Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Trabajo Práctico Integrador - Electrónica Digital II

Juego Snake Implementado con el Microcontrolador PIC16F887 y Programado en Lenguaje Assembly

---

Autor(es):

Reynoso Choque, Kevin Walter

Riba, Franco

Estudiantes de Ingeniería en Computación

Profesor:

Ing. Del Barco, Martín

Comisión:

COMÚN-7

Índice

Acceso al Repositorio en GitHub (incluye programa en assembly + archivo apk de la app de control bluethoot + diagrama de flujo + simulación en Proteus) + Informe
https://github.com/francoriba/snake_game.X
Acceso a Videos del Proyecto**:**
https://drive.google.com/drive/folders/1FLhA74nuN7D4FifSvaQ3yQJIey2IwdWz?usp=sharing

1. Introducción

El proyecto consiste en recrear el clásico juego Snake o "Viborita" utilizando el microcontrolador PIC16F887 programado con el lenguaje Assembly. El sistema consiste en encender LEDs del tablero (matriz LED de 8x8) de forma tal que se representa a la viborita y que, a su vez, aparezcan las "ratas" de forma aleatoria en el tablero. Se dirige a la serpiente por medio de los controles implementados por software que mantienen una conexión por Bluetooth. Creemos que este proyecto representa una innovación ya que los microcontroladores, en general están destinados principalmente a procesos de automatismos y de control, sin embargo nosotros, como estudiantes de Ingeniería en Computación, hemos decidido darle un enfoque en el cual le dedicamos un poco más de esfuerzo al diseño de un programa en assembly que utiliza una lógica compleja en comparación a las usadas generalmente.

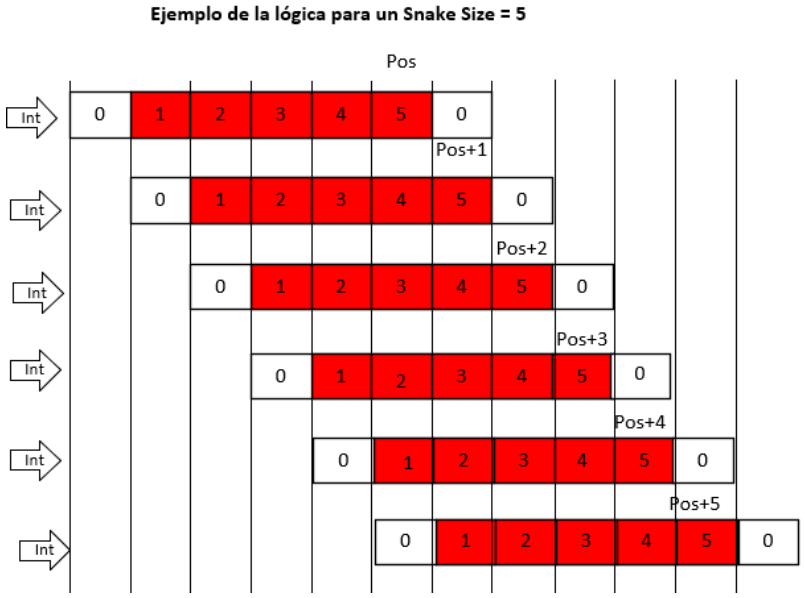
2. ¿Cómo funciona?

Dado que se utiliza una matriz de 8x8, se tendrá 64 segmentos, y para cada uno de ellos se define un registro o variable que actuará como contador (registros A0H a DFH). A su vez, utilizamos 8 registros, uno por cada columna de la matriz, de forma tal que cada bit de estos registros representa el valor lógico de la fila correspondiente.
Al trabajar con interrupciones, cada vez que se genera una interrupción por Timer 1, se modifica la fila o la columna en la que se encuentra la serpiente por medio de los 8 registros mencionados. Cuando la cabeza de la serpiente está en una determinada coordenada de fila y columna, se carga con el tamaño del snake el registro asociado a la fila y columna en que se encuentra el head de la serpiente. Esta relación se muestra en las siguientes imágenes, donde cada columna sería uno de los 8 registros, y cada fila sería cada uno de los 8 bits de alguna de las columnas.

		col							
		1	2	3	4	5	6	7	8
row	1	160	161	162	163	164	165	166	167
	2	168	169	170	171	172	173	174	175
	3	176	177	178	179	180	181	182	183
	4	184	185	186	187	188	189	190	191
	5	192	193	194	195	196	197	198	199
	6	200	201	202	203	204	205	206	207
	7	208	209	210	211	212	213	214	215
	8	216	217	218	219	220	221	222	223

		col							
		1	2	3	4	5	6	7	8
row	1	A0	A1	A2	A3	A4	A5	A6	A7
	2	A8	A9	AA	AB	AC	AD	AE	AF
	3	B0	B1	B2	B3	B4	B5	B6	B7
	4	B8	B9	BA	BB	BC	BD	BE	BF
	5	C0	C1	C2	C3	C4	C5	C6	C7
	6	C8	C9	CA	CB	CC	CD	CE	CF
	7	D0	D1	D2	D3	D4	D5	D6	D7
	8	D8	D9	DA	DB	DC	DD	DE	DF

Además de esto, cuando ocurre una interrupción se decrementan o limpian los registros dependiendo de si estaba vacío o si estaba cargado con algún valor. Esta lógica se puede comprender con el siguiente ejemplo:



Cuando se detecta que un registro está vacío, en vez de decrementarse, se limpia, y luego, para cada registro, se detecta (haciendo uso de una tabla) a qué fila y columna pertenece, y en función de ello se limpian los bits en las variables de columnas C1, C2, … y C7.
En cada interrupción,además de decrementar y limpiar los 64 registros y actualizar los valores de las columnas, se carga en el FSR la dirección del registro con el head de la serpiente y se testea:

- Si el contenido es nulo, entonces se carga dicho registro con el tamaño actual de la variable del tamaño de la serpiente.
- Si el contenido no es nulo, entonces se testea el MSB:
    - Si MSB=0, entonces ha ocurrido una colisión de la serpiente consigo misma por lo que se debe setear el flag que indica colisión y se determina el “game over” en la próxima interrupción.
    - Si MSB=1, entonces la serpiente se ha comido una rata por lo que se limpia el contador de interrupciones del Timer 1 y se incrementa la variable de tamaño de la serpiente.

Otra de las cosas que ocurre al momento de la interrupción es que se verifica si existe alguna “rata” en el tablero. En caso de no existir una, se verifica el valor de un contador (contador de interrupciones del Timer 1) el cual si tiene el valor 10, se lo limpia y se genera una bifurcación en el programa para añadir una rata al tablero. Sin embargo, este contador no se incrementa cuando ya hay una rata en el tablero. La presencia o no de una rata en el tablero se indica mediante el MSB del registro que cuenta las interrupciones del Timer 1 (rattmr). Ese bit se setea en 1 cada vez que el nibble bajo llega a 10 (o sea cuando se debe agregar una rata) y se limpia cada vez que el snake come una rata.
Se ha utilizado una forma pseudoaleatoria de obtener el valor de registro para posicionar una rata el cual consiste en cargar en cualquiera de los 64 registros, en función del tamaño que tiene la serpiente al momento de la interrupción, por medio de PCLATH y PLC. El valor aleatorio entre 0 y 63 se le suma 159 (para tomar desde el registro A0H), se lo carga al registro FSR y luego se incrementa FSR

y, por medio del registro INDF, se accede al contenido del registro el cual tendrá la dirección dada por $numRandom + 160$. Al acceder al contenido, **se verifica que se encuentre vacío** para cargarle el valor 0xFF. Cargar tal valor cumple 2 funciones:

- El segmento asociado a dicho registro permanece encendido el mayor tiempo posible permitido por el hardware en caso que la serpiente nunca coma la rata.
- Al setear en 1 el MSB del registro, se permite que, durante la inspección o barrido de registros, si se encuentra un registro con contenido no nulo, se pueda determinar si ocurrió una colisión o si la serpiente se ha comido la rata.

Si es encontrado el registro aleatorio seleccionado, significa que se debe elegir otro ya que ese registro se encuentra actualmente ocupado por la serpiente. Para elegir otro valor de registro se suma 33(d) al FSR ya que se debe asegurar que el nuevo registro no sea mayor a 223 (ver gráfico tabla de direcciones). El valor 223 es el límite para las direcciones de registro para la cual se puede llegar a necesitar elegir según el siguiente cálculo.

$$63\ (cantidad\ de\ registros\ -\ 1)\ +\ 160\ (A0H)\ =\ 223$$

Por lo tanto, si se quiere ser capaz de detectar el sobrepaso del límite mediante la generación de carry, el número que se debe sumar al FSR viene dado por:

256-223 = 33

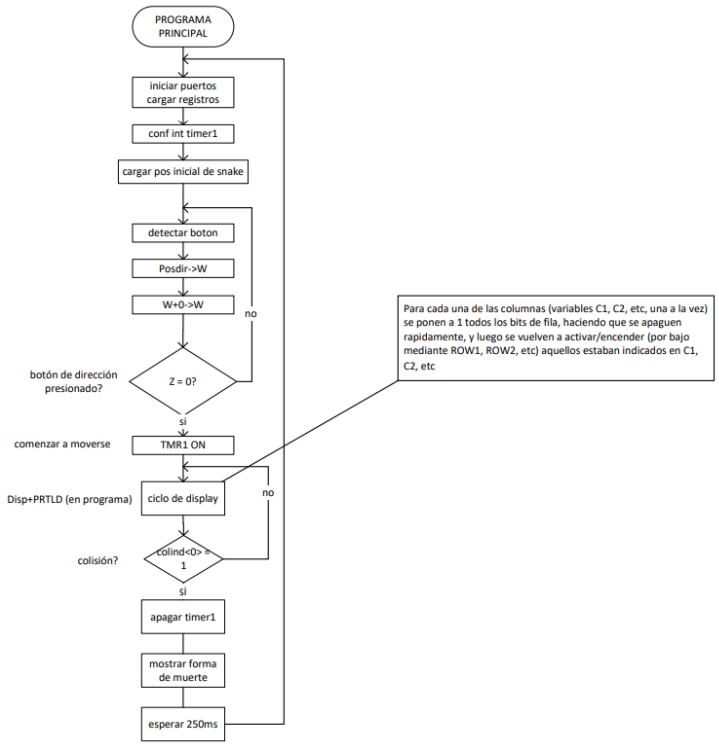
Por ejemplo, en el caso que el registro 223 esté ocupado por la serpiente, se puede detectar haciendo:

```
FSR + 33 -> FSR          ; FSR = 256 -> FSR=0 y C=1
btfss STATUS, C
```

Entonces, se levantará el flag de carry indicando que ese valor pasó el límite. Cuando ocurre, se selecciona el valor del primer registro (A0H).

En el caso en que no se produzca carry, simplemente se pasa a revisar si el nuevo valor de FSR tiene contenidos o no como se hizo anteriormente. Esto se hace de forma iterativa hasta tener un valor de registro válido para colocar la rata. Cuando se lo encuentra, también se incrementa el valor del registro TMR1 para que las interrupciones ocurren con mayor frecuencia con el fin de observar un desplazamiento más rápido de la serpiente así aumentando también la dificultad al juego.

3. Diagrama de Flujo



Dada la amplia extensión que presenta el diagrama de flujo a causa de toda la lógica involucrada, no se hace posible incluirlo completo en este documento. Sin embargo invitamos al lector a buscar el archivo de formato PDF llamado “snake_flowchart” que se encuentra en el repositorio de GitHub o en el enlace de Google Drive indicado a continuación.

- https://github.com/francoriba/snake_game.X/blob/master/snake_flowchart.pdf
- https://drive.google.com/file/d/14nKeRzMbFODcPY6BuH3LH0yZj_XQOiyw/view?usp=sharing

4. Cálculo de Interrupción del Timer 1

Dado que se desea que las interrupciones del del timer ocurran inicialmente cada 0.5s (aunque luego aumenta la frecuencia al consumir una rata) determinamos con que valores cargar los registros TMR1H y TMR1L de la siguiente manera:

$$TMR1 = 2^{16} - \frac{\frac{T_{delay}}{T_{instrucción}}}{PS}$$

Donde $f_{osc} = 4MHz \rightarrow T_{instrucción} = \frac{4}{f_{osc}} = 1\mu s$ y $PS = 8$ según las configuraciones realizadas en el programa, de esta forma resulta:

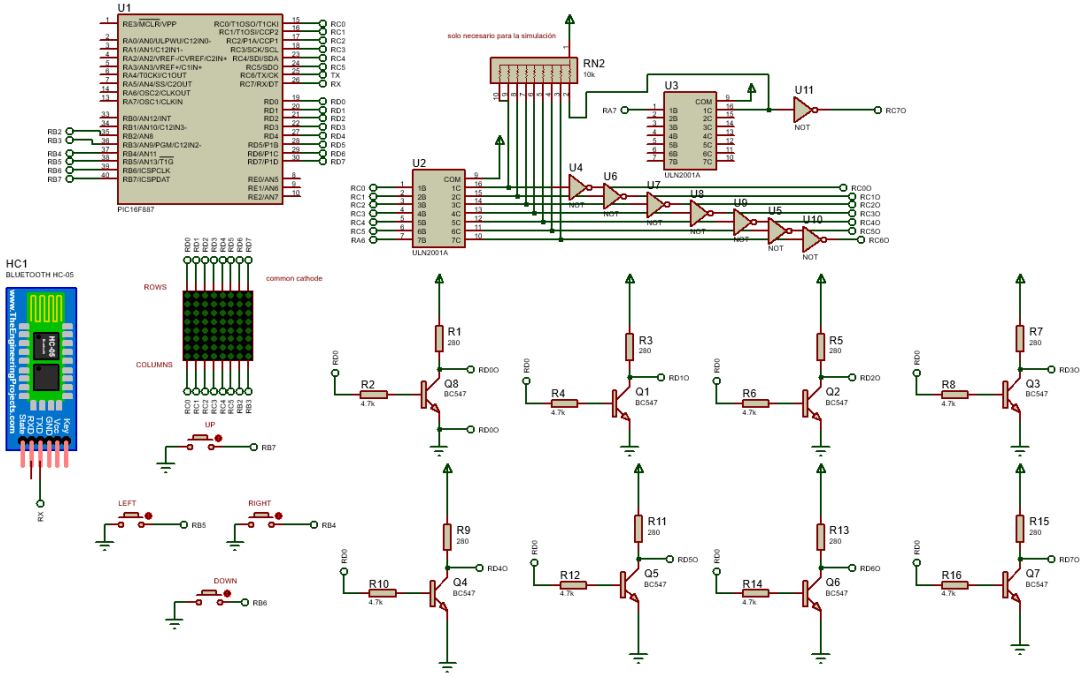
$$TMR1 = 3035 = 0BDB \rightarrow TMR1H = B \; y \; TMR1L = DB$$

5. Comunicación Serie USART

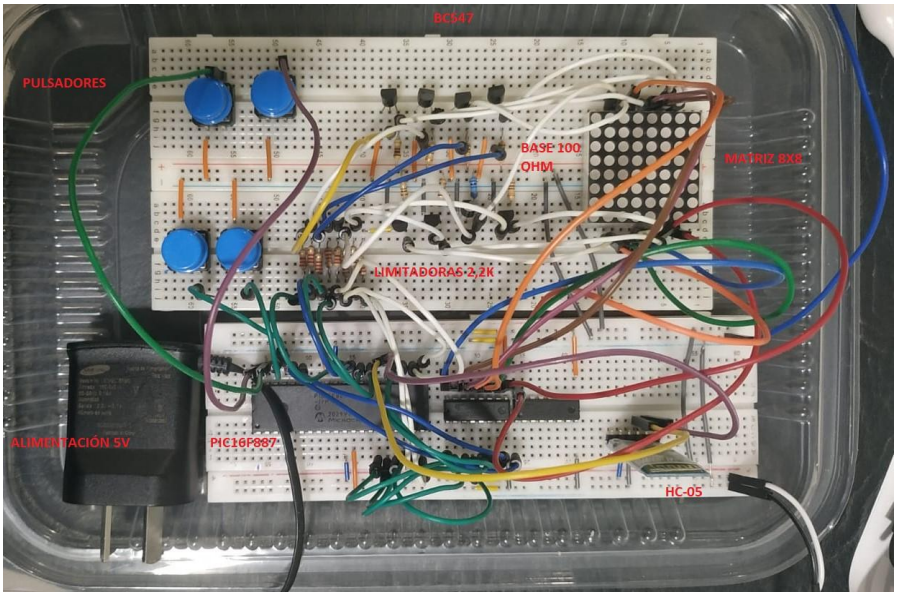
5.1. Recepción de Datos Enviados por Bluetooth mediante el Módulo HC-05

Los pines RX y TX del PIC fueron conectados a los pines TX y RX respectivamente del HC-05. Mediante un dispositivo Android se envían los datos por bluetooth. Estos datos los recibe el HC-05 y se encarga de llevarlos al PIC a través del pin de recepción de datos RX, el módulo fue configurado con ayuda de una placa Arduino UNO, mediante los comandos AT. La configuración realizada fue la necesaria para indicar que la transferencia de datos será a una velocidad de 9600 Baud, con un solo bit de parada y sin bit de paridad.

6. Diagrama Topográfico del Circuito



NOTA: La lógica de la simulación se encuentra invertida respecto a la real usada en el circuito físico, eso se debe a que la matriz LED real y la del simulador son activas por lógicas distintas.



7. Hardware Utilizado

El hardware que utilizamos en este proyecto incluye:

- PIC 16F8887 (x1)
- Matriz LED 8x8, 1088BS (x1)
- Resistores de valores 2,2 KΩ(x8), 100Ω(x8)
- Transistores NPN BC547 (x8)
- Arreglo Darlington ULN2002A (x2)
- Pulsadores/Botones (x4)
- Alimentación: cargador de teléfono celular de 5V (x1)
- Módulo Bluetooth HC-05 (x1)
- Cables Jumper (multiples)
- Cables tipo Arduino
- Protoboards (x2)

8. Resultado

El juego presenta ciertos bugs, como por ejemplo la desaparición del alimento en el tablero si transcurre mucho tiempo sin ser consumido o el hecho de que al dejar pasar suficiente tiempo, si se consume alimento, el registro usado para detectar el tipo de colisión se corre y en vez de incrementarse el tamaño del snake se interpreta como si hubiese colisionado con sigo mismo. El resto de las funciones pensamos que han sido exitosamente implementadas, hemos trabajado con 3 tipos de interrupciones, por cambio de nivel de los pines de los controles, del timer 1 para actualizar registros y generar el movimiento de los segmentos encendidos, y de el periférico para la comunicación serial. Por cuestiones de tiempo no hemos alcanzado a utilizar el ADC, cosa que nos hubiese gustado probar para poner en práctica lo aprendido en las clases teóricas.

9. Conclusión

La realización del trabajo integrador nos ha significado un gran desafío, principalmente para el diseño de la lógica del juego, que ha consumido la mayor parte del tiempo. Una de las cosas que nos generó muchos problemas fue el manejo de tablas cuando tenemos un programa tan extenso, podemos decir que hemos comprendido mucho mejor la forma de operación del PIC respecto al direccionamiento de la memoria de programa. También hemos tenido que aprender a utilizar la herramienta para crear la aplicación de control bluetooth, y poner en práctica de forma intensiva muchos conocimientos adquiridos en electrónica digital 1, esto para el diseño a nivel de hardware y conexionado de nuestro circuito. Hemos aprendido a programar el PIC usando el programador Pickit3 que no habíamos utilizado hasta el momento. Hemos implementado buenas prácticas de programación respecto al versionado de software aprendidas en la asignatura Ingeniería de Software.