Materia: Microprocesadores y microcontroladores

Maestro: García López Jesús Adán

Alumno: Christian Alfredo Montero Martínez

Practica 3: Sección de Memoria (Prueba de memoria RAM) Fecha: 02/03/2017

Práctica No. 3

Sección de Memoria (Prueba de memoria RAM)

Objetivo: El alumno hará uso de una técnica de prueba de memoria aplicándolo en un programa de prueba de memoria RAM.

Material:

- Computadora Personal (PC)
- Programa Edito de texto (ASCII), TASM y TLINK
- Tarjeta T-Juino (con intérprete 80x86)
- Manejador FTDI instalado
- Memoria RAM y Latch para T-Juino.

Equipo:

- Computadora Personal
- Programa emulador de terminal

Teoría:

Algoritmos de prueba para memoria RAM

Algoritmos de marcha/caminata (march test): Una prueba de marcha consiste en una secuencia de elementos de marcha, un elemento de marcha consiste en una secuencia de operaciones aplicadas a cada una de las celdas en uno o dos órdenes de direcciones. Los órdenes de direcciones posibles son:

Incremento: probando las celdas desde la 0 a la n-1.

Decremento: probando las celdas desde la n-1 a la 0.

Regularmente el proceso de prueba consiste en escribir el patrón deseado en cada una de las localidades para posteriormente leerlo y corroborar que el dato almacenado en dicha dirección es el esperado, en algunas implementaciones el dato se lee inmediatamente después de que se escribe, sin embargo hay quienes consideran que es una mejor práctica escribir el patrón en todas las localidades para posteriormente leerlos todos.

Algoritmo de prueba ceros y unos: Este test sencillo consiste en escribir ceros y unos en la memoria, existen algunas variantes de este algoritmo, puede ser implementado escribiendo ceros en cada una de las localidades y después leer cada una de ellas, esperando encontrar el valor cero almacenado en cada una de ellas, posteriormente se hace el mismo recorrido pero almacenando el número 1 en cada una de las localidades, para después leer el contenido de cada una de ellas esperando encontrar el valor 1. Si en algún momento se encuentra un valor diferente al esperado el algoritmo imprime un mensaje de error, otra implementación del algoritmo de ceros y unos consiste en escribir en la primera localidad el número 1 a 8 bits (0000 0001) y al recorrernos a la celda adyacente se recorre el número 1 a la izquierda (0000 0010) y así sucesivamente, al llegar al valor (1000 0000) se reinicia el patrón a 1, al terminar de escribir en todas las localidades se revisa cada una de estas.

Algoritmo de prueba “Tablero de damas”: Las celdas de memoria se dividen en dos grupos (A y B) como en un tablero de damas haciendo referencia a los dos diferentes colores, y en cada una de las celdas del grupo A se escribe el número 1, mientras que en las celdas pertenecientes al grupo B se escribe el número 0. Ese proceso de escritura se puede ver cómo escribir alternadamente ceros y unos en cada localidad de memoria. Posterior a esto se realiza la revisión de cada una de las localidades.

Algoritmo de prueba “Inversiones aleatorias”: Consiste en llenar las localidades de memoria con un patrón pseudo aleatorio, una vez que todas las localidades tengan almacenado éste patrón se revisa el contenido, si no hay errores se procede a llenar las localidades con el complemento del patrón utilizado inicialmente para finalmente volver a revisar el contenido de las localidades. Este algoritmo es muy efectivo para detectar fallos en la memoria.

Funciones peekb() y pokeb() en arquitecturas de x86 bits

La función peekb examina un byte en la memoria, la dirección que examina está definido en los parámetros que recibe (segmento y offset), la función retorna el valor que está almacenado en la dirección especificada. Se utiliza esta función en diversos algoritmos de prueba de memoria, y al invocar a la función hacen la comparación del valor que retorna con el que se espera encontrar.

El prototipo de la función es: *tipo_de_dato* peekb(*int* segmento, *int* offset);

La función pokeb sirve para almacenar en memoria un dato que se pasa como parámetro, en la dirección especificada por sus parámetros (segmento y offset), la función es de tipo void ya que no se requiere que retorne algún dato.

El prototipo de la función es: *void* pokeb(*int* segmento, *int* offset, *tipo_de_dato* dato);

Desarrollo

1. Diseñe e implemente un programa en lenguaje ensamblador para probar una zona de la memoria interna (zona a definir en la sesión de Laboratorio).

2. Diseñe e implemente un programa (lenguaje C + ensamblador)para probar la tarjeta de expansión de memoria RAM que cumpla con las siguientes requisitos:

    a) La prueba de memoria será para el rango de direcciones 2200h a FFFFh que corresponde exclusivamente a la memoria RAM externa.
    b) El programa deberá presentar la dirección a probar y luego proceder a la prueba. Si prueba presenta error entonces detener el proceso indicando error, de lo contrario continuar con la siguiente dirección de memoria. Esto hasta completar la prueba e indicar éxito al final si todas las direcciones pasaron la prueba.

    Nota: El programa deberá hacer uso de las funciones peekb() y pokeb() las cuales están implementadas en lenguaje ensamblador (fuera de línea – archivo .asm) y son llamas desde el programa en lenguaje C.

    Prototipos de las funciones
    unsigned char peekb(unsigned int segment, unsigned int offset)
    void pokeb(unsigned int segment, unsigned int offset, unsigned char data)

Implementación

El inciso A de la práctica fue muy sencillo, el objetivo era únicamente implementar un algoritmo de prueba de memoria en lenguaje ensamblador, sin hacer uso de procedimientos, ni pila, ni uso de variables, todo el código en el procedimiento principal. Mientras que en el segundo inciso se implementó el mismo algoritmo en C con los procedimientos en ensamblador. Asimismo se hizo una conexión entre el microprocesador y una memoria externa como se muestra en la siguiente figura:

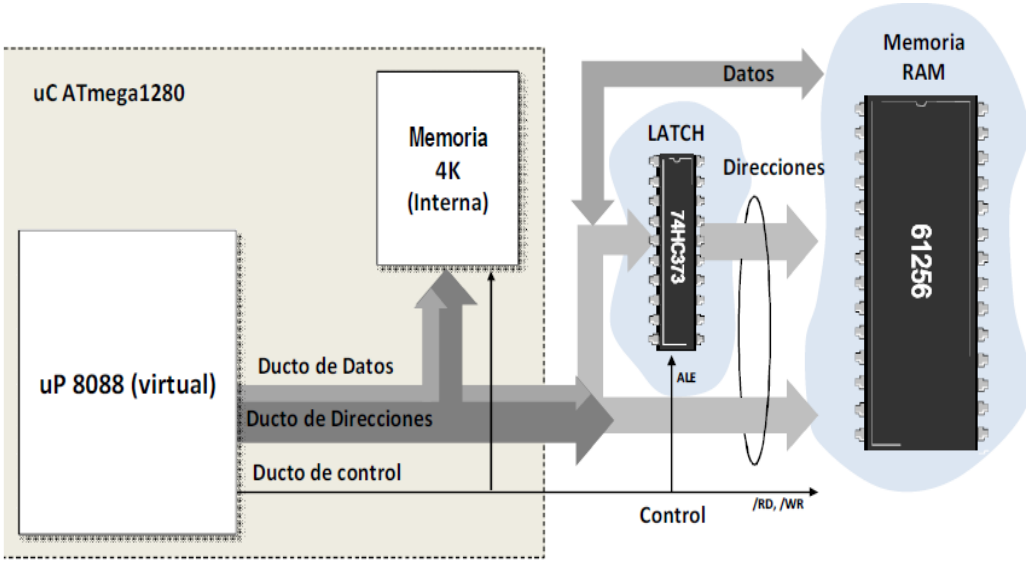

Figura 1: Esquema de memoria de T-juino

Se hicieron las conexiones correspondientes al esquema de la figura 1, utilizando un latch modelo 74373 y una memoria RAM 62256. Para probar el código desarrollado en C con los procedimientos en lenguaje ensamblador.

Resultados

Parte importante de la práctica era retomar la implementación de procedimientos en ensamblador, pasando parámetros desde C, mismos que se recuperan desde la pila en ensamblador, una vez enlazados los códigos de C y ensamblador, generamos el archivo .com y con el programa makebintj generamos el .bin para correrlo en el microprocesador, después de correrlo obtuvimos los siguientes resultados:

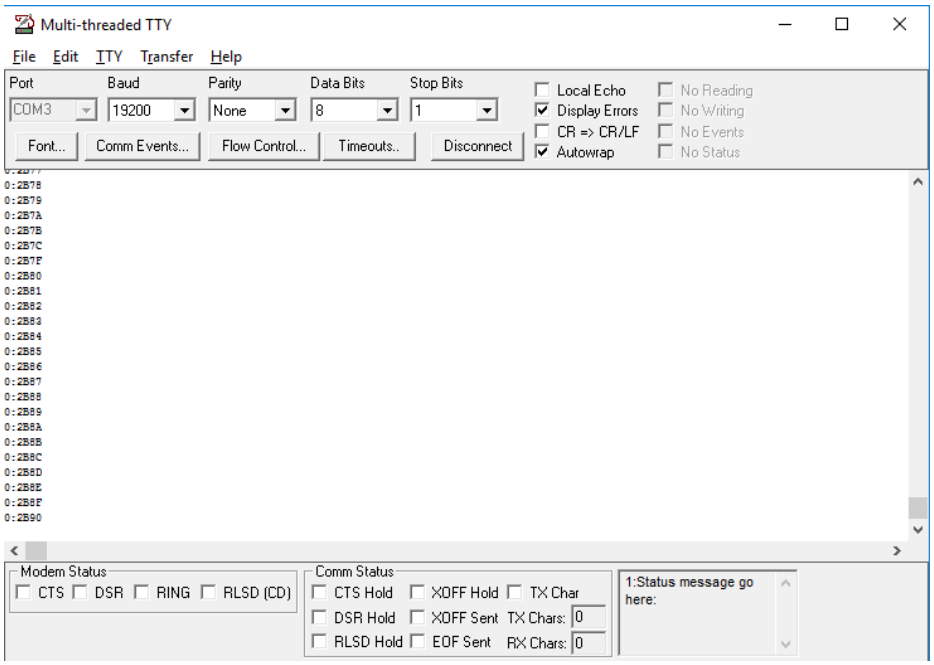

Figura 2: Corrida del programa en el microprocesador

Conclusiones y Comentarios

La práctica fue sencilla, durante la implementación de los procedimientos en ensamblador me percaté que al momento de realizar una prueba con los procedimientos “inline” los parámetros que se mandan desde C se recuperan a partir de BP + 6 mientras que al implementar los procedimientos no “inline” se recuperan a partir de BP + 4. El alambrado era sencillo, únicamente había que tener cuidado con conectar bien los pines basándonos en los datasheets de los dispositivos. Con respecto a los algoritmos de

prueba de memoria, indagando un poco en las diferentes metodologías existentes, me di cuenta que la mayoría de las metodologías son muy parecidas, todas consisten en escribir, números, patrones o cualquier dato en las localidades de memoria para posteriormente revisar el dato almacenado y compararlo con el esperado. La diferencia más representativa entre uno y otro radica en el tipo de patrón que se almacena en la memoria ya que en algunos, el margen de error es menor.