Práctica No.2 Ensamblador, Interrupciones & Temporizadores

Departamento Académico de Sistemas Digitales

Instituto Tecnológico Autónomo de México

Primavera 2019

1. Objetivos

- Programar un microcontrolador utilizando ensamblador
- Comparar código en ensamblador contra código en lenguaje de alto nivel
- Aprender a utilizar interrupciones
- Aprender a configurar y utilizar temporizadores

2. Esbozo de la solución

2.1. Ensamblador

Determinante

1. Escribir un código en C que obtenga el determinante, $a = 10, b = 20, c = 30$. (Ver 3.1)
2. Escribir en ensamblador el mismo código que el punto anterior
3. Comparar el código escrito en ensamblador con el generado por el compilador

Promedio

1. Escribir un código en C que obtenga el promedio de n números
2. Escribir en ensamblador el mismo código que el punto anterior
3. Comparar el código escrito en ensamblador con el generado por el compilador

Entradas y salidas

1. Conectar un LED y hacerlo parpadear utilizando ensamblador
2. Conectar un botón para cambiar el estado del LED utilizando ensamblador
3. Comparar el código escrito en ensamblador con el generado por el compilador

2.2. Interrupciones

1. Conectar 1 LED y hacerlo parpadear a 2Hz utilizando la función *delay*
2. Conectar un botón e implementar una función que incrementa un contador cada vez que se presiona el botón. ¿Qué limitaciones tiene éste programa?

Sensor infrarrojo

1. Reemplazar el botón por un LED infrarrojo y un *fotodiodo*, cada vez que hay un cambio en el estado del *fotodiodo* incrementar el contador
2. Asociar la función a una interrupción (ver 3.2)

2.3. Temporizadores

Modos de temporizadores

- Utilizando temporizadores (ver 3.3), hacer parpadear 1 LED utilizando el modo normal a 2Hz
- Utilizando temporizadores, hacer parpadear 1 LED utilizando el modo CTC a 4Hz

Semáforo
Implementar un semáforo con las siguientes características:

- El rojo dura 10 segundos
- El verde dura 15 segundos
- El amarillo se enciende los últimos 3 segundos del verde
- Al apagarse el amarillo se prende el rojo y al apagarse el rojo se prende el verde

3. Aspectos técnicos

3.1. Ensamblador

El compilador para procesadores AVR ofrece la posibilidad de introducir código ensamblador directamente en código *C*. El principal uso de ésta herramienta es optimizar pedazos críticos del código así como utilizar instrucciones específicas que no están directamente disponibles en *C*. Se debe llamar a la función *asm* de la siguiente manera:

```
asm volatile(code
            : output operand list
            : input operand list );
```

La primera parte es el código ensamblador, que puede utilizar datos de variables del código en C mediante la lista de operandos de entrada. Ésta así como la de operandos de salida se especifica con los modificadores (tabla 1) y operandos (tabla 2), las instrucciones disponibles se pueden consultar en la sección *34. Instruction Set Summary* de ATMEL ATmega Datasheet. Un ejemplo se muestra a continuación:

```
asm volatile(
    "mov r10, %1  \n\t"
    "mov r12, %2  \n\t"
    "add r10, r12 \n\t"
    "mov %0, r10 \n\t"
    : "=r" (var_out)
    : "r" (var_in_1), "r" (var_in_2)
    );
```

A continuación se muestra un ejemplo de cómo implementar ciclos:

```
asm volatile(
    "mov r10,%1  \n\t"
    "LDI r16, 15 \n\t"
    "LOOP_1: \n\t"
    "INC r10 \n\t"
    "DEC r16 \n\t"
    "BRNE LOOP_1 \n\t"
    "mov %0, r10 \n\t"
    : "=r" (var_out)
    : "r" (var_in_1)
    );
```

3.1.1. Código ensamblador compilado

Para obtener el código ensamblador generado por el compilador:

1. Habilitar la salida *verbose* del compilador (File $\longrightarrow$ Preferences)
2. Compilar/Verificar el código
3. En los mensajes del compilador, encontrar la dirección de un archivo *.elf*
4. Copiar el archivo a una carpeta personal
5. Utilizar el siguiente comando: `avr-objdump -S ARCHIVO.elf > file.asm`

3.1.2. Entradas y salidas

Para establecer el valor de salida desde ensamblador, se utilizan las instrucciones *SBI* y *CBI*. Adicionalmente, se tiene que establecer si el puerto se va a utilizar como entrada (0) o salida (1) mediante el puerto de *Data Direction Register* ***X*** DDR**X**. Es necesario identificar el puerto que corresponde al *pin* utilizado mediante el Mapeo de Pines. Identificando el puerto, se puede usar la macro *_SFR_IO_ADDR()*, que permite mapear registros. A continuación se muestra como prender el LED de la tarjeta:

```
void setup()
{
    DDRB =  DDRB | B10000000; // Data Direction Register B: Inputs 0-6, Output 7
}
void loop()
{
    asm volatile (
        "sbi %0, %1 \n\t" \
        : : "I" (_SFR_IO_ADDR(PORTB)), "I" (PORTB7)
    );
}
```

Para las entradas, se utilizan las instrucciones *SBIS* y *SBIC*. A continuación se muestra un código que cambia el estado del LED de acuerdo a la entrada en el PIN11:

```
void setup()
{
    DDRB =  DDRB | B10000000;
}
void loop()
{
    asm volatile (
        "cbi %0,  %1           \n" //high
        "sbis %2, %3           \n" //skip next if pin high
        "sBI %0, %1              \n" //low
        :  : "I" (_SFR_IO_ADDR(PORTB)), "I" (PORT7), "I" (_SFR_IO_ADDR(PINB)), "I" (PINB5)  :
    );
}
```

3.2. Interrupciones

El microcontrolador ATmega2560 cuenta con múltiples interrupciones. Para utilizar una interrupción en particular, es necesario:

1. Deshabilitar globalmente las interrupciones
2. Establecer como entrada el puerto correspondiente (Ver mapeo de puertos para el Arduino 2560)
3. Activar el bit de la interrupción en el puerto correspondiente (Ver mapeo de puertos para el Arduino 2560)
4. Establecer el tipo de interrupción (ver tabla 15-3 de ATMEL ATmega Datasheet)
5. Habilitar la interrupción deseada en el registro de máscaras de interrupciones externas (sección 15.2.3 de ATMEL ATmega Datasheet)
6. Habilitar globalmente las interrupciones
7. Declarar la función ISR con el parámetro correspondiente a la interrupción.

A continuación se muestra el código para activar la interrupción 1:

```
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
}
```

3.3. Temporizadores

El microcontrolador cuenta con temporizadores internos que al cumplir una condición levantan una interrupción. Los temporizadores del 2560 tienen un ciclo de reloj de 16MHz y utilizan registros de 16 bits. El microcontrolador cuenta con 16 modos distintos, aquí se presentan dos. Debido a que 16 bits no son suficientes para la mayoría de las aplicaciones (por ejemplo para contar un segundo), se utilizan pre-escalares (ver tabla 17-6 de ATMEL ATmega Datasheet). La señal de reloj y el prescalar son alimentadas a un divisor de frecuencia, cuya salida es la señal utilizada por el temporizador.

3.3.1. Modo normal

En modo normal, un registro comienza en un valor preestablecido y la interrupción se levanta al existir un *overflow* (0xFFFF a 0x0000). Se deben seguir los siguiente pasos:

1. Deshabilitar interrupciones
2. Establecer los registros de control del temporizador seleccionado en 0 (ver sección 17.11 de ATMEL ATmega Datasheet)
3. Establecer el pre-escalar seleccionado
4. Establecer el valor inicial del contador mediante la siguiente fórmula: $2^{16} - \frac{16M}{pre \cdot Hz_d}$
5. Activar la mascara de la interrupción correspondiente (ver sección 17.11 de ATMEL ATmega Datasheet)
6. Habilitar interrupciones
7. Declarar la función correspondiente (Ver tabla 14-1 de ATMEL ATmega Datasheet)

A continuación se muestra el código para activar el temporizador 1 en modo normal:

```
void setup()
{
    cli();
    TCCR1B = 0; TCCR1A = 0;
    TCCR1B |= (1 << CS12);
    TCNT1 = 3036;
    TIMSK1 |= (1 << TOIE1);
    sei();
}

ISR(TIMER1_OVF_vect)
{
    ... // Código a ejecutar al activarse la interrupción
}
```

3.3.2. Modo CTC - *Clear Timer on Compare*

En éste modo, el registro del temporizador se incrementa hasta igualar el registro OCR. Además de levantar la interrupción, al registro del temporizador se le asigna 0x0000 y se reinicia el conteo. Se deben seguir los siguiente pasos:

1. Deshabilitar interrupciones
2. Establecer los registros de control del temporizador seleccionado en 0 (ver sección 17.11 de ATMEL ATmega Datasheet)

3. Establecer los bits WGM=4 (ver tabla 17-2 de ATMEL ATmega Datasheet)
4. Establecer el valor de OCR
5. Activar la mascara de la interrupción correspondiente (ver sección 17.11 de ATMEL ATmega Datasheet)
6. Habilitar interrupciones
7. Declarar la función correspondiente (Ver tabla 14-1 de ATMEL ATmega Datasheet)

A continuación se muestra el código para activar el temporizador 1 en modo CTC:

```
void setup()
{
    cli();
    TCCR1B = 0; TCCR1A = 0;
    TCCR1B |= (1 << WGM12);
    OCR1AH = 0x3D; OCR1AL = 0x09;
    TIMSK1 |= (1 << OCIE1A);
    sei();
}

ISR(TIMER1_COMPA_vect)
{
    ... // Código a ejecutar al activarse la interrupción
}
```

Referencias Recomendadas

Arduino https://www.arduino.cc/en/Reference/HomePage.

ATMEL ATmega Datasheet http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

Ensamblador enlinea de arduino https://web.stanford.edu/class/ee281/projects/aut2002/yingzong-mouse/media/GCCAVRInlAsmCB.pdf.

Manipulación de puertos https://www.arduino.cc/en/Reference/PortManipulation

Mapeo de puertos 2560 https://www.arduino.cc/en/Hacking/PinMapping2560

GCC-AVR Inline Assembler Cookbook https://web.stanford.edu/class/ee281/projects/aut2002/yingzong-mouse/media/GCCAVRInlAsmCB.pdf

Timers http://www.avrbeginners.net/architecture/timers/timers.html

4. Anexos

Modifier	Specifies
=	Write-only operand, usually used for all output operands
+	Read-write operand
&	Register should be used for output only

Tabla 1: Modificadores para los operandos

Constraint	Used for	Range
a	Simple upper registers	r16 to r23
b	Base pointer registers pairs	y, z
d	Upper register	r16 to r31
e	Pointer register pairs	x, y, z
G	Floating point constant	0.0
I	6-bit positive integer constant	0 to 63
J	6-bit negative integer constant	-63 to 0
K	Integer constant	2
L	Integer constant	0
l	Lower registers	r0 to r15
M	8-bit integer constant	0 to 255
N	Integer constant	-1
O	Integer constant	8, 16, 24
P	Integer constant	1
q	Stack pointer register	SPH:SPL
r	Any register	r0 to r31
t	Temporary register	r0
w	Special upper register pairs	r24, r26, r28, r30
x	Pointer register pair X	x (r27:r26)
y	Pointer register pair Y	y (r29:r28)
z	Pointer register pair Z	z (r31:r30)

Tabla 2: Operandos de Entrada y Salida