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

Prueba escrita correspondiente al curso 2015

- **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 cómo funciona la técnica de Loop Unrolling y cómo ayuda a generar ILP. Además indique: ¿Se trata de una técnica estática o dinámica? ¿Por qué?

Respuesta:

Desenrrollar el loop ("loop unrrolling") consiste en ejecutar varias iteraciones sucesivas dentro de un loop externo con menos iteraciones, para explotar el "paralelismo a nivel de loop", es decir, la posibilidad de solapar la ejecución de distintas iteraciones en un loop. Se busca eliminar dependencias para explotar paralelismo, tanto de datos como de control (menos iteraciones generan menos bifurcaciones). Es una técnica habitualmente utilizada en tiempo de compilación, es decir, estática.

Pregunta 2

Suponga que se tiene un Pipeline de 5 etapas con las siguientes duraciones de las mismas:

Etapa 1: 15 ns
Etapa 2: 20 ns
Etapa 3: 60 ns
Etapa 4: 30 ns
Etapa 5: 20 ns.

Para mejorar el rendimiento del pipeline, se proponen dos mejoras del CPU con los siguientes efectos:

Mejora 1: Disminuye en 10 ns la duración de cada etapa
Mejora 2: Diminuye en 50 ns la duración de la etapa 3

Explique cuál de las dos es mejor para el rendimiento de la CPU (asuma caso ideal, sin hazards). Justifique.

Respuesta:

En un pipeline ideal la aceleración es igual al número de etapas del mismo, mientras que la latencia está limitada por la de mayor duración. Por lo tanto conviene minimizar la duración de la etapa más lenta.

Con la Mejora 1, la nueva distribución de tiempos es:

Etapa 1: 5 ns
Etapa 2: 10 ns
Etapa 3: 50 ns
Etapa 4: 20 ns
Etapa 5: 10 ns.

Es decir, la etapa más lenta emplea 50ns.

Con la Mejora 2, la nueva distribución de tiempos es:

Etapa 1: 15 ns
Etapa 2: 20 ns
Etapa 3: 10 ns
Etapa 4: 30 ns
Etapa 5: 20 ns.

Es decir, la etapa más lenta emplea 30ns.

Por lo tanto, el mejor rendimiento se logra con la Mejora 2.

Pregunta 3

Describa cómo se traduce una dirección lógica en una dirección física:
a) en un procesador 8086 con segmentación.
b) en un procesador 80386 (ia32) con segmentación y paginación.

En ambos casos, describa las modificaciones que introduce el uso de caché de instrucciones y datos, y caché de páginas (si corresponde).

Respuesta:

a) La dirección lógica segmentada se compone con un registro de segmento (CS, DS, ES, SS) y un registro offset dentro de ese segmento (ambos de 16 bits). Para construir una dirección física de 20 bits (la cantidad de bits de dirección de la arquitectura 8086), se calcula:
dir. física = segmento*16+offset
es decir, se hace un corrimiento a la izquierda de 4 bits del registro de segmento y se le suma el offset.

Cuando se utiliza caché, los bits de la dirección física se interpretan como:
tag | línea o conjunto | byte

Si hay un hit se carga la palabra de la caché, y si hay un miss se usa la dirección del bloque para acceder a memoria principal.

En 8086 no existe paginación a nivel de la arquitectura.

b) En ia32 también hay direcciones lógicas segmentadas. Los segmentos de 16 bits (CS, DS, ES, SS, FS, GS) son índices a una tabla de descriptores de segmento, que contiene la "base address" de 32 bits, a la que se suma un offset, también de 32 bits (la cantidad de bits de dirección de la arquitectura ia32). Esta suma constituye la "dirección lineal" de 32 bits. Esta dirección lineal se usa como entrada al mecanismo de paginación, que construye una dirección física interpretando la dirección lineal de la siguiente forma:
índice en directorio de páginas | índice en tabla de páginas | offset en página de mem. física

El mecanismo de paginación agrega la posibilidad de que exista un fallo de página, y para acelerar la traducción de direcciones lineales a direcciones físicas se incorpora una caché de páginas denominada "Translation Lookaside Buffer".

En ia32, se direcciona la caché (de instrucciones y datos) con la dirección física, y luego se aplica el mismo mecanismo que se describió anteriormente.

Ejercicio 1

Suponga las siguientes estructuras de datos de C para modelar archivos y directorios:

```
struct archivo{
	char* nombreArchivo;
	// Otros campos de total 30 bytes
}
struct directorio{
	int cantidadArchivos;
	archivo* archivos;
	directorio* directoriosHijos;
	char[256] nombreDirectorio;
	// Otros campos de total 256 bytes
}
```

Suponga la siguiente función para encontrar un archivo en un directorio.

```
bool existeArchivo(Directorio directorio, char* nombreArchivo){
	bool encontre = false;
	int i = 0;
	while (!encontre && i < directorio.cantidadArchivos){
		// Comparo nombre de archivo
		int j = 0;
		while (directorio.archivos[i].nombreArchivo[j] != 0 &&
			directorio.archivos[i].nombreArchivo[j] == nombreArchivo[j]){
			j++;
		}
		// Si llegué al final de los dos nombres es porque son iguales
		if (directorio.archivos[i].nombreArchivo[j] == 0 && nombreArchivo[j] == 0){
			encontre = true;
		}
		i++:
	}
	return encontre;
}
```

Se pide:

Compilar la función a assembler 8086, sabiendo que el directorio y todas las estructuras de datos se encuentran dadas como desplazamientos con respecto al registro ES. El parámetro directorio se recibe en el registro BX y la variable 'nombreArchivo' en el registro AX. El resultado se debe retornar en CX.

Solución:

```
existeArchivo PROC
      MOV SI, AX        ; ES:[SI] = nombreArchivo
      XOR CX, CX        ; encontre = false
      XOR DI, DI        ; i = 0
while:
      CMP CX, 1         ; encontre == true?
      JE fin
      CMP DI, ES:[BX]   ; i < directorio.cantidadArchivos
      JGE fin
      PUSH BX           ; preservo punteros porque son usados para recorrer
      PUSH DI           ; las estructuras
      PUSH SI
      MOV BX, ES:[BX + 2] ; BX = directorio.archivos
      SHL DI, 1
      SHL DI, 1         ; convierto índice i en puntero a estructura
      SHL DI, 1         ; multiplico por 32 (= 2^5)
      SHL DI, 1
      SHL DI, 1         ; ES:[BX + DI] = directorio.archivos[ï]
      MOV DI, ES:[BX + DI]   ; DI = directorio.archivos[i].nombre
WHILE2:
      ; directorio.archivos[i].nombreArchivo[j] != 0
      CMP byte ptr ES:[DI], 0
      JE finwhile2
      ;directorio.archivos[i].nombreArchivo[j] != nombreArchivo[j]
      MOV DL, ES:[DI]
      CMP DL, ES:[SI]   ; uso DL como auxiliar para no comparar memoria memoria
      JNE finwhile2
      INC DI
      INC SI            ; tanto DI como SI cumplen rol de j
      JMP WHILE2
finwhile2:
      CMP byte ptr ES:[DI], 0
      JNE else
      CMP byte ptr ES:[SI], 0
      JNE else
      MOV CX, 1         ; encontre = true
else:
      POP SI            ; recupero puntero a nombreArchivo
      POP DI            ; recupero i
      POP BX            ; recupero puntero a directorio
      INC DI            ; i++
      JMP WHILE
finwhile:
```

```
    RET
ENDP
```

Ejercicio 2

Considere un CPU de 16 bits, que emite direcciones de 16 bits (la cual permite direccionar completamente la memoria principal), donde la memoria es direccionable por bytes. La CPU opera con una memoria RAM de 32 bloques y una memoria Cache de 8 líneas.

a) Indique cómo se interpreta la dirección de memoria en la memoria cache para una función de correspondencia asociativa de 2 vías.

b) Suponga que la ejecución de una instrucción de memoria demora 2 ciclos en caso de que ocurra hit en el cache y 5 ciclos en caso de que ocurra miss. Suponiendo un hit rate de 90%, calcule el CPI de las instrucciones de memoria.

c) Suponga una carga de trabajo de 20 millones de instrucciones, de las cuales el 40% son instrucciones de memoria y en donde el resto de las instrucciones tiene CPI = 3. Sabiendo que la carga de trabajo ejecuta en 1 segundo, calcule la frecuencia de trabajo del CPU.

Solución:

a) Como se indica que 16 bits permite direccionar completamente la memoria principal, sabemos que esta tiene 64 KB. Como esta tiene 32 bloques, cada bloque ocupa 2KB, y como se sabe que la memoria es direccionable por bytes, se precisan 11 bits de la dirección para direccionar dentro del bloque.

   Como el cache tiene 8 líneas y se utiliza una función de correspondencia asociativa por conjuntos de 2 vías, hay en total 4 conjuntos con 2 líneas cada uno, por lo tanto se necesitan 2 bits de la dirección para ubicar al conjunto

   De lo anterior resulta que la dirección se interpreta de la siguiente manera:

   dir: tag | conjunto | byte , donde byte ocupa los bits 0 a 10 (11 bits) de la dirección, conjunto los bits 11 a 12 (2 bits) y tag los bits 13 a 15 (3 bits)

b) CPImemoria = 0.9 * 2 + 0.1 * 5 = 2.3 ciclos por instrucción

c) CPIglobal = CPImemoria * %InstMemoria + CPIotrasinstrucciones * %OtrasInstrucciones

   CPIglobal = 2.3 * 0.4 + 3 * 0.6 = 2.72 ciclos por instrucción

   Tcpu = instrucciones_ejecutadas * CPIinstrucciones * t_clk

   Tcpu = 1 s = 20.000.000 instrucciones * 2.72 * t_clk

   t_clk = $1.85 * 10^{-8}$ (aprox)

   frecuencia = 1 / t_clk = 54,4 MHz