66:20 Organización de Computadoras
Trabajo práctico 2: Memorias caché

1. Objetivos

Familiarizarse con el funcionamiento de la memoria caché implementando una simulación de una caché dada.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido. Por este motivo, el día de la entrega deben concurrir todos los integrantes del grupo.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Este trabajo práctico debe ser implementado en C[2], y correr al menos en Linux.

5. Introducción

La memoria a simular es una caché[1] asociativa por conjuntos de cuatro vías, de 2KB de capacidad, bloques de 64 bytes, política de reemplazo FIFO y política de escritura WT/¬WA. Se asume que el espacio de direcciones es

de 16 bits, y hay entonces una memoria principal a simular con un tamaño de 64KB. Estas memorias pueden ser implementadas como variables globales. Cada bloque de la memoria caché deberá contar con su metadata, incluyendo el tag, el bit `V` y la información necesaria para implementar la política de reemplazo FIFO.

6. Programa

Se deben implementar las siguientes primitivas:

```
void init()
unsigned int get_offset (unsigned int address)
unsigned int find_set(unsigned int address)
unsigned int select_oldest(unsigned int setnum)
void read_tocache(unsigned int blocknum, unsigned int way, unsigned int set)
void write_tocache(unsigned int address, unsigned char)
unsigned char read_byte(unsigned int address)
void write_byte(unsigned int address, unsigned char value)
float get_miss_rate()
```

- La función `init()` debe inicializar la memoria principal simulada en 0, los bloques de la caché como inválidos y la tasa de misses a 0.
- La función `get_offset(unsigned int address)` debe devolver el *offset* del byte del bloque de memoria al que mapea la dirección `address`.
- La función `find_set(unsigned int address)` debe devolver el conjunto de caché al que mapea la dirección `address`.
- La función `select_oldest()` debe devolver la vía en la que está el bloque más "viejo" dentro de un conjunto, utilizando el campo correspondiente de los metadatos de los bloques del conjunto.
- La función `read_tocache(unsigned int blocknum, unsigned int way, unsigned int set)` debe leer el bloque `blocknum` de memoria y guardarlo en el conjunto y vía indicados en la memoria caché.
- La función `read_byte(unsigned int address)` debe buscar el valor del byte correspondiente a la posición `address` en la caché; si éste no se encuentra en la caché debe cargar ese bloque. El valor de retorno siempre debe ser el valor del byte almacenado en la dirección indicada.
- La función `write_byte(unsigned int address, unsigned char value)` debe escribir el valor `value` en la posición `address` de memoria, y en la posición correcta del bloque que corresponde a `address`, si el bloque se encuentra en la caché. Si no se encuentra, debe escribir el valor solamente en la memoria.

- La función `get_miss_rate()` debe devolver el porcentaje de misses desde que se inicializó la caché.

Con estas primitivas, hacer un programa que llame a `init()` y luego lea de un archivo una serie de comandos y los ejecute. Los comandos tendrán la siguiente forma:

```
FLUSH
R ddddd
W ddddd, vvv
MR
```

- El comando "`FLUSH`" se ejecuta llamando a la función `init()`. Representa el vaciado del caché.
- Los comandos de la forma "`R ddddd`" se ejecutan llamando a la función `read_byte(ddddd)` e imprimiendo el resultado.
- Los comandos de la forma "`W ddddd, vvv`" se ejecutan llamando a la función `write_byte(unsigned int ddddd, char vvv)` e imprimiendo el resultado.
- El comando "`MR`" se ejecuta llamando a la función `get_miss_rate()` e imprimiendo el resultado.

El programa deberá chequear que las líneas del archivo correspondan a un comando con argumentos dentro del rango estipulado, o de lo contrario estar vacías. En caso de que una línea tenga otra cosa que espacios blancos y no tenga un comando válido, se deberá imprimir un mensaje de error informativo.

7. Mediciones

Se deberá incluir la salida que produzca el programa con los siguientes archivos de prueba:

- prueba1.mem
- prueba2.mem
- prueba3.mem
- prueba4.mem
- prueba5.mem

7.1. Documentación

Es necesario que el informe incluya una descripción detallada de las técnicas y procesos de medición empleados, y de todos los pasos involucrados en el mismo, ya que forman parte de los objetivos principales del trabajo.

8. Informe

El informe deberá incluir:

- Este enunciado;
- Documentación relevante al diseño e implementación del programa, incluyendo las estructuras de datos;
- Instrucciones de compilación;
- Resultados de las corridas de prueba;
- El código fuente completo del programa, en dos formatos: digital e impreso en papel.

9. Fecha de entrega

La fecha de entrega es el jueves 30 de Mayo de 2019.

Referencias

[1] Hennessy, John L. and Patterson, David A., Computer Architecture: A Quantitative Approach, Third Edition, 2002.

[2] Kernighan, Brian, and Ritchie, Dennis, The C Programming Language.