

# Laboratorio de Principios de Mecatrónica

## Práctica 2: Ensamblador, Interrupciones y Temporizadores

Rebeca Baños García, Leonardo García Osuna

**Resumen**—En esta práctica se aprendió a programar un microcontrolador utilizando instrucciones de ensamblador, después se compararon los resultados obtenidos en el código ensamblador así como en el código en C. Por último se aprendió a utilizar las interrupciones y a configurar temporizadores del tipo normal y CTC para poder utilizarlos.

### 1. INTRODUCCIÓN

Aprender como funciona el código de ensamblador y la manera en la que trabaja es importante para conocer el funcionamiento interno del AVR. El lenguaje de ensamblador es complicado, por eso el comparar los resultados con los obtenidos en lenguaje C nos ayuda a verificar los errores que se pueden tener y evitar cometerlos. Además el IDE de Arduino permite introducir código ensamblador dentro de C++, muy útil para optimizar partes de código para una programa cuyas funciones ya sea que no existan en C++ o sea más eficiente programarlas directamente en ensamblador.

Una parte importante de la práctica también es identificar los pines de entrada y de salida para poder tener un resultado visual a través de LEDs y ver que estén parpadeando a la frecuencia deseada.

El uso de interrupciones y temporizadores es muy importante ya que se usan internamente todo el tiempo, además se pueden utilizar para operaciones externas, por lo que se tuvo que aprender a configurarlos para el funcionamiento que se desea.

### 2. MARCO TEÓRICO

Los dispositivos utilizados en la práctica fueron los siguientes:

- **Código ensamblador:** Para programar en código ensamblador se necesitan tener las instrucciones básicas para las operaciones que se desean ejecutar. En esta práctica para el cálculo del determinante utilizamos las siguientes instrucciones básicas:

- **LDI:** Instrucción que carga de inmediato un valor constante de 8 bits a un registro.
- **MUL:** Multiplica dos números sin signo de 8 bits guardados en un registro y el resultado lo guarda en el primer registro ingresado.
- **MOV:** Copia el valor de un registro a otro registro.

- **SUB:** Resta dos valores dentro de dos registros y guarda el resultado en el primer registro ingresado.
- **SUBC:** Resta dos registros con carry y guarda el valor en el primer registro ingresado tomando en cuenta la bandera de carry.

Para el código en ensamblador del cálculo del promedio, utilizamos las siguientes instrucciones básicas:

- **ADD:** Suma el valor de dos registros sin tomar en cuenta la bandera carry y el resultado lo escribe en el primer registro ingresado.
- **INC:** Suma un 1 al valor del registro ingresado y lo guarda en el mismo registro.
- **BRLT:** Salto condicionado. Este salto verifica la bandera S (sign) y salta a la instrucción indicada solo si la bandera indica que el primer registro es menor al segundo registro en la instrucción anterior a esta.
- **JMP:** Salta a una dirección dentro de la memoria del programa para ejecutar la operación en guardada en esta dirección.

- **Entradas y salidas:** Para establecer los valores de entrada y salida desde el ensamblador se necesitan usar las siguientes instrucciones:

- **SBI:** Escribe un 1 en un bit específico de algún registro I/O.
- **CBI:** Escribe un 0 en un bit específico de algún registro I/O.

Con estas instrucciones se escribe el código de configuración para los puertos y pines requeridos.

```
void setup () {  
    // Data Direction Register B: Inputs  
    // 0-6, Output 7  
    DDRB = DDRB | B10000000;  
}  
  
void loop () {  
    asm volatile (  
        "sbi %0, %d \n\t"  
        : : "I" (_SFR_IO_ADDR(PORTB)), "I" (  
            PORTB7));  
}
```

Instrucciones para configurar el puerto (ensamblador)

- Interrupciones:** Para utilizar alguna interrupción en particular se necesita deshabilitar globalmente las interrupciones, después establecer como entrada el puerto correspondiente, activar el bit de interrupción en el puerto correspondiente, establecer el tipo de interrupción, habilitar la interrupción que se busca en el registro de máscaras de interrupciones externas. Posteriormente, se habilitan globalmente las interrupciones y se declara la función ISR. Dentro de esta función se escribe lo que se desea realizar una vez activado el interruptor.

```
void setup() {
    cli();
    DDRD &= ~(1 << DDD1);
    PORTD |= (1 << PORTD1);
    EICRA |= (1 << ISC10);
    EIMSK |= (1 << INT1);
    sei();
}
ISR(INT1_vect){
    // Código a ejecutar al activarse la
    // interrupción
}
```

Instrucciones para configurar instrucciones (ensamblador)

- Temporizadores:** El microcontrolador cuenta con temporizadores internos que cumplen una condición cada ciclo de reloj. El ciclo de reloj predeterminado es de 16MHz, por lo que si se quiere modificar la frecuencia del ciclo del reloj se utilizan pre-escalares. La señal de reloj y el prescalar son alimentadas a un divisor de frecuencia, y su salida es la que utiliza el temporizador. Para ambos modos de temporizadores se tienen que configurar en código en ensamblador en Arduino. Para los temporizadores existen dos métodos:

- **Modo Normal:** En el modo normal el registro del temporizador inicia en un valor preestablecido y va incrementando 1 cada ciclo de reloj hasta que este se desbordé, es decir, sobrepasenlos 255 que soporta el registro.

```
void setup() {
    Serial.begin(9600);
    DDRB = DDRB | B10000000;
    cli();
    TCCR1B= 0;
    TCCR1A=0;
    TCCR1B |= (1 << CS12);
    TCNT1= 3036;
    TIMSK1 |= (1 << TOIE1);
    sei();
}

void loop() {
```

Configuración del temporizador en modo normal

- **Modo CTC:** En el modo CTC, el registro del temporizador se va incrementando hasta ser igual que el valor del registro OCR. Una vez que llegué a ser igual, se reinicia y vuelve a contar a partir de 0.

```
void setup() {
    Serial.begin(9600);
    DDRB = DDRB | B10000000;
    cli();
    TCCR1B= 0; TCCR1A=0;
    TCCR1B |= B00001101
    OCR1A= 0x0F42;
    TIFR1 |= (1 << OCF1A);
    TIMSK1 |= (1 << OCIE1A);
    sei();
}
```

Configuración del temporizador 0 en modo CTC

### 3. DESARROLLO

Para realizar el código en C y en ensamblador, primero programamos en C para que la conversión a dicho ensamblador fuera más sencilla conociendo las instrucciones necesarias para implementar el algoritmo de calcular el determinante y el promedio.

Para trabajar las entradas y salidas del AVR en Arduino, primero elegimos el pin digital con el cual trabajamos, ya sea entrada o salida, y verificamos en el mapeo de pines de Arduino el número de puerto o pin para configurarlo en el código de Arduino. Para configurarlo utilizamos el código del marco teórico. Posteriormente, para agregar el botón, se alambró el botón en la protoboard y se conectó a los pines necesarios para recibir el efecto del botón y que reaccionara el LED indicado.

Para las interrupciones utilizamos el código ensamblador para configurar los pines como lo hicimos con las entradas y salidas, posteriormente dentro de la instrucción ISR ingresamos lo que se quería realizar una vez la interrupción sea activada. En la primer parte de esta sección escribimos el código necesario para hacer parpadear el LED a 2Hz con la función delay. Para la segunda parte de esta sección, conectamos el botón a la protoboard como lo hicimos con la sección de entradas y salidas y dentro del ISR escribimos el código para incrementar un contador cada que se presionara el botón. Este programa tiene la limitación que solamente llamando al compilador de Arduino se puede ver que en verdad el contador se va incrementando, pero fuera de eso no se puede tener una respuesta física. Para reemplazarlo con el LED infrarrojo y el fotodiodo solamente se cambió el botón en el alambrado y se reemplazó por el infrarrojo.

Para la sección de temporizadores, primeramente se utilizó el código del marco teórico para configurar el temporizador en modo normal. Posteriormente dentro del ISR se escribió el código para hacer parpadear un LED a 2Hz. Para la segunda parte de la práctica, utilizamos el código de temporizador en modo CTC y activamos los registros necesarios para que se hiciera la comparación, luego calculamos la frecuencia necesaria para que el LED parpadeara a 4Hz.

Para el semáforo, utilizamos el mismo código que para el temporizador en modo CTC y dentro del ISR creamos las diferentes condiciones que tenía que cumplir nuestro alambrado para que funcionara de acuerdo al problema de la práctica.

### 4. RESULTADOS

Nuestros resultados fueron positivos en cuanto a las primeras secciones de la práctica. Al convertir los códigos del

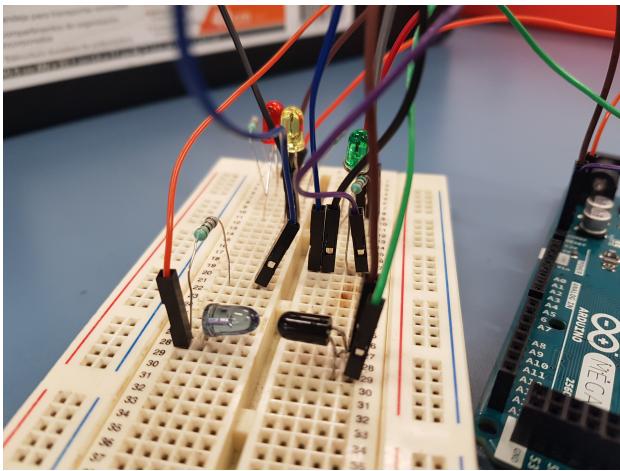


Figura 1. Interrupciones con LEDs infrarrojos.

determinante y del promedio en ensamblador fue mucho más fácil comprobar que los resultados compilados funcionaran y fueran los correctos ya que dominamos mucho mejor el código en C y la conversión fue sencilla con las instrucciones bien definidas. Para las partes siguientes de la práctica los códigos proporcionados fueron de gran ayuda, ya que nos proporcionaba una mejor idea de como reacciona el programa internamente.

La parte de entradas y salidas fue relativamente fácil ya que sólo se copió el código proporcionado y se trabajó solo con el LED del Arduino y al agregarle el botón solo se cableó en la protoboard los pines necesarios para que detectara la reacción del botón.

En la parte de interrupciones tuvimos complicaciones al conectar los LEDs infrarrojos, ya que al inicio nuestro LED emisor no encendía—pudimos constatarlo con una cámara sensible a luz IR—, por lo que no se veía el incremento en el contador. Al reemplazar dicho LED emisor —con la certeza de funcionamiento— buscamos la manera en que el receptor fuera sensible por lo que tuvimos que colocar los LEDs de tal manera que estuvieran a una distancia lo suficientemente buena para que al atravesar una hoja entre los dos, se detectara el cambio y así se incrementara el contador.

Para la parte de temporizadores tuvimos problema con el código del temporizador tipo CTC, ya que en el registro de configuración del temporizador sabíamos que teníamos que activar el bit que indicaba el WGM12, pero al correr el programa no compilaba, así que cambiamos esa instrucción por los valores necesarios para llenar el registro de configuración del temporizador en binario. El error detectado en el código sugerido, los bits CS1x se hicieron cero al indicar `TCCR1B=0;` y sólo se activaba de nueva cuenta el WGM12, sin embargo el timer permanece apagado dado que `CS10=CS11=CS12=0;` por lo tanto, para que funcione adecuadamente hicimos `TCCR1B = B00001101;`

## 5. CONCLUSIONES

- Rebeca:** La práctica fue muy interesante ya que practiqué la manera en la que escribo código en ensamblador. Además fue de gran ayuda para entender y reforzar lo visto en teoría de acuerdo a

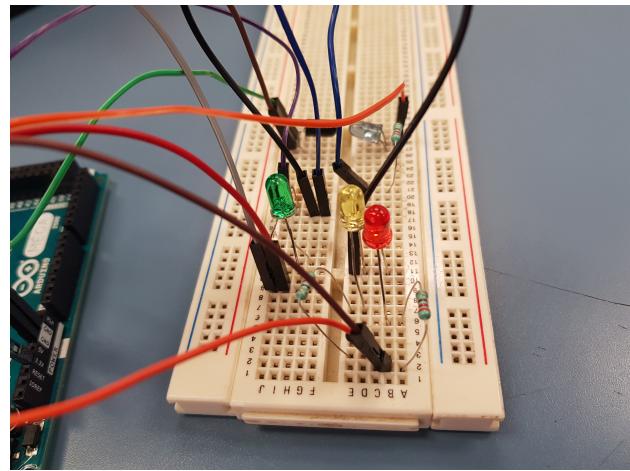


Figura 2. Semáforo de LEDs, activado por interrupciones

los temporizadores e interruptores. Fue una práctica demandante en cuanto a investigación, ya que para la solución de algunos problemas que tuvimos la encontramos en los manuales del AVR que indican el funcionamiento de cada sección de la práctica que vimos. El poder ver un resultado mediante LEDs es una manera también de entender lo que hace el programa internamente para que arroje el resultado deseado.

- Leonardo:** Programar MCs en ensamblador si bien puede ser engorroso, definitivamente es una ventaja en casos particulares en los que la función que se requiere programar no exista en el lenguaje de alto nivel que se esté usando o haya una manera más eficiente de implementarla —se tiene el control total de los ciclos (pasos) en los que se ejecuta determinada rutina—; en proyectos open source como por ejemplo PX4 hay partes pequeñas pero críticas que se escriben en lenguaje ensamblador para tener el control total del timing en las tareas, así como para reducir el uso de memoria. Asimismo, conocer el lenguaje ensamblador no tendría sentido si no se conoce cómo implementar temporizadores e interrupciones, dado que son la herramienta *de facto* para el aprovechamiento al máximo de los recursos que ofrece el microcontrolador.

## 6. ROL O PAPEL

- Rebeca:** Se encargó de escribir parte de los códigos utilizados, así como verificar fuentes de información en cuanto al mapeo de pinnes y el funcionamiento de los temporizadores.
- Leonardo:**

## 7. FUENTES CONSULTADAS

- Instrucciones en ensamblador:** [https://www.microchip.com/webdoc/avr assembler/avr assembler wb\\_instruction\\_list.html](https://www.microchip.com/webdoc/avr assembler/avr assembler wb_instruction_list.html)
- Mapeo de pinnes:** <https://www.arduino.cc/en/Hacking/ PinMapping2560>

- **Arduino Datasheet:** [http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561\\_datasheet.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf)