I N S T I T U T O T E C N O L Ó G I C O A U T Ó N O M O D E M É X I C O

Práctica 2

Laboratorio de Principios de Mecatrónica

Presenta:

HUMBERTO MARTÍNEZ BARRÓN Y ROBLES 166056

JUAN CARLOS GARDUÑO GUTIÉRREZ 157302

SEBASTIÁN ARANDA BASSEGODA 157465

Profesor:

Edgar Alejandro Granados Osegueda

Práctica 2

Ensamblador, Interrupciones y Temporizadores
Instituto Tecnológico Autónomo de México
Laboratorio de Principios de Mecatrónica

Resumen

El artículo presenta la práctica de laboratorio en el que se programó un microcontrolador utilizando el ensamblador y se comparó el código del ensamblador con código en lenguaje de alto nivel.

Todo esto se realizó con el objetivo de aprender a utilizar y aprender la configuración y uso de temporizadores.

Finalmente, los resultaron obtenidos fueron exitosos al poder implementar todo lo que demandaba la la práctica.

I. INTRODUCCIÓN

El lenguaje de ensamblador para programar microcontroladores rara vez es utilizado para aplicaciones de sistemas embebidos. Sin embargo, es un conocimiento necesario para cualquier ingeniero que tenga algún interés en desarrollar bibliotecas propias de interacción entre sensores y actuadores. Además, como formación académica, resulta útil conocer el funcionamiento de bajo nivel de los programas para comprender la funcionalidad de lenguajes de programación de niveles más altos.

Esta práctica, al abordar tres temas, se divide en tres partes: la primer parte va enfocada a programas sencillos en ensamblador y su comparación con el código que genera el compilador de un lenguaje de alto nivel, la segunda parte trata acerca del manejo de interrupciones y la tercer y última trata del control y manejo de temporizadores del microprocesador.

En la primer parte de esta práctica, nos enfocamos en comparar el código escrito en ensamblador para una funcionalidad en particular con el generado por un compilador de un lenguaje de alto nivel (al final, es esto lo que sucede en la compilación: se traduce el código de alto nivel a código en lenguaje de ensamblador). Comparar el resultado de un código escrito por personas con el generado automáticamente por el compilador podrá dar una idea del funcionamiento interno del compilador. En esta parte de la práctica, se realizaron tres ejercicios: el cáclulo del determinante de una función cuadrática, el promedio de *n* números y utilizar los puertos de entrada y de salida para hacer parpadear un LED.

En la segunda parte de la práctica, se utilizan funciones que involucran interrupciones, como la función *delay*, o bien interrupciones configuradas a mano para manejar el caso de que un elemento de un circuito cambie de estado. Para esta parte de la práctica, se realizaron cuatro ejercicios:

1. Hacer parpadear un LED a una frecuencia de 2 *Hz* utilizando la función *delay*.
2. Hacer que un contador incremente su valor al oprimir un botón.
3. Hacer que un contador incremente su valor al cambiar el estado de un *fotodiodo*.
4. Realizar el ejercicio 3 asociando el cambio de estado del *fotodiodo* a una interrupción.

En la tercer parte de la práctica, se trabajó con temporizadores. En esta parte, los ejercicios se simplificaban bastante porque ahora era posible que, gracias a una configuración particular del reloj, un evento suceda cada determinado tiempo (pues un temporizador, como se explica en la siguiente sección, no es más que un contador que envía distintas banderas dependiendo de su configuración). Para esta parte de la práctica, se realizaron tres ejercicios:

1. Hacer parpadear un LED a 2 *Hz* utilizando el temporizador de nuestra elección en modo normal.
2. Hacer parpadear un LED a 4 *Hz* utilizando el temporizador de nuestra elección en modo CTC (véase marco teórico).
3. Desarrollar un semáforo que encienda el LED rojo por 10 segundos, el verde por 15 y el amarillo por los últimos 3 segundos del verde.

II. MARCO TEÓRICO

El propósito de una notificación es avisar al microcontrolador que existe algún proceso con alta prioridad de manera que sea procesado lo antes posible. Esto provoca que el microcontrolador suspenda lo que está haciendo y pase al proceso con la prioridad alta. Es a ello lo que se le conoce como “interrupciones”. Una interrupción puede ser tanto de software o como de hardware, y provenir del sistema en sí o de una lectura del entorno como algún sensor.

Un temporizador lo que hace es contar. Hay sistemas o programas que tienen la necesidad de esperar durante un espacio de tiempo sin realizar ninguna acción. Un temporizador regula las conexiones de un circuito durante un tiempo que ha sido programado. Físicamente, el temporizador cuenta los pulsos suministrados por el reloj del sistema.

III. DESARROLLO EXPERIMENTAL

1.Determinante.

Empezamos por escribir código en C que obtuviera el determinante, a = 10, b = 20, c = 30.

```c
#include<stdio.h>

int main()
{
    int a = 10;
    int b = 20;
    int c = 30;

    bool determinante = b*b > 4*a*c;

    printf("El determinante es positivo: " + determinante);

    return 0;
}
```

Fig. 1 Código en C del Determinante

Posteriormente se redactó en ensamblador el mismo código al punto anterior. Este código se comparó con el código generado por el compilador.

Para esto, fue necesario habilitar la salida *verbose* en Arduino, verificar (o compilar) el código, buscar el archivo en formato *.elf* que se había generado, moverlo a un directorio personal y correr el comando “avr-objdump” para transformar el archivo *.elf* en un archivo de formato *.asm*.

2. Promedio

De manera similar a la primera parte de la práctica, se empezó por redactar un código en C que recibiera un arreglo de 4 elementos y posteriormente calculara el promedio. Seguimos por redactar el código en ensamblador.

```c
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

}

void loop() {
  int var_out = 0;
  asm volatile (
      "ldi r16, 0x01 \n"
      "ldi r17, 0x02 \n"
      "ldi r18, 0x03 \n"
      "ldi r19, 0x04 \n"
      "ldi r20, 0x0  \n"
      "ldi r21, 0x04 \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);
```

Fig. 2 Código en Ensamblador del promedio

Posteriormente a partir del código de C se generó el código del compilador para ser comparado con el de ensamblador.

3. Entradas y Salidas

Utilizando ensamblador se generó un código que hiciera parpadear un LED.

```c
int led = 11;
int current = HIGH;
int next = LOW;

void setup()
{
  cli();
  TCCR1B = 0; TCCR1A = 0;
  TCCR1B |= (1 << CS12);
  TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << WGM12);
  OCR1AH = 0x0F; OCR1AL = 0x42;
  TIMSK1 |= (1 << OCIE1A);
  sei();
  pinMode(led, OUTPUT);
}

void loop()
{

}

ISR(TIMER1_COMPA_vect)
{
  digitalWrite(led, next);
  int aux = next;
  next = current;
  current = aux;
}
```

Fig. 3 Código para parpadear LED

Posteriormente se conectó un botón que permitiera cambiar el estado del LED a través del Ensamblador.

4. Interrupciones

Para la siguiente parte de la práctica se conectó un LED y con el fin de que parpadeara a 2Hz se utilizó la función *delay*.

```
int led = 11;

void setup() {
  // put your setup code here, to run once:
  pinMode(led, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(led,HIGH);
  delay(500); // El tiempo de delay será: 1/2Hz * 1000ms = 500 ms
  digitalWrite(led,LOW);
  delay(500);
}
```

Fig. 4 Código para parpadear LED a 2Hz

Posteriormente, se conectó un botón y se implementó una función que incrementara un contador cada vez que el botón fuera presionado.

```
int boton = 11;
int cuenta = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(boton, INPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  if(digitalRead(boton)==LOW)
  {
    cuenta++;
    Serial.println(cuenta);
  }
  delay(500);
}
```

Fig. 5 Código con contador para el botón

Para continuar con las interrupciones se sustituyó el botón por un LED infrarrojo y un fotodiodo. De esta manera cada vez que había un cambio en el estado del fotodiodo, se incrementa el contador.

```
int fotodiodo = A0;
int cuenta = 0;
int anterior = 0;
int actual = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(fotodiodo, INPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  actual = analogRead(fotodiodo);
  if(anterior != 0 && anterior != actual)
    cuenta += 1;
  anterior = actual;
  Serial.println(cuenta);
  delay(200);
}
```

Fig. 6 Código con contador del fotodiodo

5. Temporizadores

En la siguiente parte de la práctica se utilizaron temporizadores para poder hacer que un LED parpadeara en el modo normal a 2HZ.

```
int led = 11;
int current = HIGH;
int next = LOW;

void setup()
{
  cli();
  TCCR1B = 0; TCCR1A = 0;
  TCCR1B |= (1 << CS12);
  TCCR1B |= (1 << CS10);
  TCNT1 = 0xE17C; // valor inicial: 27,724 = 0xE17C
  TIMSK1 |= (1 << TOIE1);
  sei();
  pinMode(led, OUTPUT);
}

void loop()
{

}

ISR(TIMER1_OVF_vect)
{
  digitalWrite(led, next);
  int aux = next;
  next = current;
  current = aux;
}
```

Fig. 7 Código para parpadear LED a 2Hz con temporizadores

Posteriormente, utilizando temporizadores igualmente, se hizo parpadear un LED utilizando el modo CTC a 4Hz.

```
int led = 11;
int current = HIGH;
int next = LOW;

void setup()
{
  cli();
  TCCR1B = 0; TCCR1A = 0;
  TCCR1B |= (1 << CS12);
  TCCR1B |= (1 << CS10);
  TCCR1B |= (1 << WGM12);
  OCR1AH = 0x0F; OCR1AL = 0x42;
  TIMSK1 |= (1 << OCIE1A);
  sei();
  pinMode(led, OUTPUT);
}

void loop()
{

}

ISR(TIMER1_COMPA_vect)
{
  digitalWrite(led, next);
  int aux = next;
  next = current;
  current = aux;
}
```

Fig. 8 Código para parpadear LED a 2Hz con temporizadores

6. Semáforo

En la última parte de la práctica se tenía que implementar, a través del uso de temporizadores, un semáforo con las siguientes características:

- El primer LED, de color rojo, duraba 10 segundos.
- Lo seguía uno de color verde durante 15 segundos, de los cuales durante los últimos 3 se prendía un segundo LED de color amarillo.
- Al apagarse el LED amarillo se prendía el rojo y al apagarse éste se prendía el verde.

```
int rojo = 11;
int amarillo = 10;
int verde = 9;
int cuenta = 0;
int c = 0;
int v = HIGH;
int r = LOW;
int a = LOW;

void setup() {
  // put your setup code here, to run once:
  cli();
  TCCR0B = 0; TCCR0A = 0;
  TCCR0B |= (1 << CS01);
  TCCR0B |= (1 << CS00);
  TCCR0A |= (1 << WGM01);
  OCR0A = 0xFA;
  TIMSK0 |= (1 << OCIE0A);
  sei();
  pinMode(rojo, OUTPUT);
  pinMode(amarillo, OUTPUT);
  pinMode(verde, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(verde, v);
  digitalWrite(amarillo, a);
  digitalWrite(rojo, r);
}

ISR(TIMER0_COMPA_vect)
{
  cuenta++;
  if(cuenta%1000 == 0)
  {
    c++;
    Serial.println(c);
    if(c==25)
      c = 0;
    if(c < 12)
    {
      v = HIGH;
      a = LOW;
      r = LOW;
    }
    if(c >= 12 && c < 15)
    {
      v = HIGH;
      a = HIGH;
      r = LOW;
    }
    if(c >= 15 && c < 25)
    {
      v = LOW;
      a = LOW;
      r = HIGH;
    }
  }
}
```

Fig. 9 Código para implementar el semáforo

IV. RESULTADOS

En las primeras 3 secciones del desarrollo, fue posible desarrollar los códigos en C y en Ensamblador sin mayor problema. Al comparar los códigos que escribimos con Ensamblador con los que generó el compilador, pudimos observar que el compilador genera varias instrucciones adicionales, asignando una localidad de memoria a cada variable y, por lo tanto, ocupando más ciclos de reloj. Al final, la funcionalidad lograda es la misma, pero el código escrito por el equipo resultó ser más compacto.

Las interrupciones pudieron ser implementadas de manera exitosa, como se describió en la subsección 4 de la sección III.

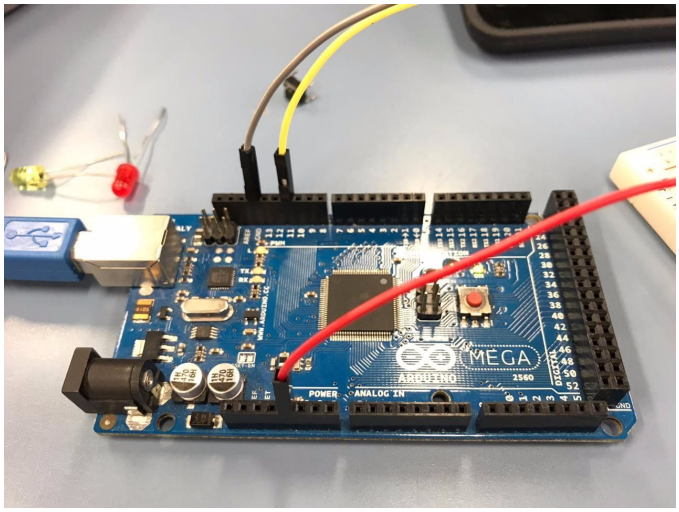

Fig. 10 Cableado para verificar las interrupciones

Al correr nuestro programa pudimos ver reflejado en el monitor cómo aumentaba el contador al presionar el botón.
En nuestro programa con el contador y el botón nos dimos cuenta de que sus limitaciones (especificadas en [1]) son que ningún otro proceso ni instrucción puede correr al tiempo de un *delay*. Se debe de detener todo lo que hace el microcontrolador para esperar el tiempo especificado en la función (esto incluye, por supuesto: lecturas sensoriales, cálculos y manipulación de *pines*). Es necesario tener cuidado al utilizar esta función en programas más complicados para tiempos mayores a unas cuantas decenas de milisegundos, pues rara vez será deseable detener todo un programa.

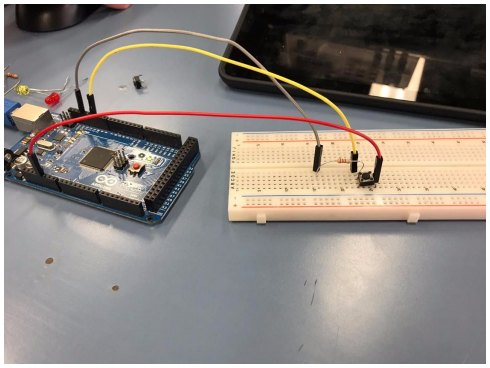

Fig. 11 Botón con el cual se incrementaba el contador

De la misma manera pudimos comprobar el incremento del contador con el fotodiodo a través del uso de una cámara para verificar que prendiera el LED infrarrojo.

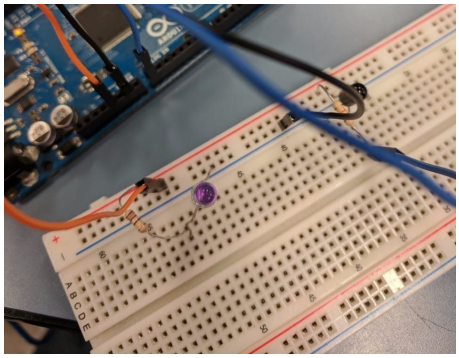

Fig. 12 Es posible ver al LED infrarrojo encenderse a través de la cámara de algunos celulares

De la misma manera pudimos comprobar el incremento del contador con el fotodiodo.

V. CONCLUSIONES INDIVIDUALES

Humberto Martínez Barrón y Robles - El microcontrolador AtMega2560 tiene una variedad interesante de opciones para lograr comportamientos diferentes en los programas. Un solo objetivo se puede lograr utilizando interrupciones o temporizadores, o quizás incluso algún otra herramienta. Estos programas, asimismo, se pueden escribir en ensamblador o en un lenguaje de alto nivel.

Un resultado importante de la práctica fue darse cuenta de la importancia de hacer programas en lenguaje de ensamblador. La mayoría de los microcontroladores permiten una interacción por medio de una interfaz de lenguajes de alto nivel. Sin embargo, conocer programas en ensamblador da una gran ventaja: para cualquier microcontrolador, seremos capaces de escribir programas si conocemos bien lo que queremos lograr y cómo lograrlo. Por ejemplo, si en un programa requerimos utilizar un temporizador, simplemente necesitaríamos buscar en la hoja de datos las instrucciones necesarias para controlar esta herramienta. Conocer los fundamentos de algo siempre rinde como recompensa algo de flexibilidad.

En cuanto a futuras interacciones con la tarjeta Arduino, será necesario que, antes de hacer un programa, se tenga muy claro qué tipo de problema es para proponer la mejor forma de resolverlo antes de tratar de saltar a una solución que, además de incierta, pueda resultar inadecuada.

Juan Carlos Garduño Gutiérrez - Por un lado, en esta práctica me llevo una importante lección para mi vida como ingeniero. Aprendí que es muy importante entender la teoría, cosas que todos nos lo dicen y sólo lo aprendemos cuando lo vivimos en carne propia. También entendí que no es lo mismo la práctica que la teoría. Por ejemplo, en clase vimos como hacer lo solicitado en la práctica y desde mi punto de vista, esta teoría no se acaba de entender hasta que te sientas y te enfrentas a la programación de la tarjeta.

Hasta ahora, me siento satisfecho con lo que hemos trabajado en el laboratorio a diferencia de la materia anterior (Circuitos Lógicos) en donde los temas teoría-laboratorio iban totalmente desfasados. En esta materia los temas que se ven en teoría están siendo reforzados e ilustrados en el laboratorio.

Por otro lado, me llevo el aprendizaje de cuando es pertinente usar temporizadores o interrupciones o cuando conviene implementar el programa en código ensamblador. Es decir, ahora estoy un poco más consciente de todos estos temas que cuando los vi en clase. Desde mi punto de vista, el equipo cumplio con los objetivo de la práctica.

Sebastián Aranda Bassegoda - En esta práctica aprendí la importancia del conocimiento teórico en el laboratorio. A diferencia de otros laboratorios en los que había tenido clase, no siempre se había relacionado de manera tan estrecha el conocimiento teórico con el práctica como sucedió en esta práctica.
Esta práctica también fue una buena oportunidad para expandir mi conocimiento respecto a la programación en ensamblador, es un programa difícil, particularmente ahora que empezamos e hizo mucha falta estar regresando el manual

para encontrar comandos que nos facilitaran nuestros objetivos del programa.
De la misma manera que esto fue un reto, también lo fueron los temporizadores y las interrupciones, me enorgullece, seguramente como al resto de mi equipo, poder haber sacado adelante todas las secciones de la práctica de manera exitosa a pesar de tener que utilizar herramientas que nos eran relativamente desconocidas.

VI. Rol o Papel

VI-A. Humberto Mart ínez

En la primera seccion de la práctica, Humberto tomó posesión por más tiempo del teclado. En la primera parte, entre Humberto y Juan Carlos redactaron el código de ensamblador y el generado por el compilador. En la segunda parte, Humberto iba a escribiendo el código, incluyendo su propia lógica y las de sus compañeros. En cuanto al reporte Humberto elaboró las siguientes secciones:

Introducción.
Desarrollo.
Sus respectivas conclusiones.

VI-B. Juan Carlos Garduño

En la práctica, Juan Carlos se encargó de investigar dudas que iban surgiendo, así como tablas e instructivos. En la primera seccion de la práctica, Humberto tomó posesión por más tiempo del teclado. En la primera parte, entre Humberto y Juan Carlos redactaron el código de ensamblador y el generado por el compilador. En la segunda parte, iba dictando algunas secciones del código a Humberto que habìa investigado al igual que del alambrado y podían ser de utilidad para la sección. De igual manera, iba revisando que la sintaxis de Humberto en el código, fuera la correcta. En cuanto al reporte Juan Carlos elaboró las siguientes secciones:

Resultados.
Colaboró en el marco teórico.
Sus respectivas conclusiones.

VI-C. Sebastián Aranda

En la primera porción de práctica, Sebastián se encargó del código en C y apoyó en menor proporción en el código de ensamblador y el generado por el compilador. En la segunda parte Sebastián colaboró con el alambrado y elaboración del código. En cuanto al reporte el elaboró las secciones siguientes:

Resumen.
Colaboró en el marco teórico.
Sus respectivas conclusiones.

VI. REFERENCIAS

1] Arduino Reference, *Delay*. Arduino. Última actualización: 5 de febrero de 2019. Disponible en https://www.arduino.cc/reference/en/language/functions/time/delay/