Ensamblador, Interrupciones & Temporizadores

Javier Montiel González C.U.159216, Andrea González Cardoso C.U. 157961,
Diego Villalvazo Suzer C.U. 155844

Resumen—En esta práctica hicimos nuestros primeros programas en ensamblador. Implementamos temporizadores, interrupciones y otros programas básicos.

1 Introducción

Para poder entender e implementar microcontroladores es importante entender cómo trabajan. En esta práctica aprendimos lo básico de ensamblador, para poder programar lo más cercano a la máquina posible.

Las interrupciones y los temporizadores son fundamentales para el funcionamiento de todos los sistemas digitales. En esta práctica aprendimos cómo se implementan y para qué sirven.

2 Marco Teórico

Las interrupciones sirven para notificar al microcontrolador que hay un proceso con alta prioridad que necesita ser procesado lo antes posible. El microcontrolador suspende lo que esté haciendo, y atiende dicho proceso. Por eso se llaman "interrupciones". Las interrupciones pueden ser del software o del hardware, es decir, pueden ser del sistema en sí o de una lectura de nuestro entorno (sensor de luz, movimiento, etc.).

Por otra parte, el trabajo de un temporizador es contar. Algunas veces los sistemas o programas tienen que "perder tiempo", es decir, esperar *n* milisegundos sin hacer nada. El temporizador nos permite regular las conexiones de un circuito durante un tiempo programado. A nivel físico, un temporizador cuenta el número de pulsos que suministrados por el reloj del sistema (o el reloj que esté seleccionado).

3 Desarrollo y resultados

Parte 1: Ensamblador

A) Se realizó un programa que calculara el discriminante de una ecuación de segundo orden con parámetros a = 10, b = 20, c = 30.
En primera instancia, elaboramos el código en C. El código se muestra a continuación:

```
int a = 10;
int b = 20;
int c = 30;
int det = 0;
boolean res;

void setup() {
  // put your setup code here, to run once:
  det = (b*b)-(4*a*c);
  if(det>=0){
    res = true;
  }
  else {
    res = false;
  }

}
```

Después, implementamos el mismo programa en lenguaje ensamblador. El código se muestra a continuación:

```
void setup(){
  DDRB = DDRB | B1000000
  Serial.begin(9600);

}
void loop(){
  int var_out = 0;
  asm volatile (
    "ldi r16, 0xa \n"
    "ldi r17, 0x14 \n"
    "ldi r18, 0x1e \n"
    "muls r18, r16 \n" //ac
    "lsl r1 \n"
    "lsl r0 \n"      //2ac
    "brcc label \n"
    "inc r1 \n"
    "label: \n"
    "lsl r1 \n"
    "lsl r0 \n"     //4ac
    "brcc label1 \n"
    "inc r1 \n"
    "label1: \n"
    "mov r11, r1 \n"
    "mov r10, r0 \n"
    "muls r17, r17 \n"  //b²
    "mov r13, r1 \n"
    "mov r12, r0 \n"
    "sub r12, r10 \n"  //b²-4ac
    "sbc r13, r11 \n"
    "movw %0, r12 \n\t"
    : "=r" (var_out)

    );

    if(var_out == -800){
      Serial.print("S");
```

```
    }

}
```

Al comparar el código escrito en ensamblador con el código generado por el compilador[1], pudimos darnos cuenta que la cantidad de líneas de código generadas es muy superior.

B) El objetivo era realizar un programa que calculara el promedio de n números. Sin pérdida de generalidad, elegimos n = 5.

Primeramente, elaboramos el código en C. El código utilizado se muestra a continuación:

```
void setup() {
 Serial.begin(9600);
 int i,suma, num;
 int arreglo[] = {2, 4, 8, 3, 6};
 double tam = 5;
 suma = 0;
 i = 0;
 double prom = 0;

 for(i; i<tam ;i = 1+i) {
  num = arreglo[i];
  suma = suma + num;
 }

 prom = suma/tam;
 Serial.print(prom, DEC);
}

void loop() {
}
```

Posteriormente, implementamos el mismo programa en lenguaje ensamblador. El código se muestra a continuación:

```
void setup() {
 Serial.begin(9600);
}

void loop() {
 int var_out = 0;
 asm volatile (
    "ldi r16, 0x07 \n"
    "ldi r17, 0x08 \n"
    "ldi r18, 0x01 \n"
    "ldi r19, 0xa  \n"
    "ldi r20, 0xf  \n"
    "ldi r21, 0x0  \n"
    "ldi r22, 0x05 \n"

    "add r16, r17 \n"
    "adc r16, r18 \n"
    "adc r16, r19 \n"
    "adc r16, r20 \n"

    "label: \n"
    "sbc r16, r22 \n"
    "inc r21 \n"
    "cp r16, r22 \n"
    "brpl label \n"
    "mov %0, r21 \n\t"
    : "=r" (var_out)
 );

 Serial.println(var_out);

}
```

De nuevo al comparar el código escrito en ensamblador con el código generado por el compilador[1], nos percatamos de la gran cantidad de líneas generadas, las cuales al ser una operación más complicada son muchas más que las que se generaron para el discriminante.

C) Se hizo un programa que hiciera parpadear un LED utilizando ensamblador. Igualmente, implementamos un botón que cambiara el estado del LED. El código se muestra a continuación:

```
void setup() {
   DDRB = B00100000;
}

void loop() {
  asm volatile(
    "cbi %0, %1 \n" \
    "sbis %2, %3 \n" \
    "sbi %0, %1 \n" \
    :    : "I"  (_SFR_IO_ADDR(PORTB)), "I"  (PORTB5),  "I"
(_SFR_IO_ADDR(PINB)), "I" (PINB5)  :
    );
    delay(300);
\ \El delay no se incluye para el código con el botón
}
```

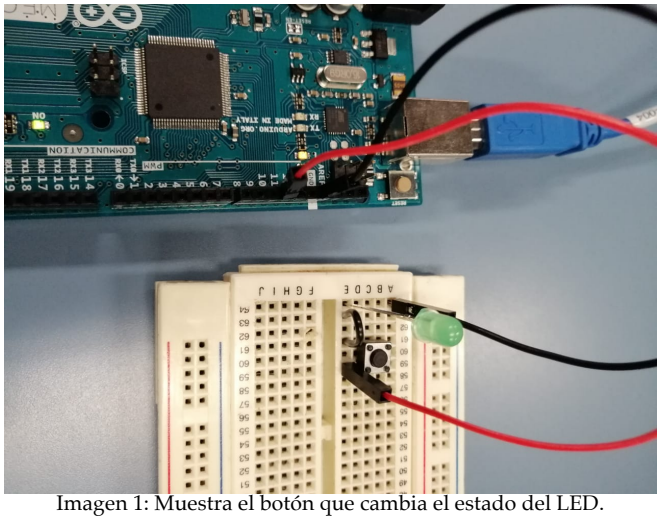

Imagen 1: Muestra el botón que cambia el estado del LED.

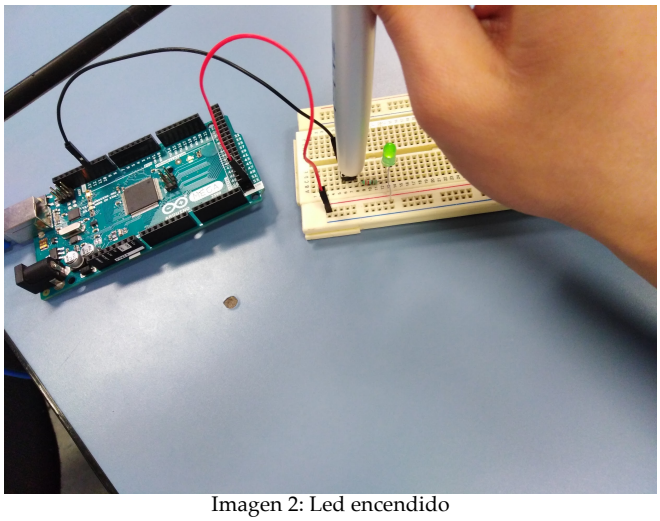

Imagen 2: Led encendido

[1] Los códigos generados por el compilador en ambos incisos, se incluyeron en archivos separados.

Parte 2: Interrupciones

A) El objetivo fue hacer que un led parpadeara a una frecuencia de 2Hz y en otro código implementar un contador asociado a un botón. Cada que se presionara el botón, debía incrementar un contador.

Código 1

```
int amarillo=50;
void setup() {
  pinMode(amarillo, OUTPUT);
}
void loop() {
  digitalWrite(amarillo, HIGH);
  delay(500);
  digitalWrite(amarillo, LOW);
  delay(500);
}
```

Código 2

```
const int buttonPin = 2;
const int ledPin = 12;

int cont = 0;
int buttonState = 0;
int lastButtonState = 0;

void setup() {
   pinMode(buttonPin, INPUT_PULLUP);
   pinMode(ledPin, OUTPUT);
   Serial.begin(9600);
}

void loop() {
  digitalWrite(ledPin, HIGH);
  buttonState= digitalRead(buttonPin);

  if (buttonState != lastButtonState) {
     cont++;
     if (buttonState== HIGH){
       digitalWrite(ledPin, HIGH);
     }
   }

  lastButtonState = buttonState;
  Serial.print(cont/2);
}
```

B) Se reemplazó el botón por un LED infrarrojo y un fotodiodo, cada vez que hay un cambio en el estado del fotodiodo incrementar el contador. Posteriormente, se incluyó dicha función a una interrupción.

```
const int fdPin = 2;
const int ledPin = 12;

int cont = 0;
int fdA = 0;
int fdP = 0;

void setup()
{
  pinMode(fdPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  cli();
  DDRD &= ~(1 << DDD1);
  PORTD |= (1 << PORTD1);
  EICRA |= (1 << ISC10);
  EIMSK |= (1 << INT1);
  sei();
}
ISR(INT1_vect)
{
  digitalWrite(ledPin, HIGH);
  fdA= digitalRead(fdPin);

  if (fdA != fdP) {
     cont++;
     if (fdA== HIGH){
       digitalWrite(ledPin, HIGH);
     }
   }
  fdP = fdA;
  Serial.print(cont/2);
}
void loop() {
}
```

Parte 3: Temporizadores

A) El objetivo fue hacer parpadear un LED utilizando el modo normal a una frecuencia de 2 Hz. Para ello se preescalo el temporizador en 1024 y se tomó como valor inicial 57724. El código implementado se muestra a continuación:

```
int pinLed = 52;
int isON = 0;

void setup() {
  pinMode(pinLed, OUTPUT);
  cli();
  TCCR1B = 0; TCCR1A = 0;
  TCCR1B |= (1 << CS12);
  TCCR1B |= (1 << CS10);
  TCNT1 = 57724;
  TIMSK1 |= (1 << TOIE1);
  sei();

}
ISR(TIMER1_OVF_vect)
{
  if(isON == 0){
    digitalWrite(pinLed, HIGH);
    isON =1;
  }else{
    digitalWrite(pinLed, LOW);
    isON = 0;
  }
  TCNT1 = 57724;

}
```

Posteriormente, se buscó hacer parpadear un LED, pero utilizando el modo CTC a una frecuencia de 4 Hz. Así pues, se preescalo en 1024 el temporizador y el valor del OCR fue de 15625.

```
int pinLed = 52;
int isON = 0;

void setup() {
  pinMode(pinLed, OUTPUT);
  cli();
  TCCR1B = 0; TCCR1A = 0;
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12);
  OCR1AH = 0x3D; OCR1AL = 0X09;
  TIMSK1 |= (1 << OCIE1A);
  sei();

}
ISR(TIMER1_COMPA_vect)
{
  if(isON == 0){
    digitalWrite(pinLed, HIGH);
    isON =1;
  }else{
    digitalWrite(pinLed, LOW);
    isON = 0;
  }
```

}

B) El objetivo fue hacer un semáforo con las siguientes especificaciones:

- Rojo dura 10 segundos.
- Verde dura 15 segundos.
- El amarillo se enciende los últimos 3 segundos con el verde.

Debido a que no se pudo obtener un valor en los parámetros del temporizador que prendiera cada led como se deseaba, lo que se hizo fue calcular los valores para una frecuencia de un hert (el equivalente a un periodo de un segundo) e incluir un ciclo que contara los segundos.

```
int verde = 52;
int amarillo = 46;
int rojo = 48;
int bandera = 0;

void setup() {
 pinMode(verde, OUTPUT);
 pinMode(amarillo, OUTPUT);
 pinMode(rojo, OUTPUT);

 cli();
 TCCR1B = 0; TCCR1A = 0;
 TCCR1B |= (1 << CS12);
 TCCR1B |= (1 << CS10);
 TCNT1 = 49911;
 TIMSK1 |= (1 << TOIE1);
 sei();

 digitalWrite(rojo, HIGH);
}

ISR(TIMER1_OVF_vect)
{
 if(bandera == 10){
  digitalWrite(rojo, LOW);
  digitalWrite(verde, HIGH);
 }
 if(bandera == 22){
  digitalWrite(amarillo, HIGH);
 }
 if(bandera == 25){
  digitalWrite(amarillo, LOW);
  digitalWrite(verde, LOW);
  digitalWrite(rojo, HIGH);
  bandera=0;
 }
 TCNT1 = 49911;
 bandera++;
}
```

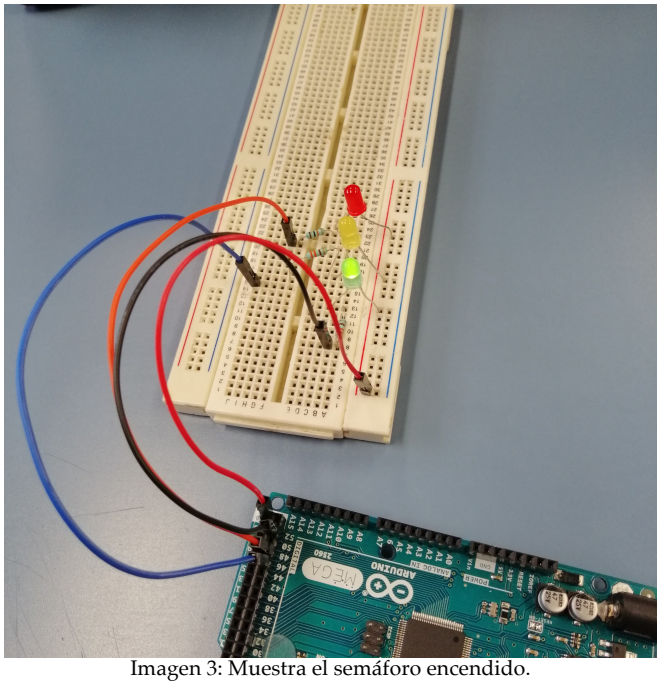

Imagen 3: Muestra el semáforo encendido.

4 CONCLUSIONES

Andrea: Durante esta práctica pudimos reforzar lo aprendido durante la primera práctica al trabajar de nuevo con Arduino en C. Además de eso, aplicamos lo visto en la teoría y programamos en lenguaje ensamblador las tareas que nos fueron asignadas, lo cual fue un gran reto. Después de muchos contratiempos, se pudo completar la práctica de forma satisfactoria y llegamos a los objetivos deseados. También nos dimos cuenta de la importancia y relación que tiene el laboratorio con la teoría, ya que nos da las bases para poder tener un desempeño adecuado.

Javier: En la práctica aprendimos a escribir código tanto desde el compilador como en el ensamblador. Al principio presentamos ciertos problemas debido a que no conocíamos todas las funcionalidades del microcontrolador requeridas para realizar la práctica. Sin embargo, al final se logró implementar exitosamente cada uno de los programas y nos hemos familiarizado mucho más con la programación del arduino.

Por otro lado, uno de los problemas que tuvimos al realizar la práctica fue la falta de un respaldo en nuestros códigos, esto provocó que tuviéramos que volver a hacer varios de ellos. Es necesario que aprendamos a utilizar el repositorio de Github o hacer uso del editor web de Arduino para evitar este tipo de contratiempos.

Diego: Cumplimos los objetivos que nos propusimos al inicio de la práctica. Entendimos cómo implementar código en ensamblador y entendimos cómo funcionan los temporizadores y las interrupciones. También, comparando ensamblador con el código generado por el compilador, logramos ver lo eficiente y simple que es el lenguaje ensamblador.

BIBLIOGRAFÍA

[1] Manual de ATmega640. Consultado el 8 de febrero de 2019. Disponible en http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf