Universidad de Valladolid
Departamento de Informática

Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

Fundamentos y Arquitectura de Computadores
I. T. Telecomunicación

Soluciones del examen extraordinario - 1 julio de 2003

Cuestión 1

Se define como "palabra de procesador" al tamaño de los registros de uso general de un procesador. Por otra parte, la "palabra de memoria" es la mínima unidad de información de la memoria que lleva asignada una dirección. Normalmente el tamaño de la palabra de memoria es de un byte, mientras que la palabra de procesador es un múltiplo potencia de dos del tamaño de la palabra de memoria, típicamente 2, 4 u 8.

Cuestión 2

$1828,25 = 2^x$; tomando logaritmos, $ln(1828,25) = x.ln(2)$; despejando x, tenemos que $x = \frac{ln(1828,25)}{ln(2)} = 10,83624765$. De este modo, $2^{10,83624765} = 2^{10}.2^{0,83624765} = 2^{10}.1,7854004$. Por lo tanto, la mantisa es $1,7854004_d = 1,11001001000100000000000_b$. El exponente es 10, es decir, 137_d en exceso a 127, o sea 10001001 en binario. El número es negativo: su signo es 1. Expresado en binario, el resultado es 1100 0100 1110 0100 1000 1000 0000 0000, o sea 0xc4e48800 en hexadecimal.

Solución alternativa: $1828,25_d = 11100100100,01_b$. Moviendo la coma diez posiciones a la derecha, tenemos que $1828,25_d = 1,110010010001_b.2^{10}$. Por lo tanto, el campo de mantisa será el siguiente: 11001001000100000000000. El exponente es 10, es decir, 137_d en exceso a 127, o sea 10001001_b. El número es negativo: su signo es 1. Expresado en binario, el resultado es 1100 0100 1110 0100 1000 1000 0000 0000, o sea 0xc4e48800 en hexadecimal.

Cuestión 3

Algoritmo de sustitución aleatoria : se selecciona un bloque al azar para su desalojo. Fácil de implementar.

Algoritmo FIFO (*first in, first out*): Se desaloja el bloque que lleve más tiempo en la cache. Utiliza un "contador de edad": cada vez que se introduce un bloque se pone su contador de edad a 0 y se incrementan los contadores de edad del resto de los bloques de ese conjunto. El contador de edad tiene un número de bits b tal que se cumpla que $2^b =$vias.

Algoritmo LRU (*least recently used*): Utiliza el principio de localidad de referencia. Se desaloja el bloque que lleve más tiempo sin ser utilizado. Se utiliza un contador de edad que se actualiza cada vez que se accede a un bloque, no sólo cuando se introduce el bloque en la cache.

Universidad de Valladolid
Departamento de Informática
Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

Cuestión 4

Un sumador con acarreo anticipado calcula las siguientes variables:

$$G_i = a_i b_i \text{ (Variable generada)}$$
$$P_i = a_i \oplus b_i \text{ (Variable propagable)}$$

En función de estas dos variables, las dos ecuaciones para s_i y c_{i+1} pueden escribirse como

$$s_i = P_i \oplus c_i$$
$$c_{i+1} = a_i b_i + (a_i + b_i)c_i = G_i + P_i c_i$$

Un sumador con acarreo anticipado realiza la suma en tres pasos:
(1) Las células sumadoras calculan, todas a la vez, las variables P_i y G_i.
(2) Con todos los valores de P_i y G_i ya disponibles, se calculan todos los acarreos.
(3) Con los acarreos y los valores de a_i y b_i se calculan simultáneamente todos los bits de suma.

Problema 1

En la figura 1 aparece el esquema de la memoria requerida. Por simplicidad, no se han conectado entre sí las líneas R y W, que van conectadas a todos los módulos de memoria que posean esa conexión.

Problema 2

El tamaño de la matriz es de 16 x 64 = 1024 elementos. Como cada elemento ocupa cuatro bytes, la matriz completa ocupa 4096 bytes. Por otra parte, la cache tiene cuatro marcos de bloque de 1024/4 = 256 bytes. Dado que en C las matrices se almacenan por filas en memoria, en un bloque de esta cache cabe exactamente una fila de la matriz.

La primera versión accede a la matriz por filas. Por lo tanto, si la cache está inicialmente vacía, la lectura del elemento `matriz[0][0]` provocará un fallo de bloque. El bloque obtenido se traerá al marco de bloque 0 de la matriz, que según el enunciado es el que le corresponde. Como una fila cabe en un bloque, se traerá toda la fila. En consecuencia, el acceso al elemento `[0][1]` y a los restantes elementos de la fila serán aciertos cache: harán falta $100 + 63 = 163$ ciclos para recorrer esa fila.

Como una fila cabe exactamente en un marco de bloque cache, el acceso al primer elemento de la fila siguiente (la fila uno) provocará de nuevo un fallo que se corresponde con el marco de bloque número uno de la cache, y aciertos en los accesos a los siguientes elementos de esa fila. Los accesos a la fila dos y tres provocarán fallos en los marcos de bloque dos y tres, respectivamente, con los mismos tiempos de acceso.

El acceso a la fila cuatro, por su parte, se corresponde con el marco de bloque cero de la cache. Como en ese marco está cargada la fila cero, esto provoca un fallo cache en la primera lectura, y aciertos en las

Universidad de Valladolid
Departamento de Informática
Escuela Universitaria Politécnica c/Francisco Mendizábal,1 47014 Valladolid Tel.:(983) 423501-02 Fax:(983) 423490

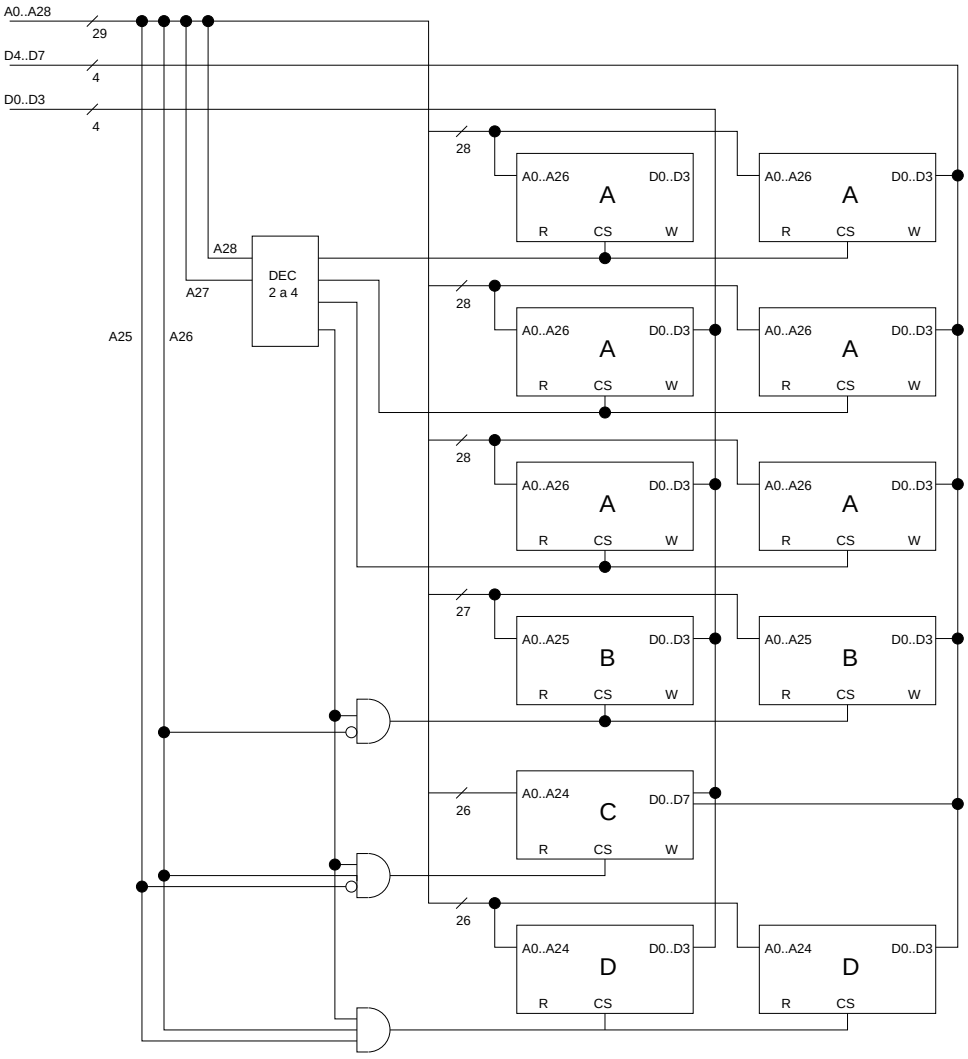

Figura 1: Conexión de los módulos de memoria del problema 2.

siguientes accesos a elementos de la misma fila: otros 163 ciclos. Puede comprobarse que el acceso a las siguientes filas también provocarán fallos al leerse el primer elemento y aciertos en los siguientes. En consecuencia, el tiempo total necesario para acceder a los elementos con la primera versión del programa será de $16.(1.100 + 63.1) =$ **2608 ciclos**

En lo que respecta a la segunda versión, el acceso al elemento `matriz[0][0]` provocará un fallo que traerá la fila cero de la matriz al marco de bloque cero de la cache. El siguiente acceso, que será al elemento `matriz[1][0]`, accederá a la segunda fila. Como esta fila no está en la cache, se provoca un nuevo fallo de bloque. Los accessos a los elementos `[2][0]` y `[3][0]` provocarán nuevos fallos. En este momento la cache está llena con las cuatro primeras filas. A continuación se accederá al elemento `[4][0]`, esto es, al primer elemento de la quinta fila, que como hemos indicado más arriba se corresponde con el marco de bloque número cero de la cache. Como en ese marco de bloque está cargada la fila cero, este acceso provocará un nuevo fallo cache. Lo mismo pasará con el elemento `[5][0]` con respecto al marco de bloque uno, etcétera.

En consecuencia, en esta segunda versión del programa cada acceso a un elemento de la matriz provoca un fallo de bloque. El tiempo total de acceso a la matriz será, por lo tanto, de $16.64.100 =$ **102400 ciclos**: la segunda versión tarda casi 40 veces más que la primera en leer los elementos de la matriz.