

Práctica No. 2 Ensamblador, Interrupciones Temporizadores

Paola Del Hierro 157073
Luis Eduardo Delfín 155966
Rafael Zardain Bejar 158948

Resumen—En esta práctica se utilizaron el lenguaje ensamblador del microcontrolador, las interrupciones y timers de esta. En la primera parte solo se programó en ensamblador y en la parte de interrupciones y temporizadores se complementó ensamblador con C. Se logró entender como funcionan estos tres conceptos.

1. INTRODUCCIÓN

El microcontrolador tiene 4 temporizadores de 16 bits y corren a 16MHz. Se pueden preescalar para poder lograr que cuenten "más lento". Cada timer está relacionado a varios registros. El registro TCNTn es el contador que puede ir de 0 a $2^{16} - 1$. Los registros TCCRAn y TCCRBn son de 8 bits cada uno y se utilizan para programar los temporizadores. Otro registro importante es el TIMSKn que contiene las banderas de los temporizadores como las de *overflow* y *compare*. [1]

2. MARCO TEÓRICO

2.1. Ensamblador

El lenguaje ensamblador es un lenguaje de programación de bajo nivel. Consiste en un conjunto de mnemónicos que representan instrucciones básicas para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Un programa escrito en lenguaje ensamblador consiste en una serie de Instrucciones que corresponden al flujo de órdenes ejecutables que pueden ser cargadas en la Memoria de un sistema basado en Microprocesador[2]

2.2. Interrupciones

Una interrupción es una condición para que el programa espere a un evento para que responda. En el caso del arduino (ATMEGA256) hay pines que están hechos para leer las interrupciones (que cambie una señal por ejemplo) y la reacción del programa a esa interrupción se escribe en el programa.

2.3. Temporizadores

Para esta práctica se utilizaron sólo dos modos: normal y CTC. En el primero el timer incrementa el valor de TCNTn hasta el valor máximo. Cuando el registro se llena se prende bandera TOV, que a su vez puede servir como

interrupción. Se puede modificar el contador preescalándolo y/o cambiando el valor inicial de TNCNTn para que el tiempo del ciclo sea menor.

CTC son siglas para *Clear Timer on Compare*. En este modo se asigna un valor a los registros de OCR (A y B) y cuando el contador llega a este valor, se reinicia y se prende la bandera OCFn. Así se puede modificar de modo constante el ciclo del contador y de la onda que genera. También se puede activar una interrupción con esta bandera. [1]

3. DESARROLLO

3.1. Ensamblador

Para poder realizar esta primera sección de la práctica, se programó en el IDE de Arduino. Durante esta parte de la práctica programamos 3 algoritmos tanto en C, como en lenguaje ensamblador. Después, se comparó el código ensamblador con el código generado por el compilador después de ejecutar el código en C.

El primer algoritmo tenía calcular el determinante de la ecuación cuadrática: $10x^2 + 20x + 30$. Se desarrolló el código tanto en C como en ensamblador. Después se comparó nuestro código en ensamblador con el generado por el compilador.

El segundo algoritmo que se desarrolló calculaba el promedio de ciertos números dados, en particular, nosotros elegimos calcular el promedio de los 8 números: 10, 9, 8, 7, 6, 5, 4, 3. Se programó el código tanto en ensamblador como en C. Para la parte de ensamblador, se decidió solo calcular la división entera. Después se comparó nuestro código en ensamblador con el generado por el compilador.

El tercer algoritmo solo se implementó en lenguaje ensamblador. Se debía conectar un LED a la protoboard y hacerlo parpadear cargando el programa en ensamblador al arduino. Después, se necesitaba conectar un botón para cambiar el estado del LED utilizando ensamblador.

3.2. Interrupciones

En el laboratorio se puso en práctica las interrupciones con señales de un LED infrarrojo. Se utilizó la función ISR (Interrupt service routine) para declarar los eventos que sucedieran en caso de activarse la interrupción. Se usa una interrupción para cuando el receptor IR recibe una señal, incrementa el contador.

3.3. Temporizadores

Para poder realizar el programa que haría parpadear un led a 2Hz, primero se configuró el timer1 en modo Normal y con un preescalamiento de 1024. Esto hace que la frecuencia cambie de 16MHz a 15.625KHz. Posteriormente se asigna un valor inicial de 57,724 al registro TCNT1, de esta manera el temporizador solo cuenta 7,812 y la frecuencia generada es de 2Hz. Finalmente se limpia la bandera de overflow, mandando un 1 al registro pertinente.

La función ISR(TIMER1_OVF_vect) permite que cada que se active la bandera de overflow del Timer1, empiece a correr alguna subrutina. También tiene la ventaja que limpia la bandera automáticamente. Previamente se creó una variable local que se inicializa con 0. Esta variable se utilizó como indicadora del estado del led. Si marcaba que el led estaba apagado (0) se prendía y se cambiaba el estado de la variable a 1. En el caso contrario apagaba el led y regresaba la variable al estado original.

En el segundo problema se utilizó la función que se describe anteriormente pero ahora con el parámetro (TIMER1_COMPA_vect) y realiza la misma subrutina. La diferencia es que se utilizó la bandera de comparación OCF1. Para esta parte se configuró el Timer1 para que funcionara en el modo CTC con un preescalamiento de 264. Esto hizo que la frecuencia disminuyera a 60.606KHz. Se asignó el valor 0x3D al registro OCR1AH y 0x09 al OCR1AL. De esta forma el valor a comparar con el valor del temporizador es 15,625 y se obtuvo una frecuencia de 4Hz.

Finalmente para el problema del semáforo se solucionó programando el Timer1 en modo normal y con un preescalamiento de 1024. Se inicializó el TCNT en 49911, de esta forma la frecuencia obtenida fue de 1Hz. Se inicializó una bandera en 0 que aumentaba cada que se prendía el TVO1. El primer led en prenderse era el rojo, cuando la bandera llegaba a 10 (después de 10 segundos), se apagaba el led rojo y se prendía al verde. Al marcar el segundo 22 se prendía el led amarillo. Finalmente en el segundo 25 se apagaban los leds amarillo y verde, se prendía el led rojo y se reiniciaba la bandera.

4. RESULTADOS

4.1. Ensamblador

Para los primeros dos algoritmos, la programación tanto en C como en ensamblador se logró. Sin embargo, para algunos del equipo fue el primer acercamiento al lenguaje ensamblador. Por lo tanto, nos involucramos en una curva de aprendizaje que nos retrasó para seguir desarrollando las demás partes de la práctica.

La comparación del código generado por el compilador y nuestro código en lenguaje ensamblador fue interesante. Por una parte, nos dimos cuenta que solo unas cuantas líneas de código en C, generaban miles de líneas en lenguaje ensamblador. Por otra parte, el código generado por el compilador no se asimilaba al generado por nosotros.

Por otra parte, para el tercer algoritmo, solo se logró el primer objetivo, el cual era hacer parpadear el LED utilizando lenguaje ensamblador. Por falta de tiempo, no se pudo implementar el botón que cambiaría el estado del LED al presionarlo.

4.2. Interrupciones

Después de probar exitosamente cambiar el contador al recibir un cambio de valor analógico en el sensor IR, se intentó implementar el código de las interrupciones. Al principio fue un poco complejo entender cómo asignar el puerto interruptor y el cableado correcto del sensor. Pero después de una ligera exploración del datasheet se logró correr exitosamente el contador.

4.3. Temporizadores

Para los primeros dos problemas se indicaba el modo en el que se debía usar el temporizador. Los escalamientos se decidieron de manera aleatoria de acuerdo al escalamiento deseado. Finalmente los valores de TCNT y OCR fueron calculados con las siguientes fórmulas.

$$f_{normal} = \frac{f_{microcontrolador}}{N * (2^{16} - TCNT)}$$

$$f_{CTC} = \frac{f_{microcontrolador}}{N * OCR}$$

Los valores fueron despejados al igualar con 2Hz y 4Hz respectivamente. Aunque la combinación de estos valores y los preescalamientos lograron obtener el resultado deseado, una mejora que se puede hacer es elegir metódicamente el preescalamiento.

Para el semáforo se utilizó la misma configuración que en modo normal (preescalamiento de 1024) y se despejó TCNT para igualar la frecuencia a 1Hz. El semáforo funciona de la manera deseada, pero el valor de TCNT se tenía que volver a asignar cada interrupción. Dado que el valor era el mismo, se pudo haber utilizado el modo CTC y así solo asignar una vez el valor al OCR.

5. CONCLUSIONES

Paola Del Hierro En esta práctica se logró aplicar directamente los conceptos vistos en clase y ayudó a entender mejor estos. La parte de ensamblador fue la más complicada y no se logró completar toda la sección de la práctica. Esto se debe a la poca familiaridad que se tiene con el lenguaje y su funcionamiento en el microcontrolador.

Luis Eduardo Delfín Durante esta práctica, se aplicaron muchos de los conceptos teóricos vistos en clase. Por un lado, aprendimos a cómo utilizar el lenguaje ensamblador, cosa que se nos dificultó mucho. Por otro lado, aprendimos a utilizar y a programar interrupciones del programa y temporizadores.

Rafael Zardain Estuvo muy interesante jugar con un Arduino con el nivel más bajo que se pueda programar. Fue un reto aprenderlo pero al final del día aprecias un poco más todo el proceso complejo que hay detrás de cosas tan triviales como hacer parpadear un LED.

6. ROL O PAPEL

Paola Del Hierro: Programación en ensamblador. Código y cableado en la sección de temporizadores.

Luis Eduardo Delfín: Programación en ensamblador. Código y cableado en la sección de interrupciones.

Rafael Zardain Bejar: Programación en ensamblador. Cableado de sensor IR, botones.

REFERENCIAS

- [1] "Datasheet". *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V* http://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
- [2] *Lenguaje ensamblador* https://www.ecured.cu/Lenguaje_ensamblador