Práctica 2 IO

Carlos Raso Alonso

Índice

Introducción y objetivos

En esta práctica se pretende que nos familiaricemos más con el simulador Ripes y su funcionamiento, así como con sus diversas partes como la memoria o las instrucciones usadas en él. También debemos entender de qué registros dispone el simulador y para qué se utiliza cada uno. Por ejemplo el registro x0 está siempre fijado a 0. En la práctica también vemos una manera de multiplicar números enteros en este simulador utilizando bucles y creamos programas que realizan diversas tareas, viendo cómo se deben manejar las instrucciones para conseguirlo.

1

- A5 vale ch, el resultado de sumar el primer y segundo elemento del vector datos
- A6 vale 4h, el resultado de sumar el segundo y cuarto elemento del vector datos
- A3 vale ffffffffh
- la instrucción bnez lo que hace es ejecutar la función bne tomando como uno de sus registros de lectura automáticamente x0, que, como sabemos, tiene un valor fijado de 0. Por lo tanto bnez es una instrucción de tipo branch con las características de este grupo de operaciones.

  La instrucción nop tiene el mismo op code y func3/7 que una suma con un inmediato. Esto es debido a que la acción que se ejecuta en esta instrucción es x0 <= x0 + 0. El registro x0 es un alambrado a 0 y no se puede alterar con esta instrucción por lo que el registro x0 permanece invariable (sigue valiendo 0) y lo único que cambia esta instrucción es el registro pc, al que se le suma 4 como en cualquier otra instrucción.
- los valores de Resultados se encuentran en las posiciones de memoria 10000010h y 10000010h respectivamente (las posiciones de los dos elementos del vector Resultados ordenados) se guarda en la posición a0 + 20 que corresponde a la posición 10000014h de memoria puesto que a0 está durante todo el programa en la primera posición de la sección de memoria .data, es decir, en la posición 10000000h. Esta posición es la posición de memoria que al inicio del programa se había asignado al segundo elemento del vector Resultados. El contenido de esta posición de memoria queda sobrescrito con el nuevo número.
- El único registro afectado al ejecutar bgez es el registro pc , al que se le suma 4 si la condición no se cumple, es decir si a3 < 0 o cambia a la posición con la etiqueta loop si la condición sí se cumple, es decir, si a3 >= 0. Para cambiar a la etiqueta loop lo que se hace es restar 12 al registro pc. El registro a3, como hemos visto, solo es consultado para realizar esta instrucción, pero no alterado.

2

- Está realizando la multiplicación de los dos números iniciales, plasmando el resultado en a0, ya que suma en a0 el número a1 a2 veces (el loop se repite un número a2 de veces)

- a2 acaba valiendo 0 tras haber realizado todos los bucles, a1 mantiene su valor ya que ninguna instrucción se lo altera y a0 finalmente alberga el resultado de la operación a1 * a2 = 3083 * 17 = 52411 = ccbbh
- El bucle se ejecuta a2 veces, es decir, 17 veces

3

- Al principio mi programa almacena en la sección .data de memoria los dos números con los que se va a trabajar en las dos primeras posiciones de memoria en un vector de dos palabras con etiqueta “datos”. Al principio del programa, en la etiqueta main, primero se carga la dirección del vector datos en el registro a1, luego se carga el contenido de la primera dirección de memoria de “datos” en el registro a2 y se hace lo propio con el contenido de la segunda dirección de memoria del vector “datos” y el registro a3. Estas dos acciones se llevan a cabo con la instrucción lw (load word) y usando como dirección de la que se toma el contenido a1 con un inmediato de 0 ó 4 según se quiera acceder a la primera posición del vector (0) o a la segunda (4). Posteriormente, en el main, tenemos tres instrucciones de tipo branch seguidas. En estas instrucciones se comprueba cuál de los dos números con los que se va a operar es mayor o si son iguales, es decir, si a2 < a3, a2 = a3 o a2 > a3. Según cuál de estas condiciones se cumple, se salta a la etiqueta de instrucciones op_eq, op_gt o a op_lt.

  Para conseguir esto se opera con tres instrucciones de branch que realizan las tres comparaciones indicadas anteriormente entre a2 y a3 y, en cada una de ellas, si se cumple la condición, se salta a la etiqueta de instrucciones correspondiente. Por ejemplo en la primera instrucción de este tipo: beq a2, a3, op_eq, se comprueba si a2 == a3, en cuyo caso, se salta a la instrucción con etiqueta op_eq. Si a2 != a3 se procede con la siguiente instrucción.

  Una vez terminado el main hay tres etiquetas en las que puede estar el pc: op_eq, op_gt o op_lt. Las tres secciones tienen una estructura similar en las dos siguientes instrucciones que duran: en la primera instrucción se hace la operación correspondiente con a3 y a2 utilizando mul, sub o add según en que operación estemos y se almacena el resultado en el registro a0. Luego tenemos un salto incondicional a la etiqueta end_op (excepto en la última operación que no necesita este salto por ser la etiqueta end_op la instrucción inmediata). He elegido realizar la multiplicación con la instrucción mul dado que de esta manera se realiza la operación con una sola instrucción y, por tanto, un solo ciclo de reloj. En caso de haberla hecho con un bucle, como en el ejercicio anterior, se hubieran necesitado muchos más ciclos de reloj, dependiendo de los valores de a2 y a3 por lo que hubiera sido menos eficiente.

  Por último, tras llegar a la etiqueta end_op, se almacena el valor del registro a0 en memoria, concretamente 24h posiciones después de la primera posición de la sección .datos de memoria, como se indica en el enunciado. Para esto se usa la instrucción sw y se guarda en la dirección a1 + 24h = a1 + 36 (en el registro a1 estaba guardada la primera posición de memoria de la sección .datos).

  En la siguiente página hay una imagen en la que se puede ver el código sin ensamblar del programa.

```
.data
datos: .word  4, 5
.text
main:
    la a1, datos
    lw a2, 0(a1)
    lw a3, 4(a1)
    beq a2, a3, op_eq
    blt a2, a3, op_lt
    bgt a2, a3, op_gt
op_eq:
    mul a0, a1, a3
    j end_op
op_lt:
    sub a0, a2, a3
    j end_op
op_gt:
    add a0, a1, a3
end_op:
    sw a0, 36(a1)
```

Conclusiones

En esta práctica he entendido mejor cómo funcionan los registros del simulador y su memoria, así como las instrucciones que los relacionan. También he conseguido manejar las instrucciones (especialmente las de salto o branch) de tal modo que pueda crear condiciones de salto del flujo de instrucciones para conseguir que el programa de distintos resultados según los valores que recibe. Además he aprendido lo que hacen algunas instrucciones o cómo las interpreta el simulador en código máquina. Por ejemplo he aprendido qué hace nop y qué tipo de instrucción es y he visto cómo se ejecutaba la función bnez. También he profundizado en mi conocimiento del registro pc a través de los saltos que tenía que realizar en alguno de los programas.