1-3-2017

Practica 3

Sección de Memoria (Prueba de memoria RAM)

Carlos Omar Calderon Meza
MICROPROCESADORES Y MICROCONTROLADORES – FCQI – INGENIERO EN COMPUTACION

Objetivo:

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

Material:

- Memoria RAM y Latch para T-Juino.

Equipo:

- Computadora Personal
- Tarjeta T-Juino.
- Protoboard
- Compuertas lógica (según diseño).
- Una Memoria RAM (2K u 8K)

Teoría:

Algoritmos de prueba para memoria RAM

Hay muchos buenos enfoques para probar la memoria. Sin embargo, muchas pruebas simplemente lanzan algunos patrones en la memoria sin mucho pensamiento o conocimiento de la arquitectura de memoria o como los errores pueden ser mejor detectados. Esto funciona bien para los fallos de memoria dura pero hace poco para encontrar errores intermitentes. Las pruebas de memoria basadas en BIOS son inútiles para encontrar errores de memoria intermitentes. Los chips de memoria consisten en una gran variedad de celdas de memoria apretadas, una para cada bit de datos. La gran mayoría de los fallos intermitentes son el resultado de la interacción entre estas células de memoria. A menudo escribir una celda de memoria puede hacer que una de las celdas adyacentes se escriba con los mismos datos. Una prueba de memoria eficaz intenta probar esta condición.

Por lo tanto, una estrategia ideal para probar la memoria sería la siguiente:

1) Escribir una celda con un cero

2) Escribir todas las celdas adyacentes con uno, una o más veces

3) Compruebe que la primera celda todavía tiene un cero

Debe ser obvio que esta estrategia requiere un conocimiento exacto de cómo las celdas de memoria se establecen en el chip. Además hay un número interminable de diseños de chip y fabricantes que hacen esta estrategia poco práctica. Sin embargo, hay algoritmos de prueba que pueden aproximarse a esta idea.

Algoritmo Inversiones en movimiento utilizado por el programa Memtest86

El algoritmo consiste en lo siguiente:

1) Llenar la memoria con un patrón

2) Comenzando en la dirección más baja

2a comprobar que el patrón no ha cambiado

2b escribe el complemento de patrones

2c incrementar la dirección

Repetir 2a - 2c

3) Comenzando en la dirección más alta

3a comprobar que el patrón no ha cambiado

3b escribe el complemento de patrones

3c decrementar la dirección

Repetir 3a - 3c

Este algoritmo es una buena aproximación de una prueba de memoria ideal pero hay algunas limitaciones. La mayoría de los chips de alta densidad de hoy almacenan datos de 4 a 16 bits de ancho. Con chips de más de un bit de ancho es imposible leer o escribir de forma selectiva un bit. Esto significa que no podemos garantizar que todas las celdas adyacentes han sido probadas para la interacción. En este caso, lo mejor que podemos hacer es usar algunos patrones para asegurar que todas las celdas adyacentes se han escrito al menos con todas las posibles combinaciones de uno y cero.

También puede verse que el almacenamiento en caché, el almacenamiento en búfer y la ejecución fuera de orden interfieren con el algoritmo de inversión en movimiento y hacen menos efectivo. Es posible desactivar la memoria caché, pero la memoria intermedia en nuevos chips de alto rendimiento no se puede desactivar. Para abordar esta limitación se creó un nuevo algoritmo que se llama **Modulo-X**. Este algoritmo no se ve afectado por la memoria caché o el búfer. El algoritmo funciona de la siguiente manera:

1) Para compensaciones de arranque de 0 – 20:

1a escribe cada 20 ubicaciones con un patrón

1b escribir todas las demás ubicaciones con los patrones de complemento

Repetir 1b una o más veces

1c revisa cada 20 ubicaciones el patrón

Este algoritmo logra casi el mismo nivel de pruebas de adyacencia que las inversiones en movimiento, pero no se ve afectado por el almacenamiento en caché o el almacenamiento en búfer. Dado que se realizan pases de escritura separados (1a, 1b) y el paso de lectura (1c) para toda la memoria, se puede asegurar que todos los búferes y memoria caché se han descargado entre pasadas. La selección de 20 como el tamaño de la zancada era algo arbitraria. Los pasos más grandes pueden ser más efectivos pero tardarían más en ejecutarse. La elección de 20 parecía ser una medida razonable entre velocidad y minuciosidad.

Acceso a memoria

Funciones **peek()** y **poke()** en arquitectura x86 (16bits)

Función peek()

Esta función se utiliza para leer un byte o una palabra almacenada en memoria, requiere de 2 parámetros; el segmento de memoria y el desplazamiento (offset), regresa el dato localizado en esa dirección.

Sintaxis:

```
 int peek (unsigned seg, unsigned off); /* leer la palabra (16 bits) en
seg:off */

char peekb (unsigned seg, unsigned off); /* leer el byte (8 bits) en
seg:off */
```

Función poke()

Esta función se utiliza para escribir un byte o una palabra en memoria, requiere de 3 parámetros; el segmento, desplazamiento (offset) y el dato a escribir, almacena el dato en la dirección indicada.

```
void poke (unsigned seg, unsigned off, int valor); /* poner palabra valor
(16 bits) en seg:off */

void pokeb (unsigned seg, unsigned off, char valor); /* poner byte valor
(8 bits) en seg:off */
```

Comentarios y conclusiones:

La mayoría o todos los algoritmos de prueba de memoria se basan en escribir y después leer bits en todas las celdas de memoria utilizando patrones complementarios, básicamente fue exactamente lo que se realizó en esta práctica primero por puro software y después por software-hardware.

Referencias:

http://racielblog.blogspot.mx/2009/10/comprobar-memoria-con-mentest-86.html

ftp://priede.bf.lu.lv/pub/Utilities/test/memt32.htm