Estructura de computadores

Doble Grado en Ingeniería Informática y Admón. de Empresas - 2º Curso

Práctica 2:
Introducción a la microprogramación

Gonzalo Vega Sanz NIU: 100386334
e-mail: 100386334@alumnos.uc3m.es

Ernesto Vieira Manzanera NIU: 100386252
e-mail: 100386252@alumnos.uc3m.es

Contenido

1. Primer ejercicio

A. Enunciado

Diseñe, implemente y pruebe las siguientes instrucciones a añadir a las existentes de MIPS32 que se tienen inicialmente:

Instrucción	Formato	Funcionalidad asociada	Registro de estado
madd (rs1), (rs2), (rs3)	CO (31-26): 011000 R1 (25-21) R2 (20-16) R3 (15-11)	Memoria[rs1] ← Memoria[rs2] + Memoria[rs3]	Se actualiza
mmul (rs1), (rs2), (rs3)	CO (31-26): 011001 R1 (25-21) R2 (20-16) R3 (15-11)	Memoria[rs1] ← Memoria[rs2] * Memoria[rs3]	Se actualiza
mxch (rs1), (rs2)	CO (31-26): 011010 R1 (25-21) R2 (20-16)	Memoria[rs1] ↔ Memoria[rs2] Intercambia el contenido de las posiciones de memoria Memoria[rs2] y Memoria[rs1]	No se actualiza
mmv (rs1), (rs2)	CO (31-26): 011011 R1 (25-21) R2 (20-16)	Memoria[rs1] ← Memoria[rs2]	No se actualiza
mma (rs1), (rs2), (rs3)	CO (31-26): 011100 R1 (25-21) R2 (20-16) R3 (15-11)	Memoria[rs1] ← Memoria[rs1] + Memoria[rs2] * Memoria[rs3]	No se actualiza

Todas las instrucciones trabajan todas con números enteros en complemento a dos de 32 bits, y ocupan una palabra.

B. Diseño de las instrucciones

Microinstrucción	Diseño en lenguaje RT	Diseño
madd rs1 rs2 rs3	**1. RT1 ← Memoria[rs2]** - MAR ← RA (rs2) - MBR ← MP[MAR] - RT1 ← MBR **2. RT2 ← Memoria[rs3]** - MAR ← RA (rs3) - MBR ← MP[MAR] - RT2 ← MBR **3. Add RT1 + RT2. Update RS** - MBR ← RT1 + RT2 - RS ← ALU } mismo ciclo **4. MP[rs1] ← MBR** - MAR ← RA (rs1) - MP[MAR] ← MBR +*fetch*	Para el diseño de esta instrucción hemos decidido trasladar los operandos a los registros temporales RT1 y RT2 para luego guardar el resultado obtenido la dirección de memoria contenida en rs1.
mmul rs1 rs2 rs3	**1. RT1 ← Memoria[rs2]** - MAR ← RA (rs2) - MBR ← MP[MAR] - RT1 ← MBR **2. RT2 ← Memoria[rs3]** - MAR ← RA (rs3) - MBR ← MP[MAR] - RT2 ← MBR **3. Multiply RT1 * RT2. Update RS** - MBR ← RT1 * RT2 - RS ← ALU } mismo ciclo **4. MP[rs1] ← MBR** - MAR ← RA (rs1) - MP[MAR] ← MBR +*fetch*	Para el diseño de esta instrucción hemos decidido trasladar los operandos a los registros temporales RT1 y RT2 para luego guardar el resultado obtenido la dirección de memoria contenida en rs1.
mxch rs1 rs2	**1. RT1 ← MP[rs1]** - MAR ← RA - MBR ← MP[MAR] - RT1 ← MBR **2. MBR← MP[rs2]** - MAR ← RA - MBR ← MP[MAR] **3. MP[rs1] ← MBR** - MAR ← RA - MP[MAR] ← MBR **4. MP[rs2] ← RT1** - MAR ← RA - MBR ← RT1 - MP[MAR] ← MBR +*fetch*	En este caso, creamos una copia de los valores de MP[rs1] y MP[rs2] en los registros temporales RT1 y RT2, para posteriormente guardar dichos valores en MP[rs2] y MP[rs1] respectivamente
mmv rs1 rs2	**1. MBR ← MP[rs2]** - MAR ← RA - MBR ← MP[MAR] **2. MP[rs1] ← MBR** - MAR ← RA (rs1) - MP[MAR] ← MBR +*fetch*	Para esta instrucción trasladamos el valor que queremos copiar de MP[rs2] a MP[rs1] al MBR, de forma que solo es necesario cambiar las direcciones de memoria almacenadas en MAR

mma rs1 rs2 rs3	**1. RT1 ← Memoria[rs2]** - MAR ← RA (rs2) - MBR ← MP[MAR] - RT1 ← MBR **2. RT2 ← Memoria[rs3]** - MAR ← RA (rs3) - MBR ← MP[MAR] - RT2 ← MBR **3. Multiply RT1 * RT2 and load in RT1** - RT1 ← RT1 * RT2 **4. RT2 ← MP[rs1]** - MAR ← RA (rs1) - MBR ← MP[MAR] - RT2 ← MBR **5. Add RT1 + RT2 and save in MP[rs1]** - MBR ← RT1 + RT2 - MP[rs1] ← MBR +*fetch*	En este caso las primeras instrucciones corresponden al mismo proceso de multiplicar los elementos contenidos en las direcciones de memoria rs1 y rs2

2. Segundo ejercicio

A. Enunciado

Desarrolle un programa en ensamblador que haga uso de las nuevas instrucciones definidas en el ejercicio 1. Las pruebas serán realizadas desde una **única** subrutina **main.**

Cada prueba se encargará de establecer los registros a los valores que estime necesarios, ejecutará la instrucción o instrucciones a probar y comprobará si el resultado es el esperado. Cada prueba se identificará con un número unívoco empezando en uno e incrementándose en cada prueba. Si el resultado es correcto ha de imprimir la cadena “OK: “ y a continuación el ordinal que identifica a la prueba. Si el resultado no es correcto (no se cumple lo pedido en la tabla de requisitos), ha de imprimir “ERROR: “ y a continuación el identificador unívoco de la prueba.

B. Pruebas de las instrucciones

A continuación se muestra una tabla con las pruebas realizadas para cada instrucción con el fin de verificar su correcto funcionamiento. En los casos que sea necesario comprobar la actualización o no actualización del registro de estado (RS), se adjunta una captura de la tabla de cambios correspondiente a cada prueba (en rojo). En aquellas situaciones donde el registro de estado deba actualizarse, se muestra el valor anterior y el posterior a la ejecución de la instrucción. En aquellas en las que RS NO deba actualizarse, se muestra una captura de la tabla de cambios completa, con el fin de verificar que no se ha producido ningún cambio en el registro de estado.

Instrucción	ID prueba	Descripción	Resultado
madd	1	Suma de dos números positivos	OK
	6	Suma de dos números negativos. Actualización de RS	OK
	cpu	SR	= 0x350000	0xa0350000
	13	Suma de un numero positivo y otro negativo.	OK
	7	Suma con resultado 0. Actualización de RS.	OK
	cpu	SR	= 0x360000	0x10360000
mmul	2	Multiplicación de dos numero positivos	OK
	11	Multiplicación de dos números negativos. Actualización de RS	OK
	cpu	SR	= 0x303100	0x20303100
	14	Multiplicación de un numero positivo y otro negativo.	OK
	12	Multiplicación de resultado 0. Actualización de RS	OK
	cpu	SR	= 0x313100	0x10313100
mxch	3	Intercambio de dos valores aleatorios. NO actualización de RS.	OK
	cpu	PC	= 0x80b8	0x80bc
	memory	0x1010	= 0x5	0x7
	memory	0x1014	= 0x7	0x5
	15	Intercambio de un valor no nulo y un valor nulo	OK
mmv	4	Copia de un valor aleatorio	OK
	10	Copia de una posición de memoria con valor 0. NO actualización de RS.	OK
	cpu	PC	= 0x8340	0x8344
	memory	0x1014	= 0xffffffff	0
mma	5	Operación combinada con operandos positivos. NO actualización de RS.	OK
	cpu	PC	= 0x8174	0x8178
	memory	0x1010	= 0x7	0x7e
	8	Operacion combinada con operandos negativos. NO actualización de RS.	OK
	cpu	PC	= 0x8280	0x8284
	memory	0x1010	= 0xfffffffe	0xfffffffd
	9	Operacion combinada con resultado 0. NO actualización de RS	OK
	cpu	PC	= 0x82e8	0x82ec

3. Tercer ejercicio

A. Enunciado

Para el segundo conjunto de pruebas se desea comparar el uso de las instrucciones nuevas con el uso de las instrucciones iniciales de MIPS32.

En concreto hay que realizar dos pruebas y para cada prueba hay que contabilizar el número de ciclos de reloj y el número de instrucciones que se ejecutan. Las dos pruebas hacen lo mismo: multiplicar dos matrices A y B y guardar el resultado en la matriz C, todas ellas son cuadradas de dimensión 3.

La primera prueba se hará mediante un programa en ensamblador que usará únicamente las instrucciones iniciales, es decir, utilizando exclusivamente las instrucciones del MIPS32 sin las añadidas en el ejercicio 1.

La segunda prueba se hará mediante un programa en ensamblador que usará tanto las instrucciones iniciales como las añadidas en el ejercicio 1, pero no podrá hacer uso de las instrucciones ADD y MULT del MIPS32 proporcionadas inicialmente sino usando las nuevas (añadidas en el ejercicio 1).

B. Primera prueba

Código ensamblador	# Ciclos	# Instrucciones

.data
matrizA: .word 4,6,3,4,5,6,7,8,9
matrizB: .word 9,8,7,6,7,4,3,2,99
destino: .space 36
.text
main:
#declaro variables
la $s3 destino
la $t0 matrizA
la $t3 matrizB
move $t4 $zero #comparador a 3
li $t5 3 #para tener el numero 3 que me va a servir ya que las matrices son de 3*3
li $s1 1 #para sumar 1
li $s2 0 #contador destino
li $s4 0#contador filas
li $s5 0#contador programa
li $s0 12#para ir al siguiente elemento
li $t6 4#para ir al elemento de la fila de abajo
move $t7 $zero#lo voy a usar para hacer cálculos durante el programa
 	
for1:
#Es el bucle que hará la operación de multiplicar una fila de la matriz A por una columna de la matriz B
beq $t4 $t5 cambiocolumnaB #va a cambio columna b porque ya habrá calculado el primer numero de la matriz y así en bucle
lw $t1, ($t0) #al no poder hacer mma guardo los valores de las direcciones en registros
lw $t2, ($t3)
mul $t1,$t1,$t2
add $t7, $t7, $t1
 	
add $t0 $t0 $t6 #para el siguiente elemento de la fila correspondiente de A
add $t3 $t3 $s0 #para el siguiente elemento de la columna correspondiente de B
add $t4 $t4 $s1 #contador +1
b for1
 	
cambiocolumnaB:#Este bloque de código se encarga de cambiar la columna de la matriz B manteniendo todavía la fila de la matriz A
#Ademas al llegar a tres el contador ira a cambiuarfilaA y este se reiniciará
 	
sw $t7 ($s3)#mete el resultado en la matriz
add $s3 $s3 $t6 #le suma 4 a la matriz s3 para que el siguiente elemento vaya en la siguiente posición
 	
move $t7 $zero #restablezco t7
move $t4 $zero #restablezco t4
add $s4 $s4 $s1 #añado 1 a s4 que cuando llegue a 3 significaría que la fila ya ha multiplicado a las 3 columnas
beq $s4 $t5 cambiofilaA #e ira a cambio de fila
 	
la $t0 matrizA#si no solo hay que cambiar la columna de B entonces pongo el primer elemento de la fila correspondiente
mul $t7 $s0 $s5#con esta y la siguiente linea podemos saber en que fila de a estamos estos elementos cambian en cambiofilaA
add $t0 $t0 $t7
la $t3 matrizB # con esto pongo la columna correspondientes s4 es
mul $t7 $t6 $s4
add $t3 $t3 $t7
move $t7 $zero
b for1
 	
cambiofilaA:#Es el encargado de al haber multiplicado una fila de la matriz A a las tres columnas de la matriz B cambiar la fila de la matriz A al llegar a 3 el programa se acabará
add $s5 $s5 $s1#contador que cuando llega a tres es que ya todas las filas han multiplicado a todas las columnas
beq $s5 $t5 fin
 	
la $t3 matrizB #pone a t3 en la primera columna
la $t0 matrizA #Y a la matriz A la mueve a la siguiente fila
mul $t7 $s0 $s5
add $t0 $t0 $t7
move $s4 $zero#ponemos el contador de las columnas a 0
move $t7 $zero#y el valor t7 para calcular tambien a zero ya que se usa en multiples operaciones
b for1
 	
fin:
#Cierro el programa
jr $ra
	2737	387

C. Segunda prueba

Código ensamblador	# Ciclos	# Instrucciones
.data matrizA: .word 4,6,3,4,5,6,7,8,9 matrizB: .word 9,8,7,6,7,4,3,2,99 destino: .space 36 .text main: la $s3 destino la $t0 matrizA la $t3 matrizB move $t4 $zero #comparador a 3 li $t5 3#para tener el numero 3 que me va a servir ya que las matrices son de 3*3 li $s1 1#para sumar 1 li $s2 0 #contador destino li $s4 0#contador filas li $s5 0#contador programa li $s0 12#para ir al siguiente elemento li $t6 4#para ir al elemento de la fila de abaj move $t7 $zero #lo voy a usar para hacer cálculos durante el programa for1: #Es el bucle que hará la operación de multiplicar una fila de la matriz A por una columna de la matriz B beq $t4 $t5 cambiocolumnaB #va a cambio columna b porque ya habrá calculado el primer numero de la matriz y así en bucle mma $s3 $t0 $t3 add $t0 $t0 $t6#para el siguiente elemento de la fila correspondiente de A add $t3 $t3 $s0 #para el siguiente elemento de la columna correspondiente de B add $t4 $t4 $s1 #contador +1 b for1 cambiocolumnaB:#Este bloque de código se encarga de cambiar la columna de la matriz B manteniendo todavía la fila de la matriz A #Ademas al llegar a tres el contador ira a cambiarfilaA y este se reiniciará add $s3 $s3 $t6#le suma 4 a la matriz s3 para que el siguiente elemento vaya en la siguiente posición move $t4 $zero#restablezco t4 add $s4 $s4 $s1 #añado 1 a s4 que cuando llegue a 3 significaría que la fila ya ha multiplicado a las 3 columnas beq $s4 $t5 cambiofilaA #e ira a cambio de fila la $t0 matrizA #si no solo hay que cambiar la columna de B entonces pongo el primer elemento de la fila correspondiente mul $t7 $s0 $s5#con esta y la siguiente linea podemos saber en que fila de a estamos estos elementos cambian en cambiofilaA add $t0 $t0 $t7 la $t3 matrizB # con esto pongo la columna correspondientes s4 es mul $t7 $t6 $s4 add $t3 $t3 $t7 b for1 cambiofilaA:#Es el encargado de al haber multiplicado una fila de la matriz A a las tres columnas de la matriz B cambiar la fila de la matriz A al llegar a 3 el programa se acabará add $s5 $s5 $s1#contador que cuando llega a tres es que ya todas las filas han multiplicado a todas las columnas beq $s5 $t5 fin la $t3 matrizB #pone a t3 en la primera columna la $t0 matrizA #Y a la matriz A la mueve a la siguiente fila mul $t7 $s0 $s5 add $t0 $t0 $t7 move $s4 $zero #ponemos el contador de las columnas a 0 b for1 fin: #cierro el programa jr $ra	2266	280

D. Análisis de los resultados

Comparando los resultados obtenidos en ambas pruebas, vemos una reducción del 27,64% en el número de instrucciones ejecutadas de la primera a la segunda prueba; y de un 17,21% en el número de ciclos empleados.

Estos datos muestran claramente la mayor eficiencia de realizar las operaciones con las nuevas instrucciones que con las antiguas. No obstante, estas simulaciones de ejecución se han realizado sobre un modelo donde las operaciones de lectura y escritura en la memoria se realizaban en el mismo ciclo. Debemos tener en cuenta que en un modelo real estos tiempos son superiores y, si bien el número de instrucciones a ejecutar es el mismo, el número de ciclos empleados para ejecutar el programa aumenta de forma proporcional al número de accesos a la memoria.

4. Conclusiones

Una vez finalizado el trabajo hemos podido comprobar lo esencial que es diseñar un correcto juego de instrucciones con el fin de reducir los tiempos de ejecución y aumentar la eficiencia del programa. Creemos que una de las partes más complicadas de el diseño de un microcódigo es el de la optimización para disminuir el número de ciclos. Aunque pensamos que los ejemplos de instrucciones que hemos dado en el trabajo son bastante óptimas, es posible que quepa una mejor implementación que mejore el rendimiento de los programas en lo que las empleemos.

Por otro lado, en la realización de este trabajo se han dedicado una media de 15 horas de las cuales la mayor parte se han dedicado al diseño y realización de las pruebas para comprobar el funcionamiento adecuado de las instrucciones pedidas; unas 2 horas fueron empleadas para el diseño de las instrucciones en microcódigo; aproximadamente 3 para el segundo ejercicio; y el resto para la memoria y arreglar posibles errores de código.