ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 2 Introducción a la microprogramación

Curso 2019/2020

Jorge Rodríguez Fraile, 100405951, Grupo 81, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares,100405834, Grupo 81, 100405834@alumnos.uc3m.es

Índice

Ejercicio 1

Instrucción	Operaciones elementales	Señales de control	Decisiones de diseño
ld Rdest, Rorig	1.-Rdest ← Rorig	1.-(SELA=10000, SELC=10101, T9, LC, A0=1, B=1, C=0)	Sacamos las direcciones de ambos registros de la instrucción teniendo en cuenta la posición que ocupan en la misma, y desplazamos el valor de uno al otro.
ldi Rdest, U16	1.-Rdest ←IR[U16]	1.-(SIZE=10000,OFFSET= 00000, T3, LC, SELC=10101, A0=1, B=1, C=0)	Sacamos de la instrucción el valor a almacenar y a la vez el registro destino relativo a la posición que ocupan en la instrucción. Después activamos la escritura en el banco de registro.
ld Rdest, (Rorig)	1.-MAR← IR[Rorig] 2.-MBR← MAR 3.-Rdest ←MBR	1.-(SELA=10000,T9,C0) 2.-(TA,R,BW =11, M1, C1) 3.-(SELC=10101,LC,T1,A0=1,B=1,C=0)	Pasamos la dirección almacenada en el registro Rorig a MAR, para acceder al valor que debemos guardar en Rdest. Sacamos de la memoria por MBR el valor a almacenar en Rdest
add_a Rorig	1.-RT1← A 2.-RT3← Rorig + RT1 3.-A← RT3	1.-(SELA= 00100, MR, T9,C4) 2.-(SELB = 10101, MA, MC,SELCOP=1010,C6) 3.-(T7,LC,SELC=00100,MR,SELP=11,M7,C7,A0=1,B=1,C=0)	Obtenemos el valor de A a partir de SelA y lo guardamos en un registro temporal para luego sumarlo. Después, obtenemos Rorig a partir de la instrucción y lo sumamos en la ALU, el resultado se guardará en RT3 temporalmente, mientras paralelamente se actualiza el estado de la operación. Por último, cargamos en el registro A el valor

			almacenado en RT3.
addi_a S16	1.-RT2← IR[S16] 2.-RT1← A 3.-A← RT2+RT1	1.-(SIZE = 10000, SE, OFFSET = 00000, T3, C5) 2.-(SELA= 00100, MR, T9,C4) 3.-(MA,MB= 01, MC,SELCOP=1010,T6, LC,SELC=00100,MR,S ELP=11,M7,C7,A0=1,B =1,C=0)	Obtenemos de la instrucción el valor a sumar a A y lo guardamos en RT2, almacenamos el valor de A en RT1, los sumamos y almacenamos de vuelta al registro A.
inc Rdest	1.-Rdest← Rdest+1	1.-(SELA=10101,MB=1 1,SELCOP=1010,MC,T 6,SELC=10101,LC,SEL P=11,M7,C7,A0=1,B=1, C=0)	Sacamos el valor del registro Rdest, le sumamos 1 en el ALU con los valores predefinidos y pasamos el resultado de vuelta al Rdest.
dec Rdest	1.-Rdest← Rdest-1	1.-(SELA=10101,MB=1 1,SELCOP=1011,MC,T 6,SELC=10101,LC,SEL P=11,M7,C7,A0=1,B=1, C=0)	Sacamos el valor del registro Rdest, le restamos 1 en el ALU con los valores predefinidos y pasamos el resultado de vuelta al Rdest.
jp S16	1.-RT2← S16 2.-RT1← PC 3.-PC← RT1+RT2	1.-(SIZE = 10000, SE,OFFSET = 00000, T3,C5) 2.-(T2,C4) 3.-(MA,MB= 01,MC,SELCOP=1010, T6,C2,A0=1,B=1,C=0)	Almacenamos el valor inmediato en RT2 y el PC en RT1, después elegimos la operación suma y el resultado lo devolvemos a PC
jpz S16	1.-if(SR.Z==1) PC← PC+S16 2.-else fetch	1.-(A0 = 0, B=0, C=0110, MADDR = jp), 2.-(A0=1,B=1,C=0)	Si el valor de Z es 1 salta a la instrucción jp, si no hace fetch para ir a la siguiente.
call U16	1.-MAR←SP-4, SP← SP-4 2.-MBR← PC 3.-MP[MAR] ← MBR	1.- (SELA = 11101, MR,MB=10,SELCOP=1 011,MC,T6,SELC=111 01,LC,C0), 2.-(T2,C1),	Obtenemos SP desde RA, y lo restamos a 4 en la ALU, el resultado se actualizará en SP y se cargará en MAR. Después, cargamos el

	4.-PC← U16	3.-(TA,BW=11,TD,W), 4.-(SIZE=10000,OFFSET=00000,T3,C2,A0=1,B=1,C=0)	valor de PC en MBR, en el siguiente ciclo lo escribimos en el valor de MAR. Por último,cargamos en PC U16.
ret	1.-MAR← SP 2.- MBR← MP[MBR] SP← SP+4 3.- PC← MBR	1.-(SELA = 11101 ,MR, T9,C0) 2.-(SELA= 11101, MR, MB=10,SELCOP=1010, MC,T6,SELC=11101,LC,TA,R,BW=11,M1,C1), 3.-(T1,C2, A0=1, B=1, C=0)	Cargamos la dirección almacenada en SP que queremos leer en MAR, sumamos 4 al SP y a la vez hacemos la operación de lectura, finalmente pasamos el valor de MBR a PC.
halt	1.-PC← R0	1.-(SELA=00000,T9,C2,A0=1,B=1,C=0)	Cargamos el valor del registro R0 (vale 0) a PC.
push Rorig	1.-SP← SP-4 MAR← SP-4 2.-MBR← Rorig 3.-MP[MAR] ← MBR	1.-(SELA=11101,MR,MB=10,SELCOP=1011,MC,T6,C0,SELC=11101,LC) 2.-(SELA=10101,T9,C1) 3.-(TA,BW=11,TD,W,A0=1,B=1,C=0)	Restamos 4 al SP y pasamos el nuevo valor a MAR y actualizamos SP, pasamos el contenido del registro a guardar a MBR. Finalmente guardamos en memoria el valor de MBR en la dirección MAR
pop Rdest	1.-MAR← SP 2.-MBR← MP[MAR] SP← SP+4 3.-Rdest← MBR	1.-(SELA=11101,MR,T9,C0) 2.- (TA, BW=11, R, M1, C1, SELA=11101, MR, MB=10,SELCOP=1010, MC,T6,SELC=11101, LC) 3.- (T1, SELC=10101,LC,A0=1,B=1,C=0)	Cargamos la dirección del valor a sacar de pila en MAR. Leemos de la memoria la dirección de MAR y la pasamos a MBR, a la vez que sumamos 4 a SP para hacer hueco en pila. Guardamos el valor de MBR (el sacado de pila) en Rdest.

PRUEBAS

NOTA: En estas pruebas, los registros que no se nombran no se modifican (excepto, obviamente, PC y RI)

Instrucción	Entradas	Resultados esperados	Resultados obtenidos
ld Rdest, Rorig	ld A BC (BC = 6)	A tiene el mismo valor que BC, 6	A = 6
ldi Rdest, U16	ldi BC,4	BC tiene el valor 4	BC = 4

ld Rdest, (Rorig)	ldi A,15
	push A
	ld BC, (SP)
		A tiene el valor de la dirección que se guarda en BC
		A = 15, B=15, SP=15
			A=15, B=15, SP=15

add_a Rorig	ldi A,15
	ldi BC, 1
	add_a BC
		A se suma a BC.
		A = A + BC =15+1=16
			A = 16

addi_a S16	ldi A,8 addi_a -2	A = A - 2 = 8 - 2 = 6	A = 6
inc Rdest	ldi BC,2 inc BC	BC = BC + 1 = 3	BC = 3
dec Rdest	ldi BC,10 dec BC	BC = BC-1 = 9	BC = 9
jp S16	.text main: (PC= 0X800)ber: ldi BC, 12 ldi A, 16 jp ber	PC = 0X800 (vuelve a la etiqueta de ber)	PC = 0X800C

jpz S16	1 CASO:(Z=1)
	ber:ldi A,0
	ldi BC,0
	add_a BC
	jpz ber
	PC(después de add_a) = 0X800C
	
	2 CASO: (Z=0)
	ber:ldi A,0
	ldi BC,24
	add_a BC
	jpz ber
	PC(después de add_a) = 0X800C
		1 CASO:
		PC = 0X800C
		y salta a la instrucción dada
		PC=0x8000
		
		2 CASO:
		PC = 0x800C y como no se cumple la condición, continua y PC=0x8010
			1 CASO:
			PC pasa de 0X800C a 0x8000
			
			2 CASO:
			PC pasa de 0x800C a 0x8010 que es la siguiente instrucción.

call U16	call 45 PC(ant.) = 0x8000	PC actual se guarda en pila y PC = 45, El valor del PC(ant.)+4 se guarda en SP	PC(actual) = 45
ret	ldi BC, 4096 push BC ret	PC debe valer el valor que hay encima de la pila: PC=0x00001000 (4096)	PC=0x00001000 (4096)
halt	halt	Poner PC a PC=0x00000000	Pasa de PC=0x00008004 a PC=0x00000000

push Rorig	ldi BC, 12
	push BC
	ld A, (SP)
		En SP (stack pointer) esté el 12.
		Para probar ver que hay en SP
			A= 12

pop Rdest	ldi BC, 256 push BC pop A	A=256 La cima de la pila	A=256

Ejercicio 2

CÓDIGO EN MIPS32:

```
.data
        vector: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

.text
        sumav: # apilar $a0 y $a1
                addi $sp $sp -8
                sw $a0 4($sp)
                sw $a1 0($sp)
                # $v0 = suma de los elementos de vector
                li $v0 0
        b1:     beq $a0 $0 f1
                lw $t0 ($a1)
                add $v0 $v0 $t0
                addi $a1 $a1 4
                addi $a0 $a0 -1
                b b1
                # desapilar $a1 y $a0
        f1:     lw $a1 0($sp)
                lw $a0 4($sp)
                addi $sp $sp 8
                # return
                jr $ra

main: # llamar a la subrutina contar
                li $a0 10
                la $a1 vector
                jal sumav
                # terminar la ejecución
                li $v0 10
                syscall
```

CÓDIGO EN Z80:

```
.data
        vector: .word 1,2,3,4,5,6,7,8,9,10

.text
        sumav: ##Apilar IX e IY
                push IX
                push IY
                 ##A = Suma del vector
                ldi A,0
                ##Bucle
                        add_a IX
                        ##COMPROBACION DE QUE EL VECTOR ESTE VACIO
                        jpz f1
                        ldi A,0
                        b1: ld BC,(IY)
                        ##INCREMENTO DE DIRECCION DE MEMORIA
                        inc IY
                        inc IY
                        inc IY
                        inc IY
                        ##SUMA DEL ACUMULADOR
                        add_a BC
                        ##Reducción de indice
                        dec IX
                        jpz f1
                        jp b1
                        ##Desapilar IX e IY
                 f1:
                 ##copia del valor al $v0 equivalente
                 ld  HL,A
                 ##desapilar
                 pop IY
                 pop IX
                        ##return
                        ret

        main:
                ldi IY,vector
          ldi IX,10
          call sumav
          fin: halt
```

COMPARACIÓN DE AMBOS LENGUAJES DE ENSAMBLADOR (MIPS32 Y Z80)

Antes de la llamada a la función, no existe diferencia entre uno u otro, solo el nombre de la instrucción. Dentro de la función nuestro procesador tiene la ventaja de que hace directamente el hueco en la pila y mete el valor del registro con la instrucción “push”, y en el MIPS tendríamos que restar al puntero de pila para crear el hueco y después pasar el valor del registro a la dirección de memoria que almacena el puntero de pila.

En cuanto al salto condicional “beq” en Z80 está el inconveniente de que la condición depende de haber hecho una operación previa, es decir, que en el resultado de la ALU, nuestro Flag.zero valga 1, esto significa que tendremos que utilizar comandos que actualicen el registro de estados y además obtenga un Flag.zero = 1 para poder hacer el salto que deseemos. En este caso, hemos usado un add_a con A=0, y al final del bucle un dec IX, hasta que IX valiera 0 y nos pudiésemos salir del bucle.

Uno de los principales problemas del Z80 es también el incremento de valores, al usar un acumulador (A) en sumas, no podemos hacer un add con valor inmediato de 4 a la dirección de memoria en la que nos encontramos, si no que tendremos que usar el comando inc 4 veces, o en su defecto, mover el valor de A a otro registro y cargar la dirección de memoria en la misma para poder sumar 4 directamente y después restaurar los valores de cada registro.

De nuevo, resulta que el comando pop es una ventaja en Z80, porque nos salva de tener que hacer por nosotros mismos espacio en pila para sacar y guardar valores, ya que se hace directamente,siguen siendo los mismos ciclos de reloj, pero a un nivel de usuario es mucho más cómodo.

Conclusiones

Programar en un lenguaje de ensamblador diferente al que estábamos acostumbrados (MIPS32), hace que tengamos más en cuenta las grandes diferencias que pueden tener dos procesadores.

También nos ha ayudado a ver nuevas formas de representar los datos en un procesador, por ejemplo, el uso de un acumulador en el Z80 le otorga una dinámica diferente a la que un simple add R1, R2, R3 puede hacer. Aunque la segunda opción sea bastante más directa y útil, la primera nos otorga dificultades y problemas que son bastantes más satisfactorios de resolver.

En definitiva, pensamos que MIPS32 es un procesador mejor estructurado y con un abanico de instrucciones más acorde con las necesidades de un ordenador actual, aunque el Z80 nos otorga una nueva forma de ver el procesador y cómo funciona, por lo que, consideramos está práctica más centrada en el conocimiento que en la propia utilidad de nuestro juego de instrucciones.

Problemas encontrados

Los principales problemas que hemos encontrado en el ejercicio 1 han sido:

Al realizar varias salidas de datos de la microinstrucción y no poderlos sacar simultáneamente por la limitación de una misma MR para todos, la solución fue sacar y almacenar el primero en un ciclo y en el siguiente ciclo sacar el otro.

Cambiar el contador de programa para saltar a otra instrucción como en el jpz, lo hemos resuelto haciendo referencia al jp con una etiqueta si se cumplia la condicion y si no pasaba a la siguiente microinstrucción.

En el ejercicio 2 los problemas que nos hemos encontrado han sido principalmente al adaptar el código darnos cuenta cómo se hacen las operaciones habituales en el Z80, han sido los siguientes:

Llamar a la función y que no pierda el PC de la instrucción que llama, para ello utilizamos el call que antes de cambiar el PC al de la instrucción lo guarda en la cima de la pila.

Otro problema que encontramos fue utilizar direcciones relativas en la función jp, ya que lo utilizabamos como un valor inmediato y al utilizar etiquetas, no saltaba correctamente.

Almacenar y sacar de pila, utilizamos push y pop respectivamente

Sumar 4 a la dirección del vector para pasar al siguiente valor, ya que en Z80 no se puede sumar a un registro que no sea A un inmediato, hemos optado por usar inc 4 veces que suma 1 cada vez.

La condición de salida del bucle, para resolver este problema ya que antes de la condición debe haber una operación en la ALU que de 0 lo que hacemos es restar 1 con dec justo antes al índice y cuando este llegue a 0, el bucle ha terminado y sale.

Regresar al programa que llama a la función, para esto utilizamos el ret que gracias a que la instrucción call guarda el PC en la pila podemos recuperarlo y regresar al programa.

Terminar el programa, utilizamos el halt que cambia el PC a 0 y no se ejecutan más instrucciones.

Horas empleadas

El desarrollo de esta práctica nos ha llevado alrededor de 7 horas, a lo largo de dos semanas, en las que hemos entendido el funcionamiento de la unidad de control (1,5 horas), para posteriormente poder escribir las instrucciones pedidas (3 horas) y adaptar un código (2,5) en un lenguaje ya conocido MIPS32 al que nosotros mismos hemos escrito.

Al mismo tiempo que íbamos desarrollando los ejercicios íbamos escribiendo lo que correspondía a cada apartado con sus pruebas incluidas, en el tiempo de adaptar el código va incluido la realización de la grabación del WepSIM.