Complemento de Arquitectura de Computadoras

Solución Examen 10 de Diciembre de 2015

(ref: scac20151210.odt)

- **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 letra. No se responderán preguntas en los últimos 30 minutos de la prueba.**
- **La prueba es individual y sin material. Duración de la prueba : 3 horas.**
- **El puntaje mínimo de aprobación es de 60 puntos.**
- **Justifique todas sus respuestas.**

Pregunta 1 (10 puntos)

En un procesador superescalar con ejecución fuera de orden, describa y comente cuál es la funcionalidad de:

1. La ventana de instrucciones.
2. La etapa Commit.

Respuesta:

Ver teórico

Pregunta 2 (10 puntos)

Explique el algoritmo de escritura a memoria write-through. ¿Es útil para solucionar el problema de coherencia de caché? ¿Por qué?

Respuesta:

Ver teórico

Pregunta 3 (12 puntos)

1. ¿De qué forma alivia a la CPU el uso de DMA para transferir datos desde E/S a memoria?
2. Describa el fenómeno de robo de ciclos de la CPU por el DMA y cómo afecta su respuesta anterior.

Respuesta:

Ver teórico

Pregunta 4 (8 puntos)

En una arquitectura con pipeline, describa la técnica de forwarding. Indique qué tipo de hazards soluciona y por qué.

Respuesta:

Ver teórico

Ejercicio 1 (30 puntos)

El algoritmo *run lenght encoding* es un algoritmo de compresión básico, el cual parte de un arreglo de caracteres y genera otro arreglo donde se indica, de forma ordenada cuántas veces aparece el caracter de forma consecutiva.

Ejemplo:

```
[A,A,A,D,D,D,C,C]       ->          [(3A),(3D),(2C)]
[A,A,A,C]                     ->          [(3A),(1C)]
[C,A,C]                       ->          [(1C),(1A),(1C)]
```

Considere el siguiente tipo de datos:

```
typedef struct codChar{
      char caracter;
      short cant;
};
```

a) Implementar en un lenguaje de alto nivel (preferentemente C), la función *run length encoding*, la cual tiene le siguiente cabezal:

```
short runLengthEncoding(char* arreglo, short lengthIn, codChar* codif);
```

El parámetro *lengthIn* indica el largo de la variable arreglo y la función devuelve el largo de la variable *codif*. Suponga que hay suficiente memoria reservada en las variables 'arreglo' y 'codif'.

b) Compilar a assembler 8086. Los parámetros *arreglo* y *codif* se pasan en SI y DI respectivamente como desplazamientos con respecto al segmento ES y el parámetro *lengthIn* se pasa en el registro CX. El resultado se devuelve en el registro AX.

Solución

b) Implementar en un lenguaje de alto nivel (preferentemente C), la función *run length encoding*, la cual tiene le siguiente cabezal:

```
short runLengthEncoding(char* arreglo, short lengthIn, codChar* codif){
      short i;
      short lengthOut = 0;
      for (i = 0; i < lengthIn; i++){
            char charActual = arreglo[i];
            short cantIguales = 0;
            while((i + cantIguales < lengthIn) &&
                        (arreglo[i + cantIguales] == charActual)){
                  cantIguales++;
            }
            codif[lengthOut].caracter = charActual;
            codif[lengthOut].cant = cantIguales;
            lengthOut++;
            i += cantIguales;
      }
      lengthOut--;  // Corrijo: size = ultimaPosicion - 1
      return lengthOut;
}
```

b)

```
      runLenghtEncoding PROC
            XOR BX, BX;             ; i = 0
            XOR BP, BP;             ; lengthOut = 0
      for:
            CMP BX, CX              ; i < lenghtIn ?
            JGE finfor
            MOV AL, ES:[SI + BX]
            XOR DX, DX              ; cantIguales
      while:
            ADD BX, DX              ; bx = i + cantIguales
            CMP BX, CX
            JGE finwhile
            CMP ES:[SI + BX], AL
            SUB BX, DX              ; bx = i
            JNE finwhile
            INC DX
            JMP while
      finwhile:
            SUB BX, DX
            ; Convierto BP (índice) en puntero
            PUSH BP                       ; guardo indice, después lo preciso
```

```
        PUSH AX
        MOV AX, BP
        ADD BP, AX
        ADD BP, AX                      ; BX = BX * 3
        POP AX
        MOV ES:[DI + BP], AL
        MOV ES:[DI + BP + 1], DX
        ADD BX, DX                      ; i += cantIguales
        POP BP                          ; recupero índice
        INC BP
finfor:
        DEC BP
        MOV AX, BP
        RET
 ENDP
```

Ejercicio 2 (30 puntos)

Se considera una máquina sin coprocesador matemático que debe emular las operaciones de punto flotante (PF) con secuencias de operaciones enteras. Su ciclo de reloj es de 10 ns. La siguiente tabla muestra las frecuencias relativas de las distintas operaciones de PF, y el número de instrucciones enteras necesarias para emular cada operación de PF.

Operación PF	**Frecuencia**	**N° de Instr. enteras p/ emularla**
Suma	50%	6
Multiplicación	40%	10
División	10%	20

Se agrega un coprocesador matemático, que elimina la necesidad de la emulación. Sin embargo, este agregado aumenta el ciclo de reloj un 20%.

Considere una carga de 150 millones de instrucciones, 20% de las cuales son enteras y 80% de PF. Asuma que el CPI=1 para las instrucciones enteras (y CPI=1 para las instrucciones de PF ejecutadas por el coprocesador).

a) Calcule los MIPS de la máquina original y de la nueva máquina con coprocesador.

b) Calcule la aceleración lograda con el agregado del coprocesador.

c) Repita el cálculo de la parte b) si se pudiera agregar el coprocesador sin afectar la duración del ciclo de reloj.

Solución

a)

Sin coprocesador:

- tenemos 20% (o sea 30 millones) son operaciones enteras que consumen 1 ciclo cada una
- tenemos 80% (o sea 120 millones) son operaciones de punto flotante, que se reparten de esta manera (0.5x6 + 0.4x10 + 0.1x20) = (3+4+2) = o sea que en promedio lleva 9 instrucciones enteras, resolver una de PF, o sea 9 ciclos
- en total CPI = 0.2+0.8x9 = 7.4
- MIPS = $1/ (1 \times 10^{6}) \times 7.4 \times (10 \times 10^{-9}) = 1/0{,}074 = 13{,}5$

Con coprocesador:

- Todas las operaciones llevan un ciclo, ya sean enteras o punto flotante
- entonces tenemos que CPI = 1
- MIPS = $1 / (1 \times 10^{6}) \times 1 \times (12 \times 10^{-9}) = 1/ 0{,}012 = 83{,}4$

b) tiempo viejo/tiempo nuevo = 7.4x10/1x12 = 74/12 = 1 / (mips viejo / mips nuevo) = 6.16

c) Siguiendo el mismo razonamiento que en la parte b), pero tomando que el ciclo de la máquina con la mejora también sea 10 ns, sería una mejora de 7.4.