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 ordinario - 14 febrero de 2003

Problema 1

Escribir una rutina en ensamblador IA-32 que reciba por la pila dos números naturales A y B de ocho bits encapsulados en un único parámetro de 16 bits y calcule $A * B$ sin utilizar la instrucción de multiplicación. Escribir también el fragmento de código encargado de llamar a la rutina, pasándole como parámetro los valores $A =$`0xfe` y $B =$`0x04`.

```
rutina: pop bx       ; Se destapa la pila.
        pop ax       ; Se extrae el parámetro.
        push bx      ; Se tapa la pila.
        mov cl, ah   ; Se mueve uno de los parámetros a cl.
        mov ah, 0
        mov ch, 0    ; Se limpian ah y ch.
        mov dx, 0    ; dx será el acumulador de resultados parciales.
        cmp cx, 0
        jz final     ; Si cx = 0, el producto dx será 0.
bucle:  add dx, ax
        loop bucle   ; Se itera hasta que cx valga cero.
final:  pop bx       ; Se destapa la pila.
        push dx      ; Se guarda el resultado.
        push bx      ; Se tapa la pila.
        ret
```

El fragmento de programa principal que invoca a la rutina será el siguiente:

```
        ...
        mov al, 0xfe
        mov ah, 4
        push ax
        call rutina
        pop dx       ; El resultado está ahora en dx.
        ...
```

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

Problema 2

1. Indicar razonadamente el tamaño de cada uno de los campos en los que cada cache divide las direcciones de memoria.

- Cache L1 de instrucciones: las direcciones serán de 32 bits, ya que la memoria tiene 4Gb. El campo de palabra tendrá 6 bits, lo que permitirá direccionar 64 palabras distintas. La cache tiene $2^{15}/2^6 = 2^9$ bloques, esto es, 512 bloques, por lo que el tamaño del campo de bloque será de 9 bits. El resto (32-9-6=17 bits) serán los bits de etiqueta.
- Cache L1 de datos: Direcciones de 32 bits, campo de palabra de 6 bits (igual que antes). La cache tiene $2^{14}/2^6 = 2^8 = 256$ bloques. Como tiene dos vías, esto hace 128 conjuntos, por lo que el campo de conjunto tendrá 7 bits. El resto (32-7-6=19 bits) serán de etiqueta.
- Cache L2: Direcciones de 32 bits, campo de palabra de 6 bits (igual que antes). L2 tendrá $2^{21}/2^6 = 2^{15}$ bloques. Es completamente asociativa: la etiqueta ocupará 32-6=26 bits.

2. ¿Qué ventajas e inconvenientes tiene la correspondencia asociativa frente a las otras dos funciones de mapa? Relacione su respuesta con el problema planteado.

La correspondencia asociativa permite evitar conflictos en el almacenamiento de dos bloques al que les correspondería el mismo marco de bloque cache (como sucede en la correspondencia directa), o entre varios bloques que se mapean a un mismo conjunto sin poder guardarse todos simultáneamente en él (como en la correspondencia asociativa por conjuntos). A cambio, la búsqueda de un bloque es más lenta, ya que hay que buscar la etiqueta entre todas las etiquetas de la cache, no sólo entre las de un conjunto concreto. Esto hace que el funcionamiento de la cache sea algo más lento. En el problema planteado, la caché L2 sólo actúa cuando se produce un fallo en L1. Teniendo en cuenta las tasas habituales de acierto de L1 (90%.. 95%), esto significa que se accede poco a L2, por lo que el retraso que supone la búsqueda de las etiquetas no representa un inconveniente grave.

3. En la dirección `0x0030cafe` *se encuentra la siguiente instrucción:* `mov ax, [0x1234]`*. Suponiendo que las caches están frias y que la instrucción antedicha ocupa tres bytes, indicar cómo afecta la ejecución de esta instrucción al sistema de caches.*

La ejecución de la instrucción tiene dos fases: la de búsqueda y la de ejecución. En la de búsqueda se trae a la CPU la instrucción: en nuestro caso, esto obliga a realizar tres lecturas de un byte sobre las direcciones `0x0030cafe`, `0x0030caff` y `0x0030cb00`. En la de ejecución, hay que leer los bytes situados en `0x00001234` y `0x0001235` y guardarlos en AX. Por lo tanto, la ejecución de esta instrucción obliga a realizar cinco accesos a memoria. Veremos la evolución de la cache para cada uno de ellos:

1. Acceso a `0x0030cafe`. Es un acceso a una instrucción, por lo que interviene la cache L1 de instrucciones. Se divide la dirección en campos:

0000 0000 0011 0000 1100 1010 1111 1110$_b$ = 00000000001100001 100101011 111110$_b$.

Bloque 100101011$_b$ = `0x12b`. El marco de bloque `0x12b` de L1 está vacío: fallo cache. Se pasa la solicitud a L2, solicitando todo el bloque. L2 divide la dirección en la que empieza el bloque solicitado según sus campos:

0000 0000 0011 0000 1100 1010 1100 0000$_b$ = 00000000001100001100101011 000000$_b$.

El bloque se busca asociativamente en L2, pero no se encuentra, ya que está la cache vacía, por lo que se trae el bloque de la memoria. Una vez traido y almacenado en L2, se pasa una copia a la cache L1 de instrucciones, que lo almacena en el marco de bloque `0x12b`, actualizando la etiqueta. Esa cache recupera la palabra situada en el byte 111110$_b$ `0x3e` y la entrega a la CPU.

2. Acceso a `0x0030caff`. Es un acceso a una instrucción, por lo que interviene la cache L1 de instrucciones. Se divide la dirección en campos:

   0000 0000 0011 0000 1100 1010 1111 1111$_b$ = 00000000001100001 100101011 111111$_b$.

   Bloque 100101011$_b$ = `0x12b`. El marco de bloque `0x12b` de L1 contiene el dato: acierto cache. L1 recupera la palabra situada en el byte `0x3f` y la entrega a la CPU.

3. Acceso a `0x0030cb00`. Es un acceso a una instrucción, por lo que interviene la cache L1 de instrucciones. Se divide la dirección en campos:

   0000 0000 0011 0000 1100 1011 0000 0000$_b$ = 00000000001100001 100101100 000000$_b$.

   Bloque 100101100$_b$ = `0x12c`. El marco de bloque `0x12c` de L1 está vacío: fallo cache. Se pasa la solicitud a L2, solicitando todo el bloque. L2 divide la dirección en la que empieza el bloque solicitado según sus campos:

   0000 0000 0011 0000 1100 1011 0000 0000$_b$ = 00000000001100001100101100 000000$_b$.

   El bloque se busca asociativamente en L2, pero no se encuentra, por lo que se trae de la memoria. Una vez traido y almacenado en L2, se pasa una copia a la cache L1 de instrucciones, que lo almacena en el marco de bloque `0x12c`. Esa cache recupera la palabra situada en el byte `0x00` y la entrega a la CPU.

4. Acceso a `0x00001234`. Es un acceso a datos, por lo que interviene la cache L1 de datos. Se divide la dirección en campos:

   0000 0000 0000 0000 0001 0010 0011 0100$_b$ = 00000000000000000000 1001000 110100$_b$.

   El campo de conjunto vale 1001000$_b$ = `0x48`. Se busca el bloque en los dos marcos de bloques que forman ese conjunto. Como no se encuentra, se solicita todo el bloque a la cache L2.

   0000 0000 0000 0000 0001 0010 0000 0000$_b$ = 00000000000000000001001000 000000$_b$.

   El bloque se busca asociativamente en L2 y tampoco se encuentra: fallo L2. Se trae el bloque de la memoria principal, se almacena en L2 (actualizando la etiqueta correspondiente) y se envía copia a L1 de datos, que lo almacena en uno de los marcos de bloque del conjunto `0x48`. Se devuelve la palabra 110100$_b$ = `0x34` a la CPU.

5. Acceso a `0x00001235`. Es un acceso a datos, por lo que interviene la cache L1 de datos. Se divide la dirección en campos:

   0000 0000 0000 0000 0001 0010 0011 0101_b = 00000000000000000000 1001000 110101_b.

   El campo de conjunto vale 1001000_b = `0x48`. Se busca el bloque entre los bloques almacenados en ese conjunto. Se produce un acierto cache (el bloque es el mismo que el que se accedió en la referencia anterior.) Se devuelve la palabra 110101_b = `0x35` a la CPU.

Cuestión 1

De entre la representación binaria de signo-magnitud y la de complemento a uno, ¿cuál es más apropiada para el cálculo automático y por qué?

Aunque ambos sistemas tienen el mismo rango de representación, $[2^{(n-1)} + 1, 2^{(n-1)} - 1]$, la ventaja del complemento a uno es que no es necesario tratar el bit de signo de forma separada en las operaciones aritméticas, ya que la suma de dos números en complemento a uno está en complemento a uno, siempre que no se provoque un desbordamiento.

Cuestión 2

Defina el concepto de ruta de datos y su relación con la Unidad de Control.

La ruta de datos es el conjunto de todos los caminos que pueden seguir los datos dentro de la CPU. Incluye los registros de uso general, los de uso específico, la unidad aritmético-lógica y los buses que conectan todos estos elementos entre sí. La Unidad de Control se encarga de regular los movimientos de datos entre todos los elementos que componen la ruta de datos, con el fin de ejecutar una a una las instrucciones que componen un programa en ensamblador.

Cuestión 3

Enumere las principales diferencias que existen entre las arquitecturas Intel IA-32 y MIPS.

- IA-32 es un juego de instrucciones CISC (muchas instrucciones, muchos modos de direccionamiento) mientras que MIPS es RISC (pocas instrucciones, pocos modos de direccionamiento).
- IA-32 utiliza un tamaño variable de palabra de instrucción, lo que complica su decodificación; MIPS utiliza un tamaño de palabra fijo.
- IA-32 utiliza una unidad de control microprogramada debido a la complejidad de su juego de instrucciones; MIPS utiliza una unidad de control cableada, mas rápida y menos compleja.

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

Cuestión 4

¿En qué se basan los algoritmos de Booth y de Booth modificado para acelerar las operaciones de multiplicación?

Se basan en la "propiedad de la cadena": un número binario que tenga una cadena de unos entre las posiciones i y j puede reescribirse como la resta de un número que tenga un 1 en la posición $i + 1$ menos otro que tenga un 1 en la posición j. Esto permite acelerar el algoritmo de suma y desplazamiento, ya que en lugar de tener que hacer una suma por cada 1 de la cadena, se hace una resta al principio y una suma al final. El algoritmo de Booth modificado mejora el anterior para el caso de que aparezcan 1s aislados.