Aspectos Avanzados de Arquitectura de Computadoras y Taller de Arquitectura de Computadoras

Prueba escrita correspondiente al curso 2013

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique el total de hojas en la primera. Escriba las hojas de un solo lado.**
- **Solo se responderán dudas de letras. No se responderán preguntas en los últimos 30 minutos de la prueba.**
- **La prueba es individual y sin material. Duración de la prueba : 2 horas y 30 minutos.**
- **Todas las preguntas tienen el mismo puntaje y el mínimo de aprobación es de un 60%.**

Pregunta 1

Explique qué es una dependencia de tipo WAW y en que tipo de procesadores aparece. Describa la técnica estudiada en el curso que se implementa en hardware para disminuir su efecto negativo sobre el rendimiento del pipeline.

Solución Pregunta 1

Una dependencia de tipo WaW es una dependencia de datos, que ocurre cuando dos instrucciones escriben en el mismo registro o posición de memoria. Si por alguna razón dos instrucciones con este tipo de dependencia escriben sus resultados en desorden, se violará el flujo de datos del programa. Asumiendo que los registros o memoria son escritos en una única etapa en el pipeline, un hazard de datos provocado por esta dependencia puede darse sólo en procesadores con la capacidad de ejecutar fuera de orden (superescalares).

La técnica estudiada en el curso para disminuir su efecto negativo en el rendimiento del pipeline es register renaming, la cual consiste en disponer en hardware de un conjunto de registros más amplio e invisible al programador y asignar dinámicamente los resultados de las operaciones a estos nuevos registros, en lugar de en el set de registros de la arquitectura. Además se debe mantener en todo momento la información de qué registro físico contiene el dato correcto de cada uno de los registros visibles de la arquitectura. Con esta técnica dos instrucciones en dependencia WaW pueden escribir sus resultados en cualquier orden, sin alterar el flujo de datos del programa.

Pregunta 2

Describa qué información se envía / recibe a través de cada uno de los tres buses del sistema cuando la CPU lee/escribe un byte de un puerto de entrada / salida, distinguiendo entre direccionamiento aislado y mapeado a memoria.

Solución Pregunta 2

Tanto para direccionamiento aislado y mapeado a memoria, en el bus de direcciones se envía el número de puerto de E/S, y en el bus de datos se envían/reciben los datos a leer/escribir.

Con respecto al bus de control, en ambas modalidades se deben activar señales que permitan

identificar si se está realizando una lectura o escritura. En el caso de direccionamiento aislado, también se debe activar una señal (típicamente llamada IO_RQ) que permita distinguir que no se está accediendo a la memoria, sino que es una operación de E/S. En el caso de direccionamiento mapeado a memoria, como las operaciones para acceder a memoria son las mismas que para acceder a los dispositivos de E/S, se debe activar una señal (típicamente llamada MEM_RQ) que indique que se está accediendo a memoria. Si se intenta hacer un acceso a una dirección de memoria mapeada a un puerto de E/S, es responsabilidad del hardware que los datos que se envíen/reciban provengan del dispositivo y NO de la memoria.

Pregunta 3

Exprese la generalización de la Ley de Amdahl en téminos de *fracción afectada* y *speedup afectado*, y deduzca la limitante de la aceleración o *speedup total* que se puede lograr.

Solución Pregunta 3

La generalización de la ley de Amdahl plantea que la aceleración (speedup) debido a una mejora E se puede expresar como $speedup_{total} = \frac{T\,\text{sin}\,mejora}{T\,con\,mejora} = \frac{T_{old}}{T_{new}}$ donde la mejora E se aplica sobre una fracción del tiempo F. Por lo tanto:

$$T_{old} = T_{no\ afectado} + T_{afectado\ old}$$

$$T_{new} = T_{no\ afectado} + T_{afectado\ new}$$

$$T_{new} = T_{no\ afectado} + \frac{T_{afectado\ old}}{speedup_{afectado}} \quad donde\ speedup_{afectado} = \frac{T_{afectado\ old}}{T_{afectado\ new}}$$

$$T_{new} = (1-F) * T_{old} + \frac{F * T_{old}}{speedup_{afectado}}$$

$$speedup_{total} = \frac{T_{old}}{T_{new}} = \frac{1}{(1-F) + \frac{F}{speedup_{afectado}}}$$

La limitante en el speedup total esta dada por la fracción afectada, es decir cuanto puedo mejorar esta limitado a cuanto es la fracción que se mejora. Este límite se puede hallar haciendo:

$$\lim_{speedup_{afectado} \to \infty} speedup_{total} = \frac{1}{(1-F)}$$

Referencias: Diapositiva Clase 5 – Rendimiento. Páginas 20-23.

Ejercicio 1

Se dispone de una lista enlazada de caracteres representada mediante un array. Cada elemento del array contiene un carácter y el índice (dentro del array) del siguiente elemento de la lista. La lista vacía se representa mediante un índice con valor negativo. Específicamente, la variable Lista se define mediante la siguiente declaración:

```
struct Item {
      char c;
      int siguiente;
}
typedef TipoLista = array [0 … N-1] of Item;
TipoLista Lista;
```

La siguiente función recursiva construye una cadena de caracteres representada de acuerdo a la convención del lenguaje C, a partir del contenido de la variable Lista:

```
void ConstruirCadena(int indiceLista, char * bufferDestino) {
      if (indiceLista < 0)
            *bufferDestino = '\0';
      else {
            *bufferDestino = Lista[indiceLista].c;
            ConstruirCadena(Lista[indiceLista].siguiente, bufferDestino + 1);
      }
}
```

Compilar en ensamblador de Intel 8086. El parámetro indiceLista se pasa en el registro BX y la variable Lista está apuntada por DS:SI. El parámetro bufferDestino se pasa en la pila como un desplazamiento con respecto al segmento apuntado por ES y la función no retira este parámetro del stack. Es decir, una llamada a la función se realiza mediante los siguientes pasos:

1- Se asigna el valor del parámetro indiceLista al registro bx

2- Se coloca bufferDestino (desplazamiento con respecto a ES) en el tope de la pila.

3- Se llama a ConstruirCadena

4- Se retira el parámetro bufferDestino de la pila

Solución Ejercicio 1

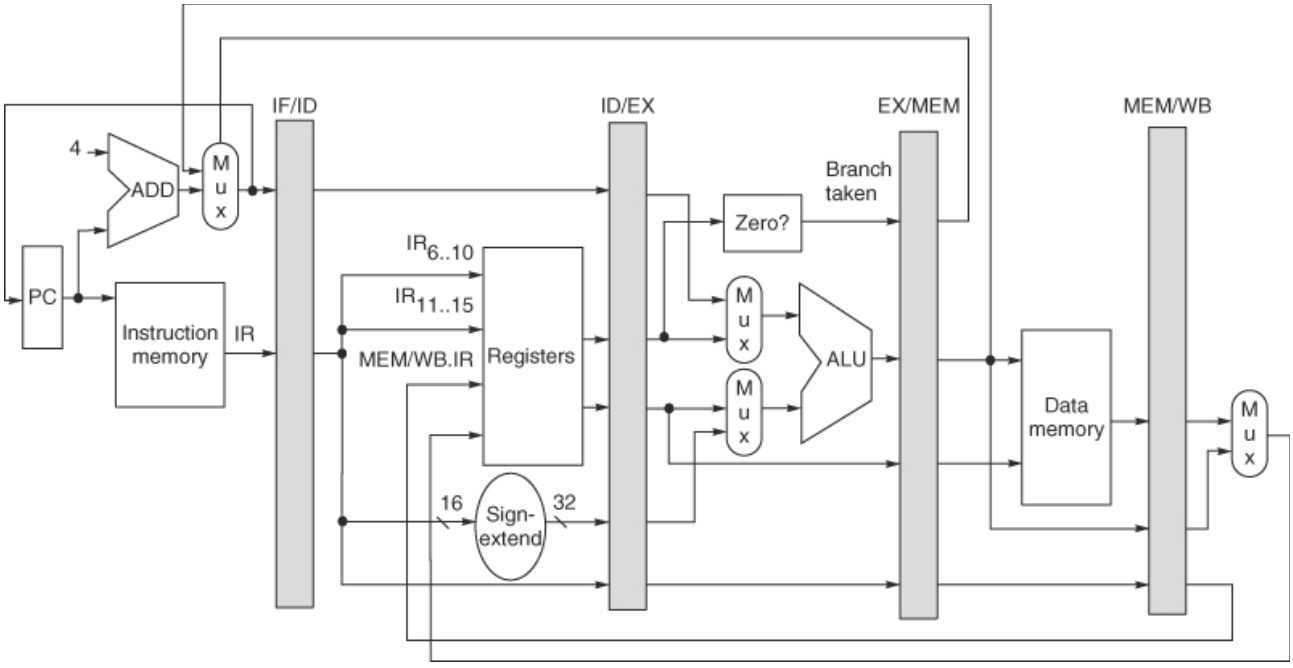
```
ConstruirCadena proc
	push bp
	mov bp, sp
	mov di, [bp + 4]	; se obtiene bufferDestino en di
	or bx, bx		; indiceLista pasa por ALU
	js paso_base		; si el indiceLista < 0

	;se multiplica índice por tamaño del nodo (3 bytes)
	mov ax, bx
	add bx, ax
	add bx, ax

	mov al, [si + bx]		; al =  Lista[indiceLista].c
	mov es:[di], al			; *bufferDestino = al
	mov bx, [si + bx + 1]		; bx = Lista[indiceLista].siguiente
	inc di				; di = bufferDestino + 1
	push di				; parámetro para llamada recursiva
	call ConstruirCadena		; llamada recursiva
	pop di				; se retira parámetro de la pila
	jmp fin
paso_base:
	mov es:[di], byte ptr 0	; marca de fin de cadena
fin:
	pop bp
	ret
ConstruirCadena endp
```

Ejercicio 2

Considere un computador con un procesador MIPS como el de la figura, y las siguientes características:



- Las memoria de instrucciones tiene una latencia menor que un ciclo de reloj. La memoria de datos consta de una cache con hit time menor que un ciclo de reloj, y luego la memoria principal. La latencia en memoria es tal que un miss en la cache provoca una detención completa del pipeline durante un ciclo.

- La cache de datos tiene una organización de correspondencia directa. Cuenta con 128 líneas y un tamaño de bloque de 128 bytes.

Considere la ejecución del siguiente fragmento de código MIPS con la memoria cache inicialmente vacía.

```
li $9, 0x1000
lw $10, 0($9)
lw $11, 4($9)
addu $12, $11, $10
addiu $9, $9, 0x4000
sw $12, 0($9)
```

a) Explique, para cada acceso a memoria, si ocurre un hit o un miss en la cache.

b) Realice un diagrama mostrando, para cada ciclo de reloj, en qué etapa del pipeline se encuentra cada instrucción. Recuerde que el hardware utilizado es el de la figura. Suponga que un valor

escrito en el banco de registros por una instrucción en la etapa WB puede ser leído durante el mismo ciclo de reloj por otra instrucción en la etapa ID.

Solución Ejercicio 2

a) Como la cache de datos tiene 128 líneas y 128 bytes por línea, los 7 bits menos significativos de la dirección indican el byte dentro del bloque a obtener, los siguientes 7 bits seleccionan la línea con la que se corresponde el bloque a acceder, y los restantes bits forman la etiqueta.

El primer acceso a memoria se realiza a la dirección 0x1000 (= 1000000000000_b). La misma causará un compulsory miss (obligatorio) por estar la cache 'vacía', y provocará que se traiga el bloque correspondiente y se ubique en la línea nro 32 (line = 100000_b).

El segundo acceso a memoria se realiza a la dirección 0x1004 (= 1000000000100_b). El bloque a buscar se encuentra en la línea nro 32, y como el tag coincide con el del bloque que se encuentra ahí (tag = 0), el acceso es un hit.

El último acceso a memoria se realiza a la dirección 0x5000 (= 101000000000000_b). Este acceso es a la misma línea, pero el tag varía (en este caso vale 1), por lo tanto provocará un miss por colisión.

b)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
li	IF	ID	EX	Mem	WB													
lw1		IF	ID	ID	ID	EX	Mem	Mem	WB									
lw2			IF	IF	IF	ID	EX	EX	Mem	WB								
addu						IF	ID	ID	ID	ID	EX	Mem	WB					
addiu							IF	IF	IF	IF	ID	EX	Mem	WB				
sw											IF	ID	ID	ID	EX	Mem	Mem	WB