Práctica 6 - Cache

Organización del Computador 1

Verano 2018

Aclaración: siempre que se informa del tamaño de una memoria cache o de una línea, se está haciendo referencia a la capacidad útil de almacenamiento (descartando la información de "control", como los bits dedicados a los campos *tag*, *line*, etc.)

Jerarquía de Memoria

Ejercicio 1 Se cuenta con una implementación de la arquitectura `ORGA 1` que posee el siguiente ciclo de instrucción dividido en 4 etapas (se incluyen tiempos de ejecución):

I. Búsqueda en memoria de la próxima instrucción (1 acceso a memoria)

II. Decodificación de la instrucción (1 ciclo de *clock*)

III. Búsqueda de operandos (0 / 1 / 2 accesos a memoria)

IV. Ejecución (1 ciclo de reloj + 0 / 1 acceso a memoria)

Se tiene el siguiente código que suma las componentes de un vector de 256 posiciones:

```
vector:      DW 0x????       //elemento 0
             DW 0x????       //elemento 1
             ...
             DW 0x????       //elemento 254

entrypoint:  MOV R0, vector  //con R0 se recorrer el vector
             MOV R1, 0x0000  //en R1 se acumulan las sumas de las posiciones
             MOR R2, 0x0001  // guardo la constante 1 en R2
             MOV R3, 0x0000  //con R3 se cuentan los ciclos.
loop1:       ADD R1, [R0]    //se suma en R1 la posición actual
             ADD R0, R2      //se avanza R0 a la siguiente posición del vector
             ADD R3, R2      //se incrementa el contador de ciclos.
             CMP R3, 0x0100  //fin del vector (256 = 0x0100)
             JL loop1
end:         ...
```

Suponiendo que el acceso a memoria tarda 10 ciclos de *clock*,

a) Realizar el seguimiento del programa y calcular la cantidad de ciclos de *clock* que tarda en ejecutarse.

b) ¿Cuántos ciclos de *clock* son desperdiciados mientras se esperan los datos de memoria?

Cache de Correspondencia Directa

Ejercicio 2 Considerar una máquina con una memoria cache de correspondencia directa de 256B, un tamaño de línea de 8B y una memoria principal de 64KB direccionable a byte:

a) Calcular la cantidad de bits de los campos: *tag*, *line* y *index*.

b) Dar las líneas de la cache en las que se almacenan los datos cuyas direcciones son: 0x111B, 0xC334, 0xD01D, 0xAAAA.

c) Listar todas las direcciones de los datos que se almacenarán en la misma línea que el contenido de la dirección 0x1A1A.

d) Suponer que la cache está vacía, y que se realizan lecturas de datos cuyas direcciones están en el siguiente orden: 0x111B, 0x1100, 0xC334, 0xD01D, 0xAAAA, 0x1118, 0xD01A. Determinar para cada lectura si ésta produjo un *miss* o un *hit*.

e) ¿Qué beneficios se obtuvieron de la implementación de la memoria caché? ¿A qué se debieron? Proponer una mejora para esta situación.

Ejercicio 3 Sea una cache de correspondencia directa. Sea M el tamaño de la memoria principal, C el tamaño de la cache y L el tamaño de la línea. La memoria principal es direccionable a byte:

a) Expresar la cantidad de bits de los campos *tag*, *line* y *index*.

b) Dar el conjunto de posiciones de la memoria principal cuyos datos se almacenarán en la misma línea que el dato de la posición x.

Ejercicio 4 Se dispone de una arquitectura como la del Ejercicio 2. Se sabe además que cada registro tiene 8 bytes (que además es el tamaño de la palabra de esta máquina). En la memoria de esta computadora está cargado a partir de la posición 0 un vector de números de 8 bytes, con un largo total de 256 números. El siguiente programa suma algunos de los números del vector, tarea que repite 1000 veces. El PC comienza en la etiqueta `main`.

```
main:       MOV R8, 0       // con R8 se cuentan las 1000 pasadas
            MOV R3, 0x80    // en R3 se guarda el tamaño del ‘‘salto’’ (en este caso,
                            // de a 16 posiciones puesto que 16*8 = 128 = 0x80)
comienzo:   MOV R0, 0       // con R0 se recorrerá el vector (0 es la posición inicial)
            MOV R1, 0       // en R1 se acumula la suma
loop:       ADD R1, [R0]    // se suma en R1 la posición actual
            ADD R0, R3      // se avanza R0 a la siguiente posición
            CMP R0, 0x800   // fin del vector (256*8 = 2048 = 0x800)
            JL loop         // si no se llegó al final del vector, continúa
            ADD R8, 1
            CMP R8, 1000    // fin de las pasadas
            JNE comienzo    // si no se terminaron las 1000 pasadas, empieza de nuevo
            DW 0
```

a) Dar la tasa de fallos que se produce en la caché (inicialmente vacía), considerando **únicamente los accesos al vector** (y no al código).

b) Dar la tasa de fallos bajo las mismas condiciones del ítem anterior, pero modificando el valor que se carga en `R3` a `0x88` (el salto es ahora de 17 posiciones, dado que `17*8 = 136 = 0x88`).

c) ¿Qué diferencia se observa entre los casos anteriores? ¿A qué se debe?

Cache Asociativa

Ejercicio 5 Sea una cache asociativa. Sea M el tamaño de la memoria principal, C el tamaño de la cache y L el tamaño de la línea. Sea B el tamaño de la unidad de direccionamiento de la memoria principal, así como de la palabra que utiliza esta máquina. Expresar:

a) La cantidad de bits del campo *tag*.

b) La cantidad de líneas de cache.

c) La cantidad de palabras almacenables en la cache.

Ejercicio 6 Analizar el programa del Ejercicio 4, pero con una cache asociativa (con las mismas características: 256B de datos, con un tamaño de línea de 8B). ¿Cuál es ahora la tasa de fallos para los ítems a) y b)? ¿Cuál es entonces la ventaja de una cache de mapeo directo?

Cache Asociativa por Conjuntos

Ejercicio 7 Sea una computadora con las siguientes características:

- Microprocesador de 32 bits, que direcciona a byte.
- Cache asociativa por conjuntos de 2 vías.
- Tamaño de línea de 4 bytes.
- Tamaño de cache de 16KB.
- Cache con algoritmo de sustitución LRU.
- Cada acceso a memoria trae 4 bytes.

Se realizan lecturas consecutivas de las siguientes direcciones de memoria en el orden que aparecen a continuación: 0x1452131E, 0x9875488E, 0x9135131C, 0x9855488E, 0x21375084, 0x9135131E, 0xBBA7b31F. Considerar que la memoria cache se encuentra vacía al empezar las lecturas:

a) Determinar para cada una de las lecturas si hubo *hit* o *miss* de cache. ¡No se olvide de considerar los accesos desalineados!

b) ¿Qué direcciones de memoria se encuentran en la memoria cache al terminar las lecturas?

Ejercicio 8 Se cuenta con una arquitectura con direccionamiento a byte, direcciones, palabras y registros de 32 bits, e instrucciones de longitud fija de 64 bits. Un compilador especializado para esta arquitectura transforma código escrito en lenguaje C al siguiente programa escrito en lenguaje ensamblador de esta arquitectura:

```
            ...
            LD R2 , -1          ; Carga en el Registro 2 el entero -1
Ciclo :     ADD R2 , 1          ; Incrementa R2 en 1
            CMP R2 , 256        ; Compara R2 contra el entero 256
            BGE FinCiclo        ; Si es mayor o igual salta a la dirección de FinCiclo
            LD [R1] , 1         ; Carga la constante 1 en la posición de memoria apuntada
                                ; por el Registro 1
            MUL [R1] , 5000     ; Multiplica el contenido de memoria apuntada por el
                                ; Registro 1 por el entero 5000
            BA Ciclo            ; Salta a la dirección de Ciclo
FinCiclo:   ...                 ; etiqueta FinCiclo
```

a) Calcular el hitrate de la ejecución del ciclo si contamos con una cache asociativa. Esta caché posee líneas de 16 bytes y un tamaño 16 KB útiles. Adicionalmente, esta caché es de código únicamente (sólo cachea los fetch de instrucciones enteras, no los fetch de los datos). Considerar que el programa comienza en la etiqueta `Ciclo`.

b) Una de las características principales de las arquitecturas RISC es un set de instrucciones muy reducido. De esta forma, muchos procesadores RISC no poseen la instrucción `MUL` (la operación de multiplicación) y la reemplazan por sucesivos `ADDs`. Las multiplicaciones por constantes se suelen reemplazar por el correspondiente número de `ADDs`, para evitar el costo adicional de realizar ciclos. Calcular el hitrate de la ejecución del ciclo suponiendo que la instrucción MUL es reemplazada por 5000 aplicaciones de la instrucción ADD.

Ejercicio 9 Dada una caché asociativa de n-vías ¿Es cierto que cada vez que el microprocesador necesita una palabra que no está en la caché esto produce que se transfiera *una única línea* de la memoria a la caché?

Políticas de sustitución

Ejercicio 10 ¿En qué propiedad del código y los datos se basa el algoritmo LRU para justificar su eficiencia? Mostrar un ejemplo en el que el algoritmo LRU degenere y produzca continuamente fallos en la cache.

Ejercicio 11 ¿Es posible implementar una política de desalojo que garantice un *hit rate* superior al 10 %? En caso afirmativo dar una política que cumpla con este requerimiento, y en caso negativo demostrar por qué.

Ejercicio 12 Se cuenta con una memoria principal de 1GB que direcciona a palabras de 2 bytes y una memoria cache de 4 MB con tamaño de línea de 16 Bytes (ambas capacidades se refieren a almacenamiento efectivo de datos). Considerar una cache asociativa por conjuntos de 4 vias y las direcciones D1, D2, D3, D4, D5 con el mismo campo *line* y distinto campo *tag*. Para la secuencia de lecturas D1, D2, D3, D1, D4, D2, D5, D1, D2 indicar:

a) El tiempo total de lectura utilizando el algoritmo de sustitución FIFO.

b) El tiempo total de lectura utilizando el algoritmo de sustitución LRU.

Considerando que:

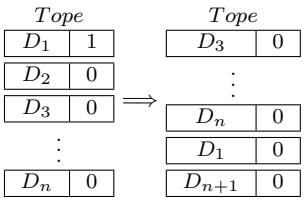
- La cache se encuentra inicialmente vacía.
- Los tiempo de lectura de una palabra en la memoria cache es de 5ns, y en la memoria principal es de 500ns.
- Cuando se produce un *miss*, el tiempo para traer el dato de memoria a cache está incluído en el tiempo de acceso a memoria principal.

Ejercicio 13 Como es sabido, el algoritmo de reemplazo FIFO no se comporta demasiado bien, siendo su principal ventaja su fácil implementación y que requiere poco costo de hardware. Para mejorarlo, se ha diseñado el algoritmo "FIFO second-chance", que se comporta relativamente mejor que FIFO con poco costo adicional. Este algoritmo utiliza una cola de la misma manera que FIFO, pero a cada dato se le asocia un bit R (referenciado). El algoritmo es el siguiente:

- Cuando un dato es movido a la cache, se lo agrega al final de la cola con su bit R en 0.
- Si el dato es referenciado mientras está en la cola, se pone su bit R en 1.
- Cuando se llena la cache se analiza el dato D que se encuentra en el tope de la cola:
  - Si D posee su bit R en 0, se desaloja D y se agrega el nuevo dato al final de la cola, con su bit R en 0.
  - Si D posee su bit R en 1, se mueve D al final de la cola y se pone R en 0.

  y se repite hasta poder desalojar un dato y agregar el nuevo.

Ejemplo del estado de la cola antes y después de agregar el dato D_{n+1}.



Suponer una cache asociativa de n líneas, inicialmente vacía:

a) Se desea comparar FIFO second-chance y FIFO. Sea la siguiente patrón de accesos:

$$D_1, D_2, \ldots, D_n, D_1, D_{n+1}, D_1, D_{n+2}, D_{n+3}, \ldots, D_{2n+1}$$

Suponer que $i \neq j \rightarrow D_i \neq D_j$. Uno de los dos algoritmos tiene mejor comportamiento que el otro si ocurre este patrón de accesos k veces consecutivas, para todo $k > 0$. Decidir de qué algoritmo se trata y justificar apropiadamente la respuesta.

b) Dar otro patrón de accesos donde, al ocurrir k veces consecutivas para todo $k > 0$, el algoritmo que mejor se comportaba en el punto anterior sea ahora el de peor desempeño.

c) Analizando los ítems anteriores, no parece haber razones para que FIFO second-chance se comporte mejor que FIFO. Sin embargo, bajo casos reales de test, FIFO second-chance efectivamente tiene una tasa de *hits* más elevada. ¿Cuál es la propiedad de los accesos a memoria que está aprovechando FIFO second-chance?

Comparación de distintas caches

Ejercicio 14 Considere un microprocesador de 16 bits, que dispone de una memoria principal de 64KB direccionable a byte y una memoria cache de 1KB con tamaño de línea de 2 palabras. Se desea comparar las caches de tipo directa, asociativa y asociativa por conjuntos de 2 vías. Tanto la cache asociativa como la asociativa por conjuntos trabajan con FIFO.

Considerar las direcciones de memoria D1 = E36Ch; D2 = 076Eh; D3 = B160h; D4 = E36E. Se realizarán 5 pedidos de lectura en el orden siguiente: D1, D4, D2, D3, D1. Para cada uno de los tipos de cache, determinar si en el momento de realizar cada uno de los pedidos se produce un *hit* o un *miss*.

En general, ¿qué mecanismo de caché es de esperarse que tenga un mayor *hit rate*? ¿Por qué se utilizan los demás mecanismos?

Ejercicio 15 Un sistema de control trabaja con palabras, registros, bus de direcciones y de datos de 32 bits, instrucciones de longitud fija de 64 bits y direccionamiento a byte. A efectos de mejorar su rendimiento durante la ejecución del fragmento de código que se muestra a continuación, se planea intercalar entre el procesador y la memoria principal un módulo de cache de 16KB. Suponiendo que la comparación produce siempre $Z = 1$, y sabiendo que la CPU dedica el 99 % de su tiempo a ejecutar las *dos últimas instrucciones* de este fragmento, diga cúal sería la tasa de aciertos esperada con cada una de las caches propuestas:

```
0x00003FF0:  MOV R1 ,0x00008000
             MOV R2 ,0x00008008
0x00004000:  CMP [R1],[R2]
             JZ 0x00004000
```

a) Cache de correspondencia directa con líneas de 2 palabras.

b) Cache de correspondencia directa con líneas de 4 palabras.

c) Cache asociativa por conjuntos de 2 vías, líneas de 2 palabras y algoritmo de reemplazo LRU.

Ejercicio 16 Se dispone de una computadora con direcciones de memoria de 16 bits, cada una de ellas direccionando a una palabra de 8 bits. Así, direcciona en total 64 KB de memoria principal. Se desea dotar a esta computadora de una memoria caché, para lo que se barajan dos opciones:

- una caché asociativa por conjuntos de 2 vías, o
- una caché totalmente asociativa

Por el espacio disponible en el CPU, solo se podrán utilizar 512B de memoria caché, disponiéndose además del espacio necesario para los *tags*. La caché se organiza en 128 líneas.

a) Indicar cómo se distribuyen los bits de una dirección de memoria en los campos correspondientes para cada una de las cachés mencionadas.

b) Se conoce que este fragmento del programa insume gran parte del tiempo de cómputo.
Junto a cada instrucción se indica el acceso a memoria necesario para el *fetch* de la instrucción, no así los necesarios para acceder a los datos.

Código		Accesos a memoria del *fetch*
	`MOV R3, 0x1030`	`0x232E, 0x232F`
`main:`	`ADD R1, [R3]`	`0x2330`
	`ADD R1, [R3+0x4502]`	`0x2331, 0x2332`
	`ADD R3, 0x0001`	`0x2333, 0x2334`
	`CMP R3, 0x1046`	`0x2335, 0x2336`
	`JNE main`	`0x2337`

Simular la ejecución hasta el salto (inclusive) utilizando la **caché asociativa por conjuntos de 2 vías con política de desalojo LRU**, teniendo en cuenta los accesos **al código y a los datos.** En cada paso indicar el contenido de la caché y detalle si se producen *hits*, *misses*, desalojos (se nalando la línea desalojada) y/o accesos desalineados.

¿Cuál es el hit rate de esta ejecución parcial?

c) ¿Cuál de las dos cachés es más conveniente para la ejecución completa del fragmento del programa presentado?

d) Indicar el *hit rate* de la ejecución completa del programa (suponiendo que el programa termina al finalizar el ciclo) utilizando la caché totalmente asociativa. Justificar.

Cache de escritura (Opcional)

Ejercicio 17 Escribir el seudocódigo de un controlador de caché *write-through* y *write-back* para una caché asociativa por conjuntos de 2 vías. Considerar el caso de una lectura y una escritura a memoria.

Ejercicio 18 Considerar el siguiente programa que se ejecuta en una computadora con unidades direccionables y direcciones de memoria de 16 bits, con una caché de correspondencia directa a razón de 16 palabras por línea sobre 32 líneas en total. Para los incisos que siguen, solo será necesario tener en cuenta los accesos a datos, ignorando los *fetchs* de instrucciones y de sus operandos. Considere además que, independientemente de la política de escrituras utilizadas, las cachés serán de tipo *write-allocate* (los datos se cargan a caché siempre que son accedidos, ya sea que se trate de una escritura o de una lectura).

```
        MOV R1, 0x0256
        MOV R2, 0x1100
        MOV [R2], 0x0000
suma:   ADD [R2], [R1]
        ADD R1, 0x0001
        CMP R1, 0x0512
        JNE suma
        DW 0x0000
```

a) ¿Cuantos accesos a memoria se producen si se utiliza una caché *write-through*?

b) ¿Cuantos accesos a memoria se producen si se utiliza una caché *write-back*?

c) En general, ¿qué mecanismo es de esperarse que tenga un mejor rendimiento? ¿Existe algún contexto en que resulte conveniente el otro?