PRÁCTICA 2

ESTRUCTURA DE
COMPUTADORES

Índice:

Ejercicio 1

A continuación tenemos la tabla con los datos solicitados en el ejercicio 1. Para mejor presentación, hemos decidido poner las decisiones de diseño a continuación de la tabla, y vincular a cada apartado en su celda correspondiente:

Instrucción	Lenguaje RT	Señales de Control	Decisiones de diseño
ld Rdest, Rorig	BR[Rdest] <- BR[Rorig]	(SELA=10000, T9, LC, SELC=10101, B, A0)	[1]
ldi Rdest , U16	BR[Rdest] <- U16	(SIZE=10000,OFFSET=0,T3,LC,SELC=10101, B, A0)	[2]
ld Rdest , (Rorig)	MAR <- BR[Rorig]	(SELA=10000,T9, C0)	[3]
	MBR <- Memory[MAR]	(TA, R, BW=11, M1, C1)	
	BR[Rdest] <- MBR	(T1, SELC=10101,LC, B, A0)	
add_a Rorig	RT1 <- BR[Rorig]	(SELA=10101,T9,C4)	[4]
	BR[A] <- BR[A] + RT1	(SELB=100, MR, MA, SELCOP=1010, MC, T6, LC, SELC=100,SELP=11,M7,C7, B, A0)	
addi_a S16	RT2 <- S16	(SE, SIZE=10000, T3, C5)	[5]
	BR[A] <- (BR[A] + RT2)	(SELA=100, MR, MB=01, SELCOP=1010, MC, SELP=11, M7, C7, T6, LC, SELC= 100, A0, B)	
inc Rdest	BR[Rorig] <- (BR[Rorig] + 1)	(SELA=10101, MB=11, SELCOP=1010, MC, T6, LC, SELC=10101, SELP=11, M7, C7, B, A0)	[6]
dec Rdest	BR[Rorig] <- (BR[Rorig] - 1)	(SELA=10101, MB=11, SELCOP=1011, MC, T6, LC, SELC=10101, SELP=11, M7, C7, B, A0)	[7]
jp S16	RT2 <- S16	(SE, SIZE=10000, T3, C5)	[8]
	RT1 <- PC	(T2, C4)	

	PC <- (RT1 + RT2)	(MA, MB=01, SELCOP=1010, MC, T6, C2, A0, B)	

jpz S16	if Flag.Zero == 1
	  μAddr <- (μAddr + 1)
	else
	  fetch()
	fi
		(C=110, B, MADDR=backtofetch)
		
		[...]
		
		backtofetch: (A0, B)
			[9]

	RT2 <- S16	(SE, SIZE=10000, T3, C5)	
	RT1 <- PC	(T2, C4)	
	PC <- (RT1 + RT2)	(MA, MB=01, SELCOP=1010, MC, T6, C2, A0, B)	
call U16	MAR<- (SP+4) SP<- (SP+4)	(SELA=11101, MR, C0, MB=10, SELCOP=1011, MC, T6, LC, SELC=11101)	[10]
	MBR <- PC	(T2, C1)	
	Memory[MAR] <- MBR PC <- U16	(TA, BW=11, TD, W, SIZE=10000, OFFSET=0, T3, C2, MC=0, B, A0)	
ret	MAR <- BR[SP] RT3 <- (BR[SP] + 4)	(SELA=11101, MR, T9, C0, MB=10, SELCOP=1010, MC, C6)	[11]
	MBR <- Memory[MAR] BR[SP] <- RT3	(TA, R, BW=11, M1, C1, T7, LC, SELC=11101, MR)	
	PC <- SP	(T1, C2, A0, B)	
halt	PC <- EXCODE(0)	(EXCODE=0, T11, C2, B, A0)	[12]
push	BR[SP] <- (BR[SP] - 4) MAR <- (BR[SP] - 4)	(SELA=11101, MR, MB=10, SELCOP=1011, MC, T6, LC, SELC=11101, C0)	[13]
	MBR <- BR[Rorig]	(SELA=10101, T9, C1)	
	Memory[MAR] <- MBR	(TA, BW=11, TD, W, A0, B)	
pop Rdest	MAR <- PC RT3 <- (PC+4)	(SELA=11101, MR, T9, C0, MB=10, SELCOP=1010, MC, C6)	[14]

	MBR <- Memory[MAR] PC <- RT3	(TA, R, BW=11, M1, C1, T7, LC, SELC=11101, MR)	
	RDEST <- MBR	(T1, LC, SELC=10101, MC=0, B, A0)	

Decisiones de Diseño:

[1] En esta instrucción se ha minimizado el número de ciclos a uno ya que en el mismo ciclo que se lee el registro se escribe en el registro destino.

[2] En esta instrucción se ha minimizado el número de ciclos a uno ya que en el mismo ciclo que se selecciona el valor inmediato con el selector se escribe en el registro destino.

[3] Esta instrucción no hemos podido reducirla más de tres ciclos de reloj, ya que en un mismo ciclo no se puede atravesar un registro ni usar el bus dos veces, por lo que en el primero usamos el bus para pasar la dirección a MAR, en el segundo para pasar el valor de memoria a MBR y como no podemos atravesar el registro necesitamos el tercer ciclo para pasar el valor al registro destino.

[4] Esta función nos hemos visto obligados a realizarla en 2 ciclos ya que al tener que usar diferentes valores para los multiplexores MR no podíamos pasar los valores de los registros a la vez a la ALU, por lo que en el primer ciclo hemos pasado el valor del registro indicado por el usuario a RT1, y en el siguiente ciclo de reloj hemos sumado ese registro temporal con en valor de A, y lo hemos cargado directamentente en el registro A ya que el bus de datos no estaba en uso.

[5] Usamos un registro auxiliar RT2 para hacer la suma, puesto que no podemos cargar el bus con dos valores distintos, esto produciría un corto.

[6] Esta función la hemos realizado en un único ciclo, ya que podemos sumar un valor del banco de registros con un inmediato (1) del multiplexor MB en un ciclo y a la vez llevarlo al banco de registro de nuevo, ya que no usamos el bus para transmitir dos datos a la vez ni atravesamos un registro.

[7] Esta función la hemos realizado en un único ciclo ya que podemos restar un valor del banco de registros con un inmediato (1) del multiplexor MB en un ciclo y a la vez llevarlo al banco de registro de nuevo, ya que no usamos el bus para transmitir dos datos a la vez ni atravesamos un registro.

[8] Guardamos tanto S16 como PC en dos registros auxiliares para luego operar con ellos y almacenarlos en PC.

[9] Para comprobar que el Flag.Zero es igual a 1, podemos usar la unidad de control. Como la ALU en MIPS no tiene instrucciones de comparación lógica, esta es la única opción que tenemos.
Cuando no es 1, salta a MADDR, que hemos definido como las instrucciones que llaman al fetch.

[10] En esta función hemos necesitado tres ciclos de reloj, ya que en el primero realizamos la suma del registro SP más el inmediato 4 del multiplexor MB, y lo guardamos en MAR y en SP, después pasamos el valor de PC a MBR, tarea que no hemos podido realizar en el primer ciclo ya que teníamos ocupado el bus de datos, y por último en el tercer ciclo pasamos el valor que teníamos en MBR a la memoria principal y aprovechamos para pasar el valor inmediato U16 seleccionado con el selector al registro PC.

[11] En esta instrucción hemos aprovechado que ciertas acciones no requieren uso del bus interno para realizar a la vez otras que sí lo utilizan, para optimizar al máximo el número de ciclos de reloj que usamos.

[12] En esta función hemos decidido generar el valor de cero con EXCODE y cargarlo directamente en PC.

[13] Igual que en instrucciones anteriores, optimizamos realizando varias acciones en el mismo ciclo de reloj, siempre y cuando no creen conflicto o cortos. En esta instrucción, cargamos un valor en la memoria pasándolo a través del MBR y además, decrecemos el puntero de pila.

[14] La función de pop la conseguimos realizar en tres ciclos de reloj, para ello en el primero aprovechamos tanto el bus de datos pasando la dirección de PC a MAR, y a la par hacemos uso de la ALU para sumarle cuatro al valor anterior de PC y lo guardamos en RT3, ya que no podemos usar el bus puesto que está ocupado, en el siguiente ciclo traemos de la memoria principal el valor a MBR, y como no podemos atravesar el registro en un ciclo, mientras aprovechamos el bus de datos que está vacío para pasar el valor del registro temporal (PC+4) al registro PC, y por último copiamos el valor de MBR en el registro de destino indicado por el usuario.

Ejercicio 2

Lo siguiente es la comparación de los juegos de instrucciones empleados en el ejercicio 2:

Z80	Mips32	Diferencias
push *reg*	addi $sp $sp -4 sw *reg* $sp	En Z80 tenemos una etiqueta que realiza el push directamente mientras que en MIPS lo tenemos que realizar manualmente con dos etiquetas.
ldi *reg, valor*	li *reg, valor*	Son iguales excepto por el nombre.
ldi A 0 add_a *reg1* jpz *etiqueta*	beq *reg1, reg2, etiqueta*	El Z80 no posee instrucciones de comparación a parte de jpz que salta si la anterior operación da cero, por lo que tenemos que sumar a cero el número a comparar y después ya podemos llamar a jpz.
ld *reg1, (reg2)*	lw *reg1, (reg2)*	Son iguales excepto por el nombre.

ld A, *reg1*
add_a *reg2*
ld *reg1*, A
	add *reg1, reg1, reg2*	El Z80 no posee instrucción de suma de dos valores, solo suma A con un valor por lo que hay que mover el valor a A, sumar y volver a moverlo.

ld A, *reg*
addi_a *valor*
ld *reg*, A
	addi *reg, reg, valor*	El Z80 no posee instrucción de suma de un valor con un inmediato, solo suma A con un valor inmediato por lo que hay que mover el valor a A, sumar y volver a moverlo .

jp *etiqueta*	b *etiqueta*	Son iguales excepto por el nombre.
pop reg	lw *reg, $sp* addi *$sp, $sp,* 4	En Z80 tenemos una etiqueta que realiza el pop directamente mientras que en MIPS 32 lo tenemos que realizar manualmente con dos etiquetas.
ret	jr $ra	La función en Z80 es más “sencilla”.
ldi *reg, etiqueta*	la *reg, etiqueta*	Son iguales excepto por el nombre.
call *etiqueta*	jal *etiqueta*	Son iguales excepto por el nombre.
halt	li $v0 10 syscall	En Z80 tenemos una etiqueta que realiza el halt directamente mientras que en MIPS 32 lo tenemos que realizar manualmente con dos etiquetas.

Conclusiones

A la hora de comparar un procesador con otro e intentar determinar qué procesador con su juego de instrucciones correspondiente es mejor, uno se da cuenta de que cada juego tiene sus ventajas y desventajas, por la parte del Z80 las instrucciones para trabajar con pila, o las instrucciones para trabajar con subrutinas son más sencillas, pero a la hora de trabajar con la ALU para realizar sumas o comparaciones es mucho más simple con MIPS32 frente a Z80.
Si que es cierto que aunque en el ejercicio 2 no ha sido necesario emplearlas, el procesador Z80 cuenta con instrucciones para decrementar e incrementar en una unidad el valor de un registro, cosa que MIPS32 no. MIPS32 tiene otras muchas ventajas como son el gran número de registros disponibles.

La práctica nos ha servido para observar cómo funciona y cómo se crea un procesador, nos ha sido de gran utilidad y creemos que nos va a servir en un futuro a la hora de programar, puesto que ahora poseemos el conocimiento de cómo funciona un procesador por dentro de manera mucho más detallada. En total estimamos que hemos podido dedicar un total de aproximadamente 14 horas de trabajo en grupo e individual a la práctica.

Problemas encontrados

A la hora de realizar la grabación comparamos el estado de los registros antes de llamar a la función y después de llamarla, de esa forma podemos observar si se suman correctamente o no los valores del vector, aunque por algún motivo que desconocemos nosotros durante la grabación vemos la tabla de comparación, pero a la hora de reproducirlo no siempre se muestra, si no se muestra se puede seleccionar manualmente durante la reproducción o se pueden observar en la tabla de registros de la derecha para ver que funciona.