Introducción

El presente trabajo tiene como objetivo predecir el funcionamiento de un programa MIPS, traducir programas escritos en lenguaje de alto nivel a MIPS e introducir al estudiante en lo que es la programación en la arquitectura MIPS, perteneciente a la familia RISC (reduced inttruction set computer), el cual gracias a su gran numero de registros, el número y el carácter de las instrucciones permite a esta arquitectura entregar un gran desempeño.

Desarrollo

El trabajo consta de 2 partes, la primera en la cual se analizará si los resultados obtenidos en el llamado "trabajo de prelaboratorio" fueron correctos utilizando el simulador MARS, el cual es un IDE para MIPS, los resultados obtenidos en el prelaboratorio fueron hechos a mano intentando predecir cual sería el comportamiento, por lo cual en esta sección se comprobará si la predicción fue acertada.

La segunda parte traducir programas que han sido escritos en lenguaje de alto de nivel a MIPS, utilizando el simulador MARS

Resultados

Parte 1

1. Los resultados obtenidos en el primer programa del prelaboratorio fueron predichos de manera correcta, dentro del simulador MARS podemos ver a los registros 8 y 9 ($t0 y $t1 respectivamente) que cuentan con los valores 0xfffffffc para $t0 y 0xfffffff8 para $t1, los que al pasarlos a decimal podemos ver que son -4 y -8 respectivamente.
2. Los resultados obtenidos en la primera parte cuando $t2 almacena un 2 fueron correctos, en concreto, $t0 termina en 2 y $t1 termina en 1, tal como fue previsto en el prelaboratorio. Para cuando $t2 almacena un 0, los resultados obtenidos en el prelaboratorio también fueron correctos, en concreto la diferencia con la sección anterior es que en la línea 3 al momento de hacer el "beq", este se cumple y da un salto a la sección de código A.
3. Los resultados obtenidos en la tercera parte fueron en su mayoría correctos, si bien es cierto, $t1 y $t2 terminaron con valor 5, como fue previsto, $t0 termino con valor 4, lo cual difiere de mi predicción, por otra parte, las direcciones de memoria 0x10010000 y 0x10010004 terminaron como fue predicho, con valor 5.

Los resultados de estas pruebas serán adjuntados en la prueba como archivos para su revisión

Parte 2

1. Los resultados obtenidos al traducir el programa a MIPS fueron los siguientes: $t0 termino con valor 20, $t1 termino con valor 25 y $t2 termino con valor 8. Adicionalmente cabe recalcar que se utilizo los registros $t3 y $t4 para realizar las operaciones intermedias de $t0 + 4 y $t1 – 9.

2. Utilizando la función beq (brench if equal), podemos hacer uso de funciones if y else dentro del código en MIPS, además, se hace uso de la función syscall en el modo 10 el cual permite terminar la ejecución del programa, esto para que al momento de hacer un salto a una sección, no se ejecute la sección siguiente. Para el caso en que $t3 almacena un 1, el valor de $t4 es 0. En el caso en que $t3 almacene un 2, el valor de $t4 es -2 y finalmente para cualquier otro valor, el valor de $t4 es 100.
3. Traduciendo este programa vemos el uso de sw (store word) y lw (load word), funciones las cuales nos permite cargar o guardar valores en distintas posiciones de memoria, para ello siempre hay que tener cuidado que la distancia entre cada dirección de memoria es de 4 bytes (32 bits). Como se esperaba, las direcciones de memoria 0x10010000(+0) y 0x10010000(+c) terminan conteniendo los valores 3 y 2 respectivamente, lo cual calza con el código C de alto nivel.
4. Para este programa, se solicita que se deba crear un ciclo while, para ello, hacemos uso de los saltos de bloque, en este caso, se hace un bloque de código solamente para hacer la pregunta la cual mantendría el ciclo while (a > 0), de ser esto correcto, se mueve al bloque correspondiente a la ejecución del while, al término de dicho ciclo se hace uso de la función j (jump) para volver a preguntar si se puede seguir haciendo un while. En parte de los resultados fueron los siguientes: $t6 termino con valor 0, $t7 termino con valor 10 y $t0 termino con valor 20, es decir, se hicieron 2 ciclos.
5. Para este programa primero tendremos que dejar en claro algunas cosas, como por ejemplo que es lo que hace arr[i] & 1, & es un operador binario el cual compara cada bits de un número, en concreto en este caso, toma en consideración solamente los números pares, puesto que todos los impares terminan en 1, ahora, para que $s1 obtenga el largo del array, utiliza operaciones hexadecimales, primero se guarda el valor de memoria del arreglo en un registro, se hace una resta y al momento de hacer la operación srl (shift right logical) el valor terminaría en 100, lo que es igual a 4. Para la resolución de este problema lo primordial fue hacer la carga de arr[i] en un registro temporal, para ello el for original en vez de ser incrementado en 1, este debe ser incrementado en 4, para respetar las distancias entre alocaciones de memoria. Finalmente, para cuando los valores de arr son igual a 10, 22, 15, 40, el resultado almacenado en $t0 termina siendo 72, puesto que solo se suman los valores pares.

Conclusión

Para finalizar, se concluye que la arquitectura MIPS posee una gran variedad de herramientas, que incluso puede llegar a ser equiparable a lenguajes de alto nivel, solo que para hacer uso de estas herramientas se debe usar mucho más ingenio. El uso de la memoria es primordial al momento de programar en MIPS y queda en claro por ejemplo al momento de buscar “las posiciones de un arreglo” puesto que se sabe que las distancias entre direcciones son de 4 bytes, y esto hay que tenerlo siempre en consideración para no producir errores. Personalmente me agrada bastante este tipo de programación, me recuerda a métodos de programación cuando se vio máquina de Turing, donde se requiere hacer muchos saltos dentro del código para así crear ciclos.