Universidad Carlos III
Grado en ingeniería informática
Curso Estructura de Computadores 2023-24
Práctica 2
Curso 2023-24

Fecha: **03/12/2023** - ENTREGA: **2**

GRUPO: **81**

Alumnos: **Mario Ramos Salsón (100495849) y Miguel Fidalgo García (100495770)**

ÍNDICE

EJERCICIO 1

NOMBRE	RT	SEÑALES	DISEÑO
la RR1, U32	MAR <- PC MBR <- MEM[MAR] PC <- PC + 4 REG <- MBR	T2, C0 Ta, R, BW=11, M1, C1, M2, C2 T1, LC, SelC = 10101, A0=1, B=1,C=0	Para él “la” hemos jugado con el IR para leer la primera palabra y con el PC que es donde está la dirección de memoria asociada a la segunda palabra. Por ello vamos con el PC memoria para sacar la dirección de memoria que queremos.
sc r1 r2 r3	MAR <- R3 MBR <- R1 MEM[MAR] <- MBR MAR <- R3 + 4 MBR <- R2 MEM[MAR] <- MBR	SelA = 01011, T9, C0 SelA = 10101, T9, C1 Ta, W, BW=11, Td SelA = 01011, MB=10, SelCop=01010, MC=1, T6, C0 SelA = 10000, T9, C1 Ta, W, BW=11, Td, A0=1, B=1,C=0	Para el “sc” hemos guardado primero el registro 1 en memoria usando la dirección de memoria guardada en r3. Después pasamos por la ALU el valor de r3 para sumarle 4 y almacenar el registro 2 en la siguiente dirección de memoria, para así tenerlos alineados.
lc r1 r2 r3	MAR <- R3 MBR <- MEM[MAR] R1 <- MBR MAR <- R3 + 4	SelA = 01011, T9, C0 Ta, R, BW=11, M1, C1 T1, LC, SelC = 10101 SelA = 01011, MB=10, SelCop=01010, MC=1, T6, C0	Para el “lc” hemos obtenido la dirección de memoria guardada en r3 para llevarla a la memoria principal y sacar la parte real del número complejo. Después pasamos la dirección de memoria por la ALU

	MBR <- MEM[MAR] R2 <- MBR	Ta, R, BW=11, M1, C1 T1, LC, SelC = 10000, A0=1, B=1,C=0	para sumarle 4 y así guardar en r2 la parte imaginaria del número complejo.
addc r1, r2, r3, r4	R1 <- R1 + R3 SR <- R1 R2 <- R2 + R4	SelA=10101, SelB=01011, SelCop=01010, MC, T6, LC, SelC=10101, SelP=11, M7, C7 SelA=10000, SelB=00110, SelCop=01010, MC, T6, LC, SelC=10000, A0=1, B=1,C=0	Para la instrucción “addc”, sacamos del banco de registros los valores dentro de r1 y r3 para posteriormente sumarlos en la ALU, guardar el resultado de vuelta en r1 y actualizar el SR con el resultado de la parte real. Para la parte imaginaria hemos realizado lo mismo con los registros r2 y r4 pero sin actualizar el SR.

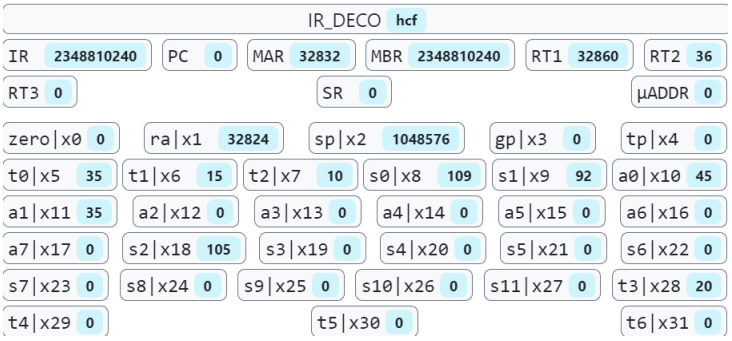
mulc r1, r2, r3, r4	RT1 <- R1 * R3 RT2 <- R2 * R4 RT3 <- RT1 - RT2 SR <- RT3 RT1 <- R1 * R4 RT2 <- R2 * R3 R2 <- RT1 + RT2 R1 <- RT3	SelA=10101, SelB=01011, SelCop=01100, MC, T6, C4 SelA=10000, SelB=00110, SelCop=01100, MC, T6, C5 MA, MB, SelCop=01011, MC, C6, SelP=11, M7, C7 SelA=10101, SelB=00110, SelCop=01100, MC, T6, C4 SelA=10000, SelB=01011, SelCop=01100, MC, T6, C5 MA, MB, SelCop=01010, MC, T6, LC, SelC=10000 T7, LC, SelC=10101, A0=1, B=1,C=0	La instrucción “mulc” realiza la multiplicación de dos números complejos en la que hay que realizar dos multiplicaciones y una resta en el caso de la parte real y dos multiplicaciones y una suma en el caso de la parte imaginaria. Para guardar los valores de las multiplicaciones utilizamos los registros temporales RT1 y RT2. Utilizamos RT3 para guardar el resultado real ya que necesitamos el R1 original para el cálculo de la parte imaginaria. Por último movemos el contenido de este registro RT3 al R1 ya que el R2 se actualiza directamente. A parte de esto tenemos que actualizar el registro de estado (SR) con la parte real.
	MBR <- SR R1 - R3 si SR.Z == 0 -> jump si no: siguiente R2 - R4	T8, C1 SelA=10101, SelB=01011, SelCop=1011, MC, SelP=11, M7, C7 A0=0, B=1, C=110, MADDR=jump SelA=10000, SelB=00110, SelCop=1011, MC, SelP=11, M7,	La instrucción “beqc” añadirá un cierto desplazamiento en el caso de que r1 y r3 sean iguales y ocurra lo mismo con r2 y r4. Lo primero que hacemos es una copia del contenido del SR ya que vamos a modificarlo. La forma de comprobar que r1 y r3 son iguales es

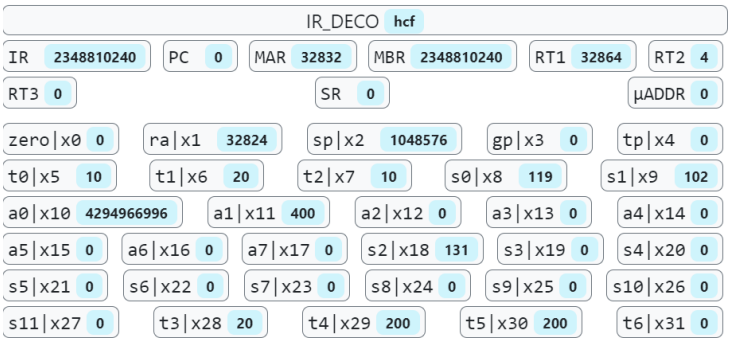
beqc r1, r2, r3, r4 offset	si SR.Z == 0 -> jump si no: siguiente RT1 <- PC RT2 <- IR(OFFSET) PC <- RT1 + RT2 jump: SR <- MBR	C7 A0=0, B=1, C=110, MADDR=jump T2, C4 SE=1, Size=110, Offset=0, T3, C5 MA, MB, SelCop=1010, MC, T6, C2 T1, C7, A0=1, B=1, C=0	hacer la resta y comprobar que sea 0 y lo mismo con r2 y r4. Como debe cumplir ambas condiciones, hacemos una detrás de la otra y si SR.Z==0 significa el flag Z no se activa, por lo que la resta es distinta de 0 y por tanto son distintos. En ese caso, saltaremos a la etiqueta jump que restaura el valor guardado en el MBR al registro de estados (SR). Si se cumplen ambas condiciones, lleva el PC a RT1 y el desplazamiento a RT2, hace su suma y actualiza directamente el PC.
call addr	RA<- PC PC <- IR	T2, LC, SelC=1, MR SE=0, Size=10100, Offset=0, T3, C2, A0=1, B=1,C=0	Esta instrucción equivale al jal ra funcion que hacíamos en ensamblador. Guarda PC en el registro RA y añade la dirección dada al PC, de manera que sea la siguiente ejecución que se vaya a ejecutar
ret	PC <- RA	SelA=1, MR, T9, C2, A0=1, B=1,C=0	Esta instrucción equivale al jr ra. Restaura el RA guardado al PC de forma que la siguiente instrucción que se ejecute vuelva a la línea guardada.
hcf	PC <- 0 SR <- 0	SelA=0, MR, T9, C2, C7, A0=1, B=1,C=0	Resetea a 0 tanto el PC como el SR (se actualiza).

EJERCICIO 2

TABLA DE RESULTADOS

	Ciclos de reloj sin extensión (s2)	Ciclos de reloj con la extensión (s1)	Mejora (%)
A == B (mul)	131	102	22,14%
A != B (suma)	105	92	12,38%





A != B
s1 (with extension) = 85
s2 (without extension) = 105

105 - 100%
92 - X
X = 87,62 -> Mejora = 100 - 87,62 = 12,38%

A == B
s1 (with extension) = 95
s2 (without extension) = 135

131 - 100%
102 - X
X = 77,86 -> Mejora = 100 - 77,86 = 22,14%

En el caso de que se vaya a trabajar con números complejos, merece en gran medida trabajar con la extensión del ejercicio 1. Esto se debe a que mejora en ciertos casos la eficiencia del programa hasta en un 30%. En un programa pequeño esta mejoría puede llegar a ser inapreciable, pero en uno extenso puede significar que la duración del programa llegue a ser ⅓ más rápido que si no contase con las instrucciones añadidas.

DIFERENCIAS, VENTAJAS E INCONVENIENTES

Entre las diferencias encontradas en la extensión, tenemos un notorio descenso en el acceso a posiciones de memoria. Esto se debe a que por ejemplo en el caso de la instrucción *“addc”*, esta ya se encarga de sumar las partes reales y las partes imaginarias por separado. Esto significa que solo se va a acceder a memoria una vez para buscar la instrucción. Sin embargo, si no se utiliza la extensión, tendríamos que sumar por separado las partes reales e imaginarias, teniendo que acceder a memoria en más ocasiones haciendo el programa notoriamente más lento. Esto no solo pasa con la instrucción *“addc”*, si no que también con la *“mulc”* o la *“beqc”*. Por ello si se utiliza la extensión de los complejos y se hacen uso de todas sus instrucciones, el programa aumentaría en gran medida su eficacia.

Entre las ventajas encontradas, podríamos destacar las mencionadas en el apartado de arriba. Al acceder menos veces a memoria debido a la eficacia de las instrucciones, conseguimos que se mejore la eficiencia del programa aumentando su rendimiento en más de un 20%. Cabe destacar la mejora en eficiencia que hemos conseguido al implementar nuestro método “lc” ya que hemos obtenido que, sin este, el porcentaje de mejora era tan solo de un 3% pero al incluirlo sobrepasa el 20% ya que solo se accede una vez a memoria y se mueven ambos valores (el real y el imaginario) al registro asignado.

Con respecto a los inconvenientes lo único que se puede destacar es que resulta más sencillo trabajar con las instrucciones de ensamblador (función *no_ext*) que modificando directamente las señales (función *with_ext*) aunque como se ha demostrado esta segunda opción sea más eficiente.

Es por tanto que concluimos que el uso de la extensión es muy positivo ya que aunque requiera de más conocimiento a nivel de microprocesador luego los resultados obtenidos resultan favorables. Además, con las instrucciones de complejos no es necesario preocuparse de tratar por separado la parte real y compleja de los números si no que siempre se trabaja con ellas de forma conjunta.

CONCLUSIÓN

Como conclusión del trabajo realizado, sacamos diversas cosas positivas. Entre ellas el hecho de tener un mayor conocimiento acerca de cómo funciona un procesador por dentro lo cual será de utilidad no solo para esta práctica si no para la asignatura en general.

También queremos destacar el gran uso y la gran ayuda que nos ha aportado el simulador proporcionado, ya que era bastante intuitivo y muy útil a la hora de analizar el programa que estábamos ejecutando. Esto se debe a que nos permite ver de forma muy intuitiva que está pasando en todo momento durante la ejecución para poder analizar nuestro código y entender realmente cómo funciona, y en el caso de haber un error corregirlo rápidamente.

Con respecto a los problemas encontrados, no hemos sido capaces de encontrar algunos como tal. Si que hemos notado un poco de lentitud al comenzar el trabajo, debido a que teníamos que aprender a usar el simulador *WepSim*, pero una vez lo entendimos no hubo más inconvenientes.

Para finalizar queremos destacar que hemos invertido un total de 20 horas aproximadamente entre los dos integrantes del grupo, lo cual consideramos un tiempo apropiado para las dimensiones de la práctica ya que creemos que no hemos perdido mucho el tiempo y que lo hemos invertido bastante bien.