PRÁCTICA 1

Índice

Objetivos:

El objetivo de esta práctica es familiarizarse con el simulador Ripes utilizando los procesadores RSCP R5SP, así como entender las bases del lenguaje assembly relacionado con este simulador y su relación con el lenguaje máquina. Para lograrlo se realizan varios programas sencillos en este simulador.

1.

```
.data
etiqueta: .word 1
.text
lw a1, 0(a1)
sw a1, 3(a1)
bne t1, t2, etiqueta
addi t2, t3, 11
sub zero,a2,a3 # Esta instrucción es correcta pero no hace nada
lw a1, 3(a0)
```

2.

a)

el resultado final del programa al haber realizado todos los bucles se guarda en la posición de memoria Ox10000014 (en hexadecimal) lo que es equivalente a la posición 4 + el valor de a0 al final del programa

b)

primero el programa almacena los números 3, 5, 2, 8 a las posiciones de memoria a las que asigna como etiquetas xx, yy, oo y zz respectivamente. Luego asigna al registro a0 la dirección de memoria equivalente a la etiqueta xx. Luego hace que los registros a1 y a2 guarden sendos ceros (para cada registro usa un proceso distinto usando la instrucción sub o add. Luego, tras fijar la etiqueta loop a la que se volverá posteriormente define lo que se hará en cada una de las iteraciones del bucle. Primero se asigna al registro a7 el valor de a1 – 4, luego se comprueba si a7 == 0, en cuyo caso el programa salta a la etiqueta final para ejecutar luego la última instrucción. Si no se ha pasado a la etiqueta final, se almacena en a3 el contenido de la posición de memoria almacenada en a0, en a2 se almacena el valor que tiene a2 + el valor actual de a3, a0 se incrementa en 4 (haciendo que la próxima vez que se use este registro su valor sea la posición de memoria correspondiente a a la próxima palabra) y a a1 se le incrementa 1. Luego el flujo de ejecución salta a la etiqueta loop gracias a la instrucción j loop. Por último, cuando el programa ha salido del bucle y llega a la instrucción final lo que hace ese almacenar en la posición de memoria a0 + 4 el valor que almacena a2. En definitiva en cada iteración del bucle en a2 se guarda la suma acomulada de xx, yy, oo, zz. En cada iteración se va sumando en a2 (que al principio está iniciado en 0) el contenido de xx, yy, oo y zz respectivamente y esta suma es acomulativa, con lo cual, tras 4 iteraciones, a2 guarda el valor xx + yy + oo + zz. Por lo tanto tras ejecutar la última iteración a2 habrá tomado el valor de xx + yy + oo + zz, que es el resultado que se acaba almacenando en Ox10000014.

c)

La sentencia de flujo utilizada es un while que engloba todo el loop. Solo se sale de este bucle si a7 es 0 por lo que tras la primera suma del bucle hay un if (a7 == 0) y dentro del if un break para salir del bucle.

d)

a1 empieza el bucle siendo 0 y se le suma 1 al principio de cada iteración. Al principio de cada iteración se le asigna a a7 a1 – 4 lo que hace que en la quinta iteración a a7 se le asigne 0 (a1 = 4) y por tanto se salga del bucle. Es decir, en cierto modo a1 se encarga de llevar la cuenta del número de iteraciones que tiene que hacer el bucle.
La función de a0 es guardar la posición de memoria en la que se irán guardando los distintos números que se vayan obteniendo. Es por ello que en cada iteración se le suma cuatro para que vaya accediendo a las posiciones de memoria contiguas y almacenando allí los números. También se usa para determinar la posición de memoria en la que se almacenará el valor final de a2. Este valor se almacena, como ya se ha dicho, en la posición de memoria a0 + 4 siendo a0 el valor de a0 al salir del bucle

3.

e)

El “executable code” es básicamente la expresión de “source code” en binario. El source code es el código expresado en assembly, el “executable code” es código máquina. El “source code es transfomado en “executable code” mediante el ensamblador.

f)

El pc empieza en la posición indicada (0) de memoria y al terminar cada instrucción se le suma 4 para que pase a la siguiente instrucción a menos que se haya procesado una instrucción de salto, en cuyo caso se va a la dirección de memoria que indica la instrucción correspondiente. Esto es porque cada instrucción está almacenada en la posición de memoria contigua a la instrucción anterior y cada instrucción es una palabra (word). Es por ello que hay que sumar 4 para ir a la siguiente instrucción natural (porque hay que sumar 4 bits que hay contenidos en una palabra). Esto se puede ver en la ventana instruction memory o también en la sección .text de la memoria, que podemos comprobar, efectivamente que empieza en el 0.

g)

En las 5 primeras instrucciones después de .data lo que se hace es almacenar números en las etiquetas asociadas a una posición de memoria xx, yy, oo y zz. Al empezar la sección .text primero se carga en el registro a0 el contenido de la posición de memoria referenciada por la etiqueta xx, luego se guarda en el registro a1 a1 – a1, es decir, el valor 0. Luego, en a2 se guarda el valor a0 – a0, es decir, 0. Luego se define la etiqueta dentro de la memoria de instrucciones loop. Luego En el registro a7 se guarda el valor a1 – 4, luego se comprueba si el registro a7 es 0, en cuyo caso se va a la etiqueta de instrucciones final. Después se guarda en el registro a3 el contenido de la posición de memoria a0. Posteriormente se guarda en el registro a3 el resultado de a2 + a3. Luego al registro 0 se le suma 4 y, por último, al registro a1 se le suma 1 para luego saltar a la etiqueta de instrucciones loop. En el momento que se salta a la etiqueta de instrucciones final, se ejecuta la última instrucción: el valor del registro a2 es almacenado en la posición de memoria a0 + 4. En efecto al comprobar cada instrucción una a una se comprueba que todas tienen el comportamiento esperado

4.

h)

Con el RSCP se necesitan 20 ciclos de reloj para finalizar el programa mientras que con el R5SP se necesitan 33 ciclos de reloj

i)

diagramas de ejecución con RSCP(los 2 primeros) y con R5SP (los 2 últimos)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Invalid instruction	•													
lw x10 16(x10)		•												
auipc x5 0x10000			•											
lb x5 -8(x5)				•										
auipc x0 0x10000					•									
sw x10 0(x0)						•								
auipc x11 0x10000							•							
lw x11 0(x11)								•						
beq x11 x10 12 <salta>									•			•		
sub x11 x11 x10										•			•	
jal x0 0x20 <loop>											•			•
auipc x13 0x10000														

	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
auipc x5 0x10000															
lb x5 -8(x5)															
auipc x0 0x10000															
sw x10 0(x0)															
auipc x11 0x10000															
lw x11 0(x11)	•														
beq x11 x10 12 <salta>		•			•			•			•				
sub x11 x11 x10			•			•			•						
jal x0 0x20 <loop>				•			•			•					
auipc x13 0x10000													•		
lw x13 -24(x13)														•	
jal x0 0x34 <end>															•

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Invalid instruction	IF	ID	EX	MEM	WB																										
lw x10 16(x10)		IF	ID	EX	MEM	WB																									
auipc x5 0x10000			IF	ID	EX	MEM	WB																								
lb x5 -8(x5)				IF	ID	EX	MEM	WB																							
auipc x0 0x10000					IF	ID	EX	MEM	WB																						
sw x10 0(x0)						IF	ID	EX	MEM	WB																					
auipc x11 0x10000							IF	ID	EX	MEM	WB																				
lw x11 0(x11)								IF	ID	EX	MEM	WB																			
beq x11 x10 12 <salta>									IF	ID	-	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB		
sub x11 x11 x10										IF	-	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID				
jal x0 0x20 <loop>												IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF				
auipc x13 0x10000													IF	ID				IF	ID				IF	ID				IF	ID	EX	MEM
lw x13 -24(x13)														IF					IF					IF					IF	ID	EX
jal x0 0x34 <end>																														IF	ID

	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
Invalid instruction																														
lw x10 16(x10)																														
auipc x5 0x10000																														
lb x5 -8(x5)																														
auipc x0 0x10000																														
sw x10 0(x0)																														
auipc x11 0x10000																														
lw x11 0(x11)																														
beq x11 x10 12 <salta>	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB													
sub x11 x11 x10	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID															
jal x0 0x20 <loop>	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF															
auipc x13 0x10000	IF	ID				IF	ID				IF	ID				IF	ID	EX	MEM	WB										
lw x13 -24(x13)		IF					IF					IF					IF	ID	EX	MEM	WB									
jal x0 0x34 <end>																		IF	ID	EX	IF/MEM	ID/WB	EX	IF/MEM	ID/WB	EX	IF/MEM	ID/WB	EX	IF/ME

j)

Con el procesador R5SP multiciclo se llegan a realizar 5 fases de 5 instrucciones a la vez mientras que con el RSCP solo se realiza una única fase de una instrucción a la vez.

Conclusiones:

Finalmente creo que he asimilado bien el funcionamiento del simulador en el que trabajamos así como las instrucciones necesarias para controlarlo en lenguaje assembly y cómo se ejecutan estas instrucciones dependiendo del procesador que estemos usando.