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, y correr al menos en Linux.

5. Introducción

La memoria a simular es una caché [1] asociativa por conjuntos, en que se puedan pasar por parámetro el número de vías, la capacidad y el tamaño de bloque. La política de reemplazo será FIFO y la política de escritura

será 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 bit V, el *tag*, y un campo que permita implementar la política de FIFO.

6. Programa

6.1. Comportamiento deseado

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ tp2 -h
Usage:
  tp2 -h
  tp2 -V
  tp2 options archivo
Options:
  -h,  --help       Imprime ayuda.
  -V,  --version    Versión del programa.
  -o,  --output     Archivo de salida.
  -w,  --ways       Cantidad de vías.
  -cs, --cachesize  Tamaño del caché en kilobytes.
  -bs, --blocksize  Tamaño de bloque en bytes.
Examples:
  tp2 -w 4 -cs 8 -bs 16 prueba1.mem
```

El tamaño del caché es en total, sin contar metadatos: si tiene 4 KB de tamaño, 4 vías y bloques de 256 bytes, tendrá 4 bloques por vía. Si falta alguno de los parámetros del cache o el archivo de entrada no es suministrado, el programa debe reportarlo como un error.

6.2. Primitivas de caché

Se deben implementar las siguientes primitivas:

```
void init()
unsigned int find_set(int address)
unsigned int find_earliest(int setnum)
void read_block(int blocknum)
void write_byte_tomem(int address, char value)
char read_byte(int address, char *hit)
char write_byte(int address, char value, char *hit)
char get_miss_rate()
```

- La función `init()` debe inicializar los bloques de la caché como inválidos, la memoria simulada en 0 y la tasa de misses a 0.

- La función `find_set(int address)` debe devolver el conjunto de caché al que mapea la dirección `address`.
- La función `find_earliest(int setnum)` debe devolver el bloque más 'antiguo' dentro de un conjunto, utilizando el campo correspondiente de los metadatos de los bloques del conjunto.
- La función `read_block(int blocknum)` debe leer el bloque `blocknum` de memoria y guardarlo en el lugar que le corresponda en la memoria caché.
- La función `write_byte_tomem(int address, char value)` debe escribir el valor `value` en la memoria.
- La función `read_byte(address, char *hit)` debe retornar el valor correspondiente a la posición de memoria `address`, buscándolo primero en el caché, y escribir 1 en `*hit` si es un hit y 0 si es un miss.
- La función `write_byte(int address, char value)` debe escribir el valor `value` en la posición correcta del bloque que corresponde a `address`, si está en el caché, y en la memoria en todos los casos, y devolver 1 si es un hit y 0 si es un miss.
- La función `get_miss_rate()` debe devolver el porcentaje de misses desde que se inicializó el cache.
- `read_byte()` y `write_byte()` sólo deben interactuar con la memoria a través de las otras primitivas.

Con estas primitivas (más las funciones que considere necesarias), hacer un programa que lea de un archivo una serie de comandos, que tendrán la siguiente forma:

```
init
R ddddd
W ddddd, vvv
MR
```

- Los comandos de la forma "init" se ejecutan llamando a la función `init` para inicializar la caché y la memoria simulada.
- Los comandos de la forma "`R ddddd`" se ejecutan llamando a la función `read_byte(ddddd)` e imprimiendo el resultado y si es hit o miss.
- Los comandos de la forma "`W ddddd, vvv`" se ejecutan llamando a la función `write_byte(int ddddd, char vvv)` e imprimiendo si es hit o miss.

- Los comandos de la forma "`MR`" se ejecutan llamando a la función `get_miss_rate()` e imprimiendo el resultado.

El programa deberá chequear que los valores de los argumentos a los comandos estén dentro del rango de direcciones y valores antes de llamar a las funciones, e imprimir un mensaje de error informativo cuando corresponda.

7. Pruebas

Se deberá incluir la salida que produzca el programa con los siguientes archivos de prueba, para las siguientes cachés: [4 KB, 4WSA, 32bytes] y [16KB, una vía, 128 bytes].

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

8. Informe

El informe deberá incluir:

- Este enunciado;
- Una descripción detallada de las decisiones de diseño y el comportamiento deseado para el caché.
- El código fuente completo del programa.

9. Fecha de entrega

La fecha de entrega y presentación es el jueves 24 de junio de 2021.

Referencias

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