GUAYMAS MATIAS

PRÁCTICA 5

HOJA N° 1

FECHA

1) a) 16 ciclos, 7 instrucciones y 2,286 CPI.

b) Se generan 4 ataques RAW

- L.D F2, n2(r0)

ADD.D F3, F2, F1 → Esta instrucción tiene que esperar a que la anterior escriba en F2. (1 RAW)

- ADD.D F3, F2, F1

MUL.D F4, F2, F1

S.D F3, n51 (r0) → Esta instrucción tiene que esperar a que la suma termine (etapa A3) para usar F3 (2 RAW)

- MUL.D F4, F2, F1

S.D F3, n51 (r0)

S.D F4, n52(r0) → La instrucción tiene que esperar a que MUL.D termine (etapa M6) para usar F4 (1 RAW)

c) Los ataques estructurales son provocados por conflictos por los recursos. En WinMips puede suceder cuando dos instrucciones en unidades de ejecución distintas intentan acceder a la etapa MEM simultáneamente

- ADD F3, F2, F1 ⟶ La etapa MEM de la instrucción S.D se encuentra al mismo tiempo con la etapa MEM

MUL.D F4, F2, F1 de ADD.D y se produce un ataque estructural.

S.D F3, n51 (r0)

- MUL.D F4, F2, F1 ⟶ La etapa MEM de la instrucción S.D se encuentra al mismo tiempo con la etapa MEM de

S.D F3, n51 (r0) MUL.D y se produce un ataque estructural

S.D F4, n52 (r0)

d) Aparece un ataque WAR ya que F1 será usado en escritura y hay que esperar que termine de usarse la instrucción anterior en lectura.

e) AD: Termina en una sola etapa porque F1 y F2 están disponibles por encontrarse el NOP. Al terminar AD en una etapa no hay WAR con la instrucción siguiente.

2) MTC1 rf, fd : copia los 64 bits del registro entero rf al registro fd de punto flotante.

- MFC1 rd, ff : copia los 64 bits del registro ff de punto flotante al registro rd entero.

- CVT.D.L fd, ff : convierte a punto flotante el valor entero copiado al registro ff, dejándolo en fd

- CVT.L.D fd, ff: convierte a entero el valor en punto flotante contenido en ff, dejándolo en fd.

3) HECHO EN LA PC.

4) HECHO EN LA PC.

5) El procesador MIPS64 posee 32 registros, de 64 bits cada uno, llamados r0 a r31 (también conocidos como $0 a $31). Sin embargo, resulta más conveniente para los programadores darles nombres más significativos a esos registros. La siguiente tabla muestra la convención empleada para nombrar a los 32 registros mencionados:

Registros	Nombres	¿Para que se los utiliza?	¿Preservado?
r0	$zero	Siempre tiene el valor 0 y no puede cambiar	
r1	$at	Assembler Temporary - Reservado para ser usado por el ensamblador	
r2-r3	$v0-$v1	Valores de retorno de la subrutina llamada	
r4-r7	$a0-$a3	Argumentos pasados a la subrutina llamada.	
r8-r15	$t0-$t7	Registros temporarios. No son conservados en el llamado a subrutinas	
r16-r23	$s0-$s7	Registros salvados durante el llamado a subrutinas.	X
r24-r25	$t8-$t9	Registros temporarios. No son conservados en el llamado a subrutinas	
r26-r27	$k0-$k1	Para uso del kernel del sistema operativo	
R28	$gp	Global Pointer - Puntero a la zona de la memoria estática del programa	X
R29	$sp	Stack Pointer - Puntero al tope de la pila.	X
R30	$fp	Frame Pointer - Puntero al marco actual de la pila.	X
R31	$ra	Return Adress - Dirección de retorno en un llamado a subrutinas	X

5) HECHO EN LA FOTOCOPIA

6) a) El programa calcula 16 elevado a la cuarta potencia y almacena el resultado en la etiqueta "resultado". En valor1 guarda la base de la potencia y en valor2 guarda el exponente. Luego carga los valores en los registros y salta a una subrutina, que se encarga de hacer la potencia 16^4 y trae el resultado en un registro. Luego se guarda en la posición de memoria "result". Se emplea subrutinas y convención de registros.

b) La instrucción JAL salta a la dirección rotulada "a la potencia" y copia en $ra la dirección de retorno. Por otra parte, JR salta a la dirección contenida en $ra.

c) En $ra se almacena la dirección de retorno en un llamado a subrutina. En este caso $ra contiene el valor "c" los registros $a0 y $a1 son los argumentos/parámetros pasados a la subrutina llamada. Por último, $v0 contiene el valor de retorno de la subrutina llamada.

d) Se plantea el caso de múltiples subrutinas. Para que cada una de las subrutinas sepa a qué dirección de memoria deben retornar es guardando el $ra de la primer subrutina usando la pila, pusheando dicho registro, y una vez que se retorna de la segunda subrutina hacer un pop del $ra.

Aclaración: para usar la pila debemos inicializar el Stack Pointer: DADDi $sp, $zero, 0x4000

- para pushear: DADDi $sp, $sp, -8
  SD $ra, 0($sp)
- para pop: LD $ra, 0($sp)
  DADDi $sp, $sp, 8

En caso de ser varias subrutinas no se necesita guardar $ra en la última subrutina a ejecutar.

7) HECHO EN LA PC

8) HECHO EN LA PC

9) HECHO EN LA PC

11) HECHO EN LA PC

11) HECHO EN LA PC

12) a) HECHO EN LA PC

b) EN LA SIG HOJA.

(*) Es posible escribir una subrutina sin utilizar la pila ya que cuando llamamos a una subrutina el ra se va modificado y para no alterarlo, en lugar de una llamada recursiva, se podría utilizar un bucle simple para realizar el cálculo. Es usar $v0 tanto como operando y resultado, e ir multiplicando por sí mismo guardado en $a0, decrementándolo hasta llegar a 0.