorno de la interrupción del Timer0

;-------------------------------------------------------------------------------
----
```

4)

- Estimaremos cuánto tiempo de la ejecución total de la CPU se dedica a **atender la interrupción del Timer0**.
- Calcularemos el tiempo total que toma la rutina de interrupción y compararemos esto con el tiempo total de procesamiento.

Cálculo del tiempo en la interrupción:

1. **Duración de la interrupción**:
    - La rutina de interrupción tiene las siguientes instrucciones:
        - `in`, `sbi`, `inc`, `cpi`, `brne`, `ldi`, `out`, y `reti`.

- Estimamos que estas instrucciones toman aproximadamente **12 ciclos de reloj** en total.

2. **Tiempo por ciclo de reloj**:
   - El microcontrolador tiene un reloj de **16 MHz**, por lo que cada ciclo de reloj toma:
     $T_{ciclo} = \frac{1}{16,000,000} = 62.5 \, \text{ns}$.

3. **Tiempo total en la rutina de interrupción**:
   - El tiempo que toma la rutina es:
     $T_{interrupción} = 12 \times 62.5 \, \text{ns} = 750 \, \text{ns} = 0.75 \, \mu s$.

Cálculo del tiempo total de la CPU:

1. **Frecuencia de las interrupciones**:
   - El Timer0 genera **125 interrupciones por segundo** (cada 8 ms).

2. **Tiempo total dedicado a interrupciones**:
   - En total, en un segundo, la CPU pasa:
     $T_{total\ interrupciones} = 125 \times 0.75 \, \mu s = 93.75 \, \mu s$.

3. **Porcentaje del tiempo total**:
   - El porcentaje del tiempo total que la CPU pasa en interrupciones es:
     $\text{Porcentaje de tiempo} = \frac{93.75 \, \mu s}{1 \, \text{seg}} \times 100 = 0.009375\%$.
   - Esto significa que el tiempo que la CPU dedica a manejar la interrupción es muy pequeño, alrededor de **0.009375% del tiempo total** de ejecución.

EXPLICACION DEL CODIGO HASTA AHORA:

Programa principal:

1. Configuración de los puertos:

```
start:
```

- El programa comienza en la etiqueta `start`.

```
ldi   r16,  0b00111101
out   DDRB, r16         ; Configuro los 4 LEDs del shield c
omo salidas
out   PORTB, r16        ; Apago los LEDs
```

- `ldi r16, 0b00111101`: Carga en el registro `r16` el valor binario `0b00111101`. Este valor configura ciertos pines del puerto **B** (PB2, PB3, PB4 y PB5) como salidas para controlar los LEDs del shield.
    - `0b00111101` significa que:
        - PB0, PB1, PB6 y PB7 son entradas (0).
        - PB2, PB3, PB4 y PB5 son salidas (1).
- `out DDRB, r16`: Escribe el valor de `r16` en el registro **DDRB** (Data Direction Register for Port B), configurando los pines del puerto B como entradas o salidas según el valor cargado.
- `out PORTB, r16`: Apaga los LEDs escribiendo el mismo valor en el puerto **PORTB**, lo que efectivamente pone los pines PB2, PB3, PB4 y PB5 en bajo (apagados).

```
ldi   r16,  0b00000000
out   DDRC, r16         ; 3 botones del shield como entrada
s
```

- `ldi r16, 0b00000000`: Carga el valor `0b00000000` en `r16`, lo que configurará los pines del puerto **C** como entradas (todos 0).
- `out DDRC, r16`: Configura el puerto **C** (DDRC) para que todos sus pines sean entradas, probablemente para los botones del shield.

---

2. Configuración del Timer0 y su interrupción:

```
ldi   r16,  0b00000010
out   TCCR0A, r16       ; Modo CTC: cuenta hasta OCR0A y vu
elve a cero
```

- `ldi r16, 0b00000010`: Carga en `r16` el valor `0b00000010` para configurar el Timer0 en modo **CTC** (Clear Timer on Compare Match). Esto significa que el contador del temporizador contará hasta el valor en **OCR0A** y luego se reiniciará a cero.
- `out TCCR0A, r16`: Configura el registro de control **TCCR0A** (Timer/Counter Control Register A) para activar el modo CTC.

```
ldi   r16,  0b00000101
out   TCCR0B, r16       ; Prescaler = 1024
```

- `ldi r16, 0b00000101`: Carga en `r16` el valor `0b00000101`, que configura el prescaler del temporizador en 1024. El prescaler reduce la frecuencia del reloj del temporizador dividiéndolo por 1024, lo que permite contar más lentamente.
- `out TCCR0B, r16`: Escribe este valor en **TCCR0B**, configurando el prescaler.

```
ldi   r16, 124
out   OCR0A, r16        ; Divido el contador entre 125 (par
a interrupciones cada 8ms)
```

- `ldi r16, 124`: Carga el valor 124 en `r16`. Este valor se usará para configurar el **OCR0A** (Output Compare Register A). El valor 124 se elige porque el temporizador cuenta desde 0, y esto provoca una comparación después de 125 ticks (0 a 124), lo que genera una interrupción cada 8 ms.
- `out OCR0A, r16`: Escribe el valor 124 en el registro **OCR0A**.

```
ldi   r16,  0b00000010
sts   TIMSK0, r16       ; Habilito la interrupción por comp
aración con OCR0A
```

- `ldi r16, 0b00000010`: Carga `0b00000010` en `r16`, que habilita la interrupción por comparación con OCR0A (el temporizador genera una interrupción cuando alcanza el valor 124).
- `sts TIMSK0, r16`: Configura el registro **TIMSK0** (Timer Interrupt Mask Register) para habilitar la interrupción correspondiente.

---

3. Inicialización de registros como contadores:

```
ldi   r24,  0x00        ; Contador para las interrupciones
(125 interrupciones = 1 segundo)
ldi   r20,  0x00        ; Contador de segundos
ldi   r21,  30          ; Límite de 30 segundos
ldi   r22,  0x00        ; Registro que llevará el estado de
l LED
```

- Se inicializan varios registros:
  - `r24`: Contador de interrupciones (se incrementará en cada interrupción).
  - `r20`: Contador de segundos.
  - `r21`: Límite de 30 segundos.
  - `r22`: Registro para controlar el estado del LED.

---

4. Programa principal:

```
comienzo:
sei                         ; Habilito las interrupciones g
lobales (set interrupt flag)
```

- `sei`: Habilita las interrupciones globales. A partir de aquí, el programa responde a las interrupciones cuando ocurren.

```
loop1:
cpi   r20, 30               ; Comparo si han pasado 30 segu
ndos
brne  skip_sound            ; Si no han pasado 30 segundos,
no hago nada
```

- `cpi r20, 30`: Compara el valor de `r20` (contador de segundos) con 30.
- `brne skip_sound`: Si no han pasado 30 segundos (`r20` ≠ 30), salta a la etiqueta `skip_sound`.

```
sbi   PINB, 3               ; Enciendo el LED en PB3 (se pu
ede cambiar por sonido)
ldi   r20, 0x00             ; Reinicio el contador de segun
dos
```

- `sbi PINB, 3`: Si han pasado 30 segundos, hace un **toggle** en el LED conectado al pin PB3.
- `ldi r20, 0x00`: Reinicia el contador de segundos (`r20`).

```
skip_sound:
nop                         ; No hacer nada (se puede añadi
```

```
r otro código aquí)
rjmp  loop1                 ; Loop infinito principal
```

- `nop`: No hace nada (operación vacía). Puede reemplazarse con otras instrucciones si es necesario.
- `rjmp loop1`: Salta a la etiqueta `loop1` para repetir el ciclo infinitamente.

Rutina de interrupción del Timer0:

```
_tmr0_int:
in    r25,SREG              ; Guardo SREG en r25 para prese
rvar estado de las banderas
inc   r24                   ; Incremento contador de interr
upciones
```

- `in r25, SREG`: Guarda el registro de estado **SREG** en `r25` para preservar las banderas (esto es necesario porque las interrupciones pueden afectar las banderas del procesador).
- `inc r24`: Incrementa el contador de interrupciones (`r24`).

```
cpi   r24, 125              ; Comparo si han pasado 125 int
errupciones (1 segundo)
brne  _tmr0_out             ; Si no han pasado, salir de la
rutina
```

- `cpi r24, 125`: Compara si el contador de interrupciones ha alcanzado 125 (equivalente a 1 segundo).
- `brne _tmr0_out`: Si no ha alcanzado 125, salta a `_tmr0_out` para salir de la interrupción.

```
ldi   r24, 0x00             ; Reinicio el contador de inter
```

```
rupciones (r24)
sbi   PINB, 2               ; Hago un "toggle" en el pin PB
2 (LED)
inc   r20                   ; Incremento el contador de seg
undos
```

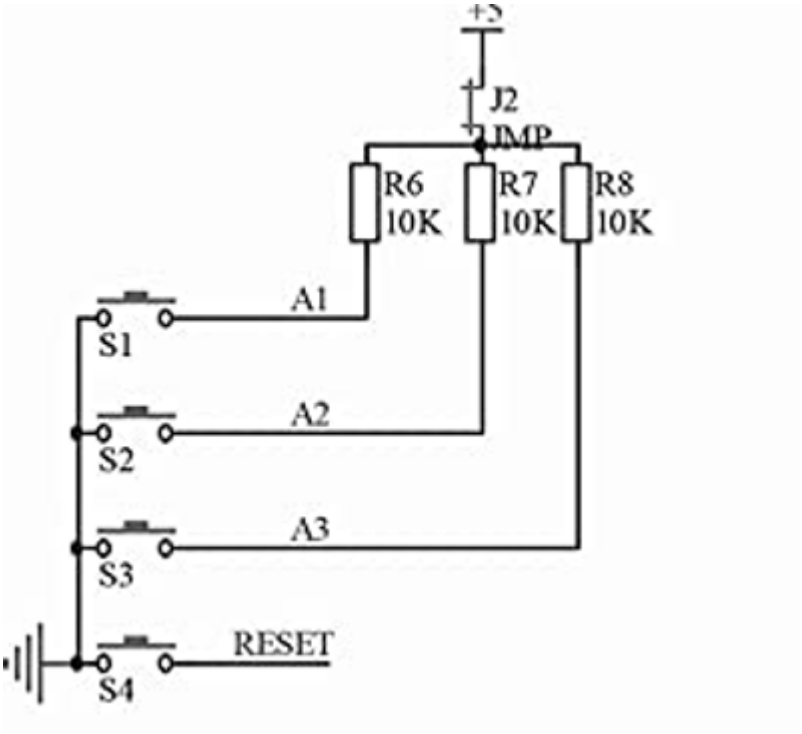
- `ldi r24, 0x00`: Reinicia el contador de interrupciones.
- `sbi PINB, 2`: Hace un **toggle** en el LED conectado al pin PB2.
- `inc r20`: Incrementa el contador de segundos.

```
_tmr0_out:
out   SREG, r25             ; Restauro SREG desde r25 (con
1 ciclo de reloj)
reti                        ; Retorno de la interrupción de
l Timer0
```

- `out SREG, r25`: Restaura el valor original del registro de estado **SREG**.
- `reti`: Finaliza la rutina

PARTE 2

1)En esta imagen se encuentran los botones del modulo.



Los pines 1,2,3 se encuentran en el PUERTO C 1,2,3.

INTERRUPCIONES

CODIGO FINAL DEL LAB 2