Parte II

ARQUITECTURA DE COMPUTADORES

3º, IST, ITT, IT y IT-ADE URJC

ARQUITECTURA DE SISTEMAS AUDIOVISUALES II

4º SAM, URJC

Fuenlabrada, 13 de Diciembre de 2016

IMPORTANTE:

- No se pueden utilizar libros ni apuntes
- Todas las cuestiones deberán responderse de forma razonada

Problema 1 (2,5 puntos)

El siguiente código se ejecuta en el nanoMIPS segmentado.

```
1    loop:     lw      $t1, 0($t1)
2              lw      $t1, 0($t1)
3              subi    $t3, $t3, 1
4              beq     $t3, $zero, else
5              sw      $t1, A($t3)
6              beq     $zero, $zero, end
7    else:     lw      $t4, 0($t0)
8              add     $t2, $t1, $t4
9              sw      $t2, B($t3)
10    end:     addi    $t0, $t0, 4
11             bne     $t0, $s0, loop
```

Señala dependencias tipo RAW, entre qué instrucciones se producen y el número de parones generados, tanto por dependencias RAW como de otro tipo. Si suponemos que ya se han realizado los ciclos de llenado, calcula el CPI para las dos posibles secuencias de instrucciones según se cumpla o no la condición de la instrucción 4 (1 punto). Explica por qué se deben introducir exactamente ese número de parones en las dependencias RAW (0,5 puntos). Recalcula el CPI para las dos secuencias de código si ahora introducimos adelantamiento para datos y control (1 punto).

Problema 2 (2'5 puntos)

Se diseña una jerarquía de memoria con un único nivel de memoria caché, con instrucciones y datos separados. El 78% de las referencias a datos son lecturas y 22% escrituras. La memoria caché de instrucciones tiene un tiempo de acceso 1,1 ns, una tasa de fallos del 6% y el tamaño del bloque es de 24 palabras. La memoria caché de datos tiene un tiempo de acceso de 1,3 ns, una tasa de fallos del 9% y un tamaño de bloque de 16 palabras. La latencia de memoria principal es de 94 ns. Completa la siguiente tabla indicando el tráfico con el siguiente nivel de la jerarquía consumido si se trata de una caché de escritura directa (con ubicación) que incluye un buffer de escritura con una tasa de acierto del 93%. (1,5 puntos).

Tipo de acceso	Acciones	Tráfico siguiente nivel

Calcula el tiempo medio de acceso a dicha jerarquía de memoria (1 punto).

EXAMEN PRÁCTICAS

ARQUITECTURA DE COMPUTADORES

3º ST, TT, T y Teleco-ADE - URJC

Fuenlabrada, 2 de Diciembre de 2016

Se pide implementar (respetando el convenido de llamada a subrutina) el programa **lista_ordenada.asm** que crea una lista simplemente enlazada en la que todas las inserciones se hacen de mayor a menor valor. Por ejemplo, las inserciones de los nodos con valores 40, 12 y 50 tiene como resultado una lista en la que el primer nodo de la lista es el 50, después le sigue el 40 y por último el 12 (que apunta a null).

En el `main` se crea el primer nodo fuera del bucle y después, dentro del bucle, se realiza **`insert_in_order`** N veces (hasta que se introduzca un 0). Después, se debe imprimir la lista de menor a mayor, es decir, en nuestro ejemplo, se imprimen los valores 12, 40 y 50. El registro **$s0** se utilizará en el `main` para apuntar al primer nodo de la lista. `main` debe mantener actualizado el valor de **$s0** con los valores que le devuelve **`insert_in_order`** . Así, si **`insert_in_order`** le devuelve un 0 (un puntero nulo), significa que el primero de la lista sigue siendo el mismo. Por el contrario, si le devuelve una dirección distinta de cero, entonces será la del nodo que ahora pasa a ser el primero de la lista.

En C, la estructura de datos para representar un **nodo** de la lista sería la siguiente:

```
typedef struct _node_t {
      int val;              /* valor del nodo; tamaño palabra */
      struct _node_t *next; /* puntero al nodo siguiente */
} node_t;
```

El nodo siguiente (“next”) al último nodo de la lista es NULL. Se pide implementar tres funciones:

(obligatoria)`node_t * create(int val, node_t *next)`: crea un nuevo nodo e inicializa los campos del nodo con los argumentos recibidos. Devuelve la dirección del nuevo nodo creado.

(obligatoria)`node_t * insert_in_order(node_t *first, int val)`: recibe como parámetros la dirección del primer nodo de la lista y el valor del nodo a añadir. Crea un nuevo nodo por medio de la subrtuina **`create`** con el valor indicado y lo inserta en orden, de tal forma que los elementos de la lista queden ordenados de mayor a menor valor. Devuelve un puntero nulo si el primer nodo de la lista sigue siendo el mismo. Por el contrario, devuelve la dirección del nuevo nodo insertado cuando éste pasa a ser el primero de la lista.

(opcional)`void print(node_t *first)`: recorre la lista de forma **recursiva** imprimiendo los elementos de menor a mayor valor. Algoritmo:

```
if (first->next =! null) {
      print(first->next);
}
printf(“%d\n”, first->val);
return;
```

EXAMEN PRÁCTICAS

ARQUITECTURA DE SISTEMAS AUDIOVISUALES II

4º SAM, URJC

Fuenlabrada, 7 de Diciembre de 2016

Se pide implementar (respetando el convenido de llamada a subrutina) el programa **pila.asm** que crea una lista lineal en la que todas las inserciones se hacen por un extremo de la lista, la cima. Este tipo de lista recibe también el nombre de lista *LIFO* (Last In First Out). Por ejemplo, si se introducen los nodos con valores 34, -2 y 50, la cima de la pila sería el número 50, que apuntaría al nodo -2, etc. El nodo 34 apuntaría a null.

En el `main` se debe realizar `push` N veces (hasta que se introduzca un 0). Después, se debe eliminar un nodo con el valor que indique el ususario e imprimir el valor del nodo eliminado. Por último, se debe imprimir la pila resultante. El registro **$s0** se utilizará en el `main` para apuntar a la cima de la pila. `main` debe mantener actualizado el valor de **$s0**.

En C, la estructura de datos para representar un **nodo** de la lista sería la siguiente:

```
typedef struct _node_t {
      int val;              /* valor del nodo; tamaño palabra */
      struct _node_t *next; /* puntero al nodo anterior */
} node_t;
```

Se pide implementar tres funciones:

(obligatoria)node_t * create(int val, node_t *prev): crea un nuevo nodo e inicializa los campos del nodo con los argumentos recibidos. Devuelve la dirección del nuevo nodo creado.

(obligatoria)node_t * push(node_t *top, int val): recibe como parámetros la dirección del nodo situado en la cima de la pila y el valor del nodo a añadir. Crea un nuevo nodo con la función **create** y lo inserta en al cima de la pila. Devuelve la dirección del nuevo nodo creado.

(obligatoria)node_t * remove(node_t *top, int val): recibe como parámetro la dirección del nodo en la cima de la pila. Recorre la pila buscando el nodo con valor "val". Si lo encuentra, lo elimina de la pila: el nodo anterior al buscado apuntará ahora al nodo siguiente (next) al nodo buscado. Además, devuelve la dirección del nodo buscado. Si no lo encuentra, devuelve un null. Se supone que NO SE PUEDE ELIMINAR EL NODO CIMA.

(opcional)void print(node_t *top): recorre la lista de forma **recursiva** imprimiendo los elementos en orden temporal en el que fueron introducidos en la pila. Algoritmo:

```
if (top->next =! null) {
      print(top->next);
}
printf("%d\n", top->val);

return;
```