Organización del Computador
Práctico 7: Memoria*

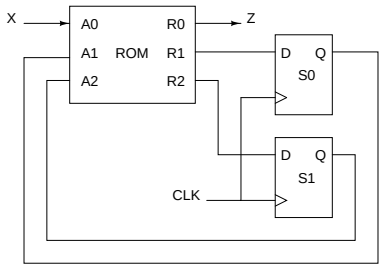
B. Gonzalez Kriegel - N. Wolovick

5 de junio de 2001

1. (a) Un mapa de memoria tiene 2^{24} localizaciones direccionables. ¿Cuál es el menor ancho en bits que las direcciones pueden tener para poder referenciar toda la memoria?

   (b) Dar la dirección de memoria más baja y más alta en un mapa de memoria de bytes direccionable por 20 bits, donde los direccionamientos son *alineados* en palabras de 2 bytes.

2. Se implementa una FSM con una ROM como *lookup-table* y dos FFD.



Locación $A_0 A_1 A_2$	Valor $R_0 R_1 R_2$
0 0 0	0 0 1
0 0 1	1 1 0
0 1 0	1 0 0
0 1 1	0 0 0
1 0 0	1 0 1
1 0 1	1 1 1
1 1 0	0 0 1
1 1 1	0 0 0

   Construya la tabla y el diagrama de transición de estados para esta máquina de estados finitos.

3. Con memorias de 2^n palabras y p bits por palabra muestre:

   (a) Como construir una memoria de $2^n \times 4p$.

   (b) Como construir una memoria de $2^{n+2} \times p$.

4. ! Diseñe el circuito de un decodificador árbol 4-a-16 sin tener en cuenta restricciones de *fan-out*, pero respetando *fan-in*≤ 2.

5. Una memoria caché de *mapeo directo* consiste en 128 ranuras, cada una conteniendo un bloque de 16 palabras. La memoria contiene 256K[1] palabras. El tiempo de acceso a la caché es de 10ns y el tiempo necesario para llenar una ranura de la caché desde la memoria principal es de 200ns.

   (a) Muestre como esta estructura de caché divide las direcciones de memoria.

   (b) Calcule el porcentaje de aciertos de un programa que itera 10 veces sobre las direcciones de memoria de la 10 a la 200.

*Los ejercicios marcados con **!** son un poco más complicados, los marcados con ***** sirven para ganarse un punto por aterisco en los parciales y los marcados con **+** son para ganarse un punto por cruz en el examen final. No se puede acumular más de 2 puntos por persona por parcial o por final.

[1] Cuando hablamos de memorias 1K=1024

(c) Compute el tiempo de acceso efectivo a memoria para este programa teniendo en cuenta que no se usa *load-through*.

6. Una caché de *mapeo completamente asociativo* tiene 16 ranuras para bloques de 8 palabras. La memoria principal es de 2^{16} palabras y la cache está vacía inicialmente. El tiempo de acceso a la caché es de 40ns y el tiempo requerido para transferir las 8 palabras desde la memoria principal a la ranura es de 1μs.

   (a) Calcule el tamaño de los campos *tag* y *word*.

   (b) Compute el porcentaje de aciertos para un programa que ejecuta la secuencia de instrucciones $20 \ldots 45(28 \ldots 45)^4$.

   (c) Calcule el tiempo de acceso efectivo a memoria, suponiendo nuevamente que no se utiliza *load-through*.

7. ! Explique como el caché de mapeo directo y el caché completamente asociativo pueden ser vistos como una particularización de un mapeo asociativo de conjunto de N elementos.

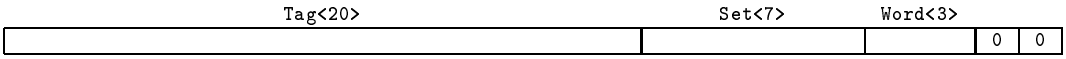
8. Dada una jerarquía de memoria con caché de mapeo directo de 2^{14} ranuras de 2^5 palabras y memoria principal de 2^{32} palabras.

   (a) ¿Cuán alejados deben estar los accesos a memoria para que todas causen una falla en la caché?

   (b) Usando el ejercicio (8a) calcule la taza de aciertos y el tiempo de acceso efectivo para un programa con $T_{miss} = 1000ns$ y $T_{hit} = 10ns$. Asuma que se opera utilizando *load-through*.

9. ! A primera vista, la presencia o ausencia de memoria caché no parece tener nada que ver con la decisión de ubicar registros de entrada/salida en el espacio de direcciones de memoria. Piense en los problemas potenciales que esto podría traer.

10. ! La arquitectura *ARC* se ve mejorada con la incorporación de una *caché unificada de instrucciones y datos* de mapeo directo con política de lectura *load-through*, que divide la dirección de memoria de 32 bits (la cual referencia bytes) de la siguiente manera:



Dado el siguiente programa:

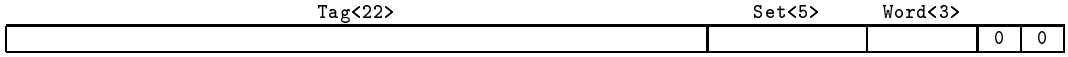
```
        .org  4096
        sethi 0, %r3
        add   %r0, 60, %r1
loop:   ld    %r1,%r2
        xor   %r3, %r2, %r3
        addcc %r1, -4, %r1
        bpos  loop
        jmpl  %r15+4, %r0
```

   (a) Obtenga la secuencia de accesos a la memoria, generado tanto por el *instruction fetching* como por las instrucciones de carga y almacenamiento.

   (b) Calcule la tasa de aciertos y el tiempo de acceso promedio de esta caché dado que $T_{miss} = 100ns$ y $T_{hit} = 10ns$. Suponga vacía la caché inicialmente.

   (c) !! ¿Cómo es posible mejorar la performance de este programa sin cambiar su semántica?

11. ! La arquitectura *ARC* se ve mejorada con la incorporación de una caché *unificada de datos y programas* asociativa de conjunto con asociatividad 3 –*3-way set associative*– que divide la dirección de memoria de 32 bits (la cual referencia bytes) de la siguiente manera:



(a) Calcule la capacidad en bytes de una ranura y de un conjunto. También compute la capacidad total de la caché en Kb y la longitud interna de una ranura completa.

(b) Obtenga una secuencia cíclica de accesos a la memoria que provoque máxima tasa de fallas bajo cualquier política de reemplazo. Explique.

(c) ¿Tiene sentido una caché asociativa de conjunto de 3 vías, o en general de n vías donde $\neg(\exists k \cdot n = 2^k)$? Explique.

12. + Intente mostrar como hacer *benchmarks tramposos*. Se tienen 2 máquinas con el mismo procesador y memoria principal, pero organizaciones de memoria caché diferentes. Asuma que el *miss time* es 10 veces el tiempo de un acierto para ambas máquinas. Asuma también que escribir una palabra de 32 bits toma 5 veces más que que un acierto de caché (para una caché *write-through*) y que escribir un bloque entero de 32 bytes toma 10 veces más que un acierto de caché (para el esquema *write-back*). El caché está unificado, es decir contiene instrucciones y datos.

Caché A 128 conjuntos, 2 elementos por conjunto, bloques de 32 bytes, con *write-through* y *no-write-allocate*.

Caché B 256 conjuntos, 1 elemento por conjunto, bloques de 32 bytes, con *write-back* y *write-allocate*.

(a) Describa un programa en ensamblador que haga correr la máquina A lo más rápido posible respecto a la máquina B.

(b) Idem pero con la máquina B ganándole a la A.

13. + En este ejercicio se correrá un programa para evaluar el comportamiento de un sistema de memoria. Un programa en C para UNIX que se encuentra para bajar en la página de la materia, nos permite hacer estas mediciones (`stride.c`).

En la primera parte este programa se presenta una función para obtener una medición precisa del tiempo. En la segunda parte, un arreglo se lee y escribe en rangos crecientes de longitud 2^n y por cada uno de estos rangos se hacen accesos de a saltos (*strides*) cada vez más grandes, desde 2^0 hasta 2^{n-1}. El código se repite varias veces a fin de obtener mayor precisión en la medición.

La tercera parte repite exactamente el mismo lazo pero no efectua operaciones con la memoria para poder medir y luego restar el tiempo de *overhead* que tiene la repetición. La parte final imprime los tiempos de acceso de la memoria para cada tamaño y salto.

Resulta importante asegurar que el compilador hace caso al modificador `register`, a fin de no interferir la memoria con lecto-escrituras de las variables que controlan los lazos.

(a) Grafique los resultados experimentales con el eje X indicando saltos de memoria y el eje Y los tiempo de acceso y una serie por cada tamaño de de caché. Use escala logarítmica para el eje X.

(b) ¿Cuántos niveles de caché hay en la jerarquía de memoria?

(c) ¿Cuál es el tamaño de la cache L1? ¿Cuál es su asociatividad?

(d) ¿Cuál es el tamaño de la cache L2 (si existe)? ¿Cuál es su asociatividad?

(e) ¿Hay algo más que usted haya descubierto acerca de la jerarquía de memoria a partir de estos experimentos?