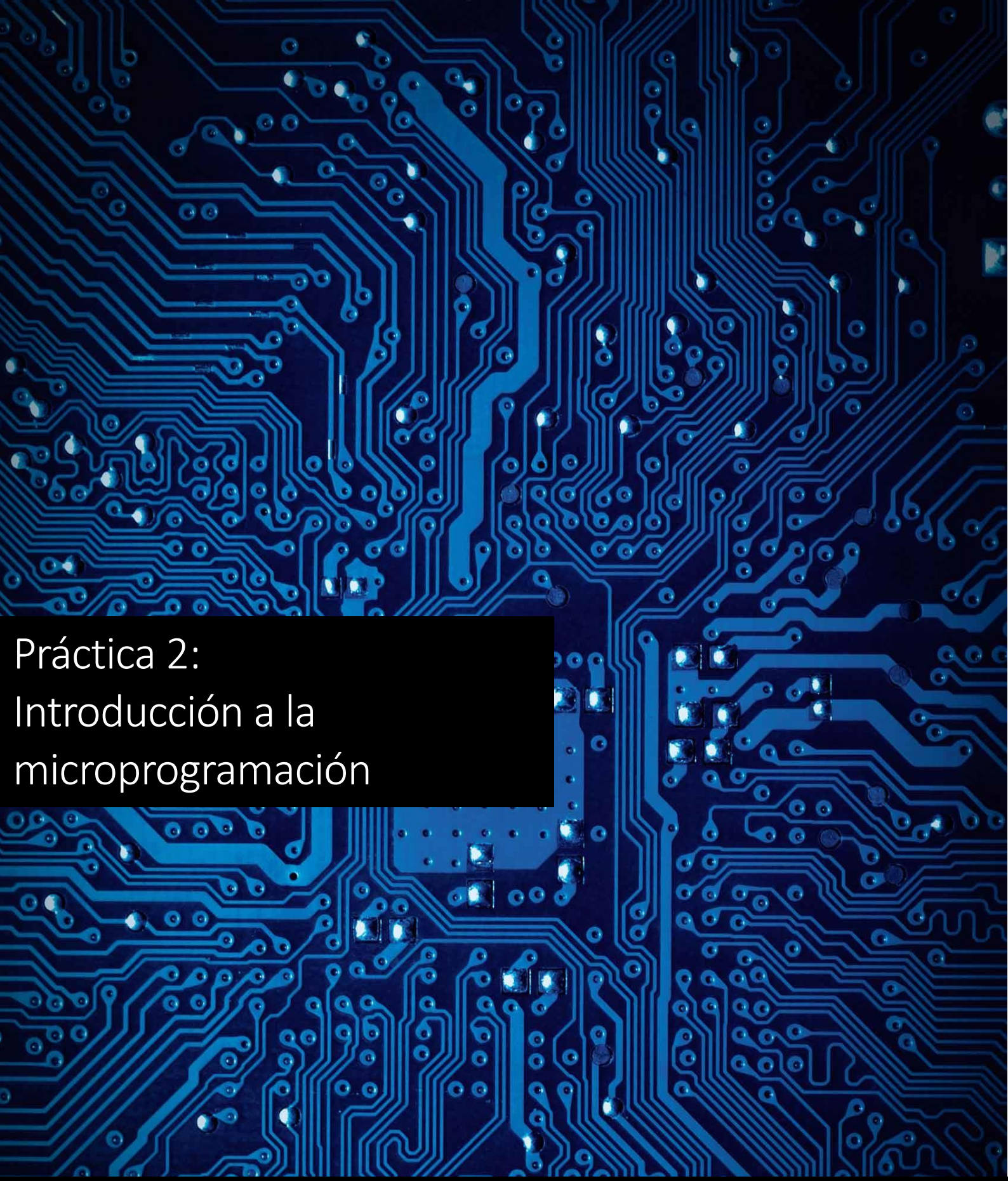

Práctica 2: Introducción a la microprogramación

Fernando Bermúdez 100405854, Grupo 80, 100405854@alumnos.uc3m.es
Pablo-Tomás Campos 100406000, Grupo 80, 100406000@alumnos.uc3m.es
Universidad Carlos III de Madrid
17-10-2019

Contenido

Ejercicio 1:

Debido a la falta de espacio en la tabla, las decisiones de diseño se encuentran al final del documento. Para llegar a ellas pulsa el hipervínculo ubicado en la fila correspondiente de *Decisiones de diseño*.

INSTRUCCIÓN	DISEÑO EN LENGUAJE RT	SEÑALES DE CONTROL	DECISONES DE DISEÑO
ld RDEST, RORIG	BR[RDEST]←BR[RORIG]	(SELA=10000, SELC=10101, T9, LC, A0, B, C=0)	Decisiones de diseño
ldi RDEST, U16	BR[RDEST]←U16	(SIZE=1111,OFFSET=0,T3,MR=0,SELC=10101,LC, A0,B,C=0)	Decisiones de diseño
ld RDEST, (RORIG)	MAR←BR[RORIG] MBR←M[MAR] BR[RDEST]←MBR	(MR=0, SELA=10000, T9, C0), (TA, R, BW=11, M1, C1), (MR=0, SELC=10101, T1, LC, A0, B, C=0)	Decisiones de diseño
add_a RORIG	RT1←BR[RORIG] BR[A]←RT1 + BR[A]	(SELA=10101, MB=11, SELCOP=1100, MC, T6, C4), (SELB=00100, MR, MA, SELCOP=1010, MC, T6, SELC=00100, SELP=11, M7, C7, LC, A0, B, C=0)	Decisiones de diseño
addi_a S16	RT1←S16 BR[A]←RT1 + BR[A]	(SIZE=01111, OFFSET=00000, SE, T3, C5), (MR, SELA=100, MB, MC, SELCOP=1010, T6, SELC=100, LC, SELP=11, M7, C7, A0, B, C=0)	Decisiones de diseño
inc RDEST	BR[RDEST]←BR[RDEST] + 1	(SELA=10101, MB=11, MC, SELCOP=1010, SELC=10101, SELP=11, M7, C7, LC, T6, A0, B, C=0)	Decisiones de diseño
dec RDEST	BR[RDEST]←BR[RDEST - 1	(MR=0, SELA=10101, MA=0, MB=11, MC, SELCOP=1011, T6, SELC=10101, LC, SELP=11, M7, C7, A0, B, C=0)	Decisiones de diseño
jp S16	RT1←PC RT2←S16 PC←RT1 + RT2	(T2, C4), (SE, OFFSET=0, SIZE=10000, T3, C5), (MA, MB, MC, SELCOP=1010, T6, C2, A0, B, C=0)	Decisiones de diseño

jpz S16	IF (SR.Z == 1) {
	microsalto a backfetch
	}
	RT1←PC
	RT2←S16
	PC←RT1 + RT2
	backfetch: salto a fetch
		(A0=0, B, C=110, MADDR=backfetch),
		
		(T2, C4),
		(SE, OFFSET=0, SIZE=10000, T3, C5),
		(MA, MB, MC, SELCOP=1010, T6, C2, A0, B, C=0)
		backfetch: (A0, B, C=0)
			Decisiones de diseño

call U16	SP←SP - 4, MAR←SP MBR←PC M[MAR]←MBR, PC←U16	(SELA=11101, MR, MB=10, MC, SELCOP=1011, T6, SELC=11101, LC, C0), (T2, M1=0, C1), (TA, BW=11, TD, W, SIZE=10000, T3, C2, A0, B, C=0)	Decisiones de diseño
ret	RT3←SP + 4, MAR←SP MBR←M[MAR], SP←RT3 PC←MBR	(MR, SELA=11101, T9, C0, MA=0, MB=10, SELCOP=1010, MC, C6), (TA, R, BW=11, M1, C1, T7, MR, SELC=11101, LC), (T1, M2=0, C2, A0, B, C=0)	Decisiones de diseño
halt	PC←BR[1] - BR[1]	(SELA, SELB, SELCOP=1011, MR, MC, T6, C2, A0, B, C=0)	Decisiones de diseño
push RORIG	SP←SP - 4 ,MAR←SP, MBR←BR[RORIG] M[MAR]←MBR	(MR, SELA=11101, MA=0, MB=10, SELCOP=1011, MC, T6, SELC=11101, LC, C0), (MR=0, SELA=10101, T9, M1=0, C1), (TA, TD, W, BW=11, A0, B, C=0)	Decisiones de diseño
pop RDEST	MAR←SP, RT3←SP + 4 MBR←M[MAR], SP←RT3 RDEST←MBR	(SELA=11101, MR, T9, C0, MB=10, MC, SELCOP=1010,C6), (TA, BW=11, R, M1, C1, SELC=11101, MR, T7, LC), (T1, SELC=10101, MR=0, LC, A0, B, C=0)	Decisiones de diseño

Decisiones de diseño

ld RDEST, RORIG

Debido a que se pide reducir al mínimo el número de ciclos en las instrucciones, hemos incorporado todo en un solo ciclo de reloj. Ejercicio 1

ldi RDEST, U16

Debido a que se pide reducir al mínimo el número de ciclos en las instrucciones, hemos incorporado todo en un solo ciclo de reloj. Ejercicio 1

ld RDEST, (RORIG)

En el primer ciclo enviamos a MAR el valor del registro Rorig, para tras esperar hasta el siguiente ciclo coger el valor de esa posición en memoria, ya que de lo contrario estaríamos atravesando el registro. De la misma forma, en el mismo segundo ciclo cogemos este valor y lo almacenamos en MBR, para finalmente en el último ciclo guardar el valor en el registro Rdest. Ejercicio 1

add_a RORIG

Para acceder a Rorig hace falta tener MR=0, mientras que para acceder al registro A hace falta tenerlo a 1. Es por esto que en un primer ciclo sacamos Rorig, lo multiplicamos por 1 en la ALU y lo guardamos en RT1. En el siguiente ciclo activamos MR sacando así A y sobrescribimos su valor con su suma con RT1 (Rorig). Ejercicio 1

addi_a RORIG

En el primer ciclo sacamos el valor inmediato y lo movemos al registro RT1, para en el siguiente ciclo operar con él en la ALU, puesto que debemos esperar a otro ciclo o estaríamos atravesando el registro. En el segundo ciclo sacamos el registro A del banco de registros y hacemos la suma. Como aún no se ha usado el bus interno en este ciclo, directamente guardamos el resultado en el registro A. Ejercicio 1

inc RDEST

Debido a que se pide reducir al mínimo el número de ciclos en las instrucciones, hemos incorporado todo en un solo ciclo de reloj. Ejercicio 1

dec RDEST

Debido a que se pide reducir al mínimo el número de ciclos en las instrucciones, hemos incorporado todo en un solo ciclo de reloj. Ejercicio 1

jp S16

Con el fin de que esta instrucción se asemeje a su contrapartida en MIPS-32 (Branch), hemos decidido que recibiera direcciones relativas a PC, de tal forma que si se ejecuta “jp b1”, PC apuntará a la etiqueta b1, al igual que en MIPS-32. Ejercicio 1

jpz S16

En esta instrucción hemos tenido la misma filosofía que con *jp*, puesn que queríamos que los saltos fueran relativos a PC. Ejercicio 1

call U16

Para ahorrar ciclos de ejecución, en el mismo ciclo en el que restamos a SP 4, lo enviamos al MAR. En el siguiente ciclo PC envía su dirección a MBR. En el siguiente ciclo, en el que se guarda PC en memoria, aprovechamos para sobrescribir PC con U16 aprovechando que el bus de datos está libre. Ejercicio 1

ret

Como en la instrucción anterior, con el fin de ahorrar ciclos, realizamos en el primer ciclo tanto la suma a SP 4, guardándola en RT3, como llevar a MAR el valor de SP. En el siguiente ciclo llevamos a MBR el contenido en memoria de la dirección almacenada en MAR. Como no se ha usado el bus interno aún, aprovechamos para en el mismo ciclo llevar a SP el valor de RT3. En el tercer ciclo llevamos a PC el valor de MBR. Ejercicio 1

halt

Dado que el objetivo es sobrescribir PC con un 0, podríamos activar C2 y cargar el contenido del bus, 0 al no haber nada activado. Sin embargo, no nos fiamos de la estabilidad eléctrica del bus, por lo que hemos tomado otra vía para generar un 0 en el bus de datos. Para ellos, restamos cualquier registro con sí mismo, sin actualizar el registro de estado, cargando el resultado de la operación en PC. Ejercicio 1

push R_{ORIG}

Con la intención de reducir el número de ciclos al mínimo, en el primer ciclo restamos a SP 4, y enviamos el resultado tanto a MAR como a SP. En el siguiente ciclo enviamos a MBR el valor del registro a guardar en pila. En el último ciclo sobrescribimos el valor en la posición de memoria contenida en MAR por el valor contenido en MBR. Ejercicio 1

pop R_{DEST}

En el primer ciclo de esta instrucción enviamos SP a MAR, a la vez que guardamos en RT3 SP+4. En el siguiente ciclo, mientras MBR recibe la dirección almacenada en MAR, SP se sobrescribe con el valor de RT3 ahorrándonos así un ciclo de ejecución. En el siguiente ciclo sobrescribimos Rdest con el valor de MBR. Ejercicio 1

Ejercicio 2

Tras correr el código pedido implementado con el juego de instrucciones del Z80, estos son los resultados obtenidos:

REGISTRO	VALOR EN DECIMAL
R5	10
R7	55
R8	10
R9	4096 (dir de *vector*)
PC	0

Para comparar el juego de instrucciones del Z80 con el de MIPS32, hemos considerado oportuno visualizar a la vez el programa en ensamblador con el juego de instrucciones de MIPS32 que se nos ha proporcionado en la práctica con su equivalente usando las instrucciones del Z80, para ver cómo se aplican las mismas funcionalidades con el nuevo juego de instrucciones, viendo directamente las ventajas y desventajas del juego de instrucciones del Z80.

Pinchando en los rectángulos de colores irá a la explicación de porque la instrucción en cuestión es mejor/peor respecto a su contrapartida en MIPS32.

```
.text
sumav:
      push $8
      push $9
      ldi $7 0
      b1:
            ldi $4 0
            add_a $8
            jpz f1
            ld $5 ($9)
            ld $4 $7
            add_a $5
            ld $7 $4
            ld $4 $9
            addi_a 4
            ld $9 $4
            dec $8
            jp b1
      f1:
            pop $9
            pop $8
            ret

main:
      ldi $8 10
      ldi $9 vector
      call sumav
      halt
```

```
.text
sumav:
      addi $sp $sp -8
      sw $a0 4($sp)
      sw $a1 0($sp)
      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
      f1:
            lw $a1 0($sp)
            lw $a0 4($sp)
            addi $sp $sp 8
            jr $ra

main:
      li $a0 10
      la $a1 vector
      jal sumav
      li $v0 10
      syscall
```

Push/Pop es más simple

Saltos condicionales más complejos

Sumar registros es más complejo

Sumar/Restar uno es más sencillo

Saltos a etiquetas indénticos

Fin de ejecución más sencillo

Comparación del juego de instrucciones

ldi:

Idéntica a la instrucción li de MIPS. Ejercicio 2

ld:

Instrucción para mover a un registro, desde otro (move) o desde memoria (lw), dependiendo de cómo se invoca. Ejercicio 2

add_a/addi_a:

Z80 no goza de instrucciones para sumar registros directamente, si no que tiene que recurrir al registro acumulador, A, como un registro temporal a la hora de hacer operaciones con registros. En este aspecto consideramos que el Z80 es bastante inferior a MIPS32. Ejercicio 2

inc/dec:

Sin embargo, el Z80 posee instrucciones más sencillas para incrementar o decrementar en uno cualquier registro, lo cual simplifica significativamente el código. Ejercicio 2

jp:

Idéntico a su contrapartida en MIPS-32 (Branch), gracias a las decisiones de diseño implementadas. Ejercicio 2

jpz:

Esta instrucción es mucho más limitada que su contrapartida en MIPS-32, donde se pueden realizar todo tipo de saltos condicionales, mientras que esta instrucción solo funciona con una única condición, la cual además es compleja de implementar. Ejercicio 2

call/ret

Estas instrucciones son idénticas a *jal* y *jr $ra* en MIPS32, pues llaman a subrutinas y retornan al punto de invocación. Ejercicio 2

halt:

Consideramos que esta es una forma más eficaz de terminar la ejecución de un programa, puesto que es más corto que hacer *syscall 10*. Ejercicio 2

pop/push:

Esta operación es de gran utilidad pues simplifica mucho el código respecto a MIPS32, en el cual push y pop se realiza en mínimo dos operaciones. Ejercicio 2

Conclusiones

Creemos que la práctica ha estado muy bien orientada, puesto que nos ha permitido aplicar y profundizar en los conceptos vistos en clase.

La única dificultad encontrada fue a la hora de realizar jp/jpz, puesto que no supimos si debería hacerse de forma relativa a PC o de forma absoluta. Al final salimos de dudas y nos decidimos a hacerla de forma relativa.

El número de horas destinadas a la práctica ronda las seis horas: cuatro fueron usadas en la realización del ejercicio 1, una en el ejercicio dos y otra en la memoria.