Examen de Arquitectura de Computadoras

20 de julio de 2018

Instrucciones:

- Indique su nombre, apellido y número de cédula en todas las hojas que entregue.
- Escriba las hojas de un solo lado.
- Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.
- Empiece cada ejercicio en una hoja nueva.
- No puede utilizar material ni calculadora.
- **Apague su celular.**
- La duración del examen es de tres horas. En dicho tiempo debe también completar sus datos.
  Solo se contestarán dudas de letra. No se aceptan preguntas en los últimos 30 minutos del examen.
- Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.

Pregunta 1

Minimice las siguientes expresiones booleanas utilizando mapas de Karnaugh:

$f(x,y,w,z) = \Sigma(0,2,6,7,10,12,14)$

$g(x,y,w,z) = \pi(1,3,4,5,8,9,11,13,15)$

Pregunta 2

Describa el set de registros de la arquitectura 8086.

Pregunta3

Explique cómo se obtiene el número representado a partir del código binario en una representación en punto flotante IEEE 754 de simple precisión.

Pregunta 4

Suponga una CPU con pipeline. Explique qué es un hazard de control, así como las técnicas de salto demorado y predicción de saltos. Para cada una de ellas, indique si para implementarse requieren asistencia del programador/compilador o no. Justifique.

Problema 1

La empresa ***PORKEYE*** está ingresando en el mercado de “radares” de fiscalización de tránsito y ha diseñado el siguiente dispositivo para detectar velocidad excesiva y/o cruce con luz roja.

El dispositivo consta de una cámara y sensores que permiten reconocer la matrícula del auto y la distancia a la que se encuentra de la ubicación del “radar”. El cruce de calle se encuentra a 25 metros del radar y la calle que cruza tiene 5 metros de ancho. Cuando un auto sobrepasa el límite de 72 km/h o cruza con roja el sistema le toma una **única** fotografía con flash, **por cada tipo** de infracción, a los efectos de registrar la misma

Se cuenta con las siguientes interrupciones:

- ***autoDetectado()***: Se ejecuta cada vez que una nueva matrícula es detectada por la cámara. En el puerto de E/S de solo lectura de 16 bits en la dirección CHAPA queda accesible un identificador numérico relacionado a la matrícula del auto detectado.
- ***timer()***: Rutina que se ejecuta a una frecuencia de 10 Hz.

También se cuenta con los siguientes puertos:

- Puerto de E/S de 16 bits de solo escritura en la dirección LEERDISTANCIA, donde se debe escribir el identificador del auto del cual se quiere saber la distancia a la que está del radar.
- Puerto de E/S de 16 bits de solo lectura en la dirección DECIMETROS, que contiene la distancia al radar del auto cuyo identificador se haya escrito previamente en el puerto en la dirección LEERDISTANCIA (la distancia está expresada en décimas de metro). Tener presente que por la manera en que está instalado el radar los autos se alejan del mismo. El valor 0 en dicho puerto significa que el auto solicitado salió del alcance del radar.
- Puerto de E/S de 8 bits de solo lectura en la dirección SEMAFORO, que indica que luz está encendida para la calle que se controla (luz en rojo = cruce prohibido). El b7 (más significativo) indica el estado de la luz verde (1 = encendida), el b6 el de la amarilla y el b5 el de la roja.
- Puerto de E/S de 8 bits de solo escritura en la dirección FOTO. Al escribir en este puerto la cámara toma la foto con flash. En los dos bits más significativos se deben indicar la o las infracciones cometidas por el auto en cuestión (b7 en 1 = cruce con roja, b6 en 1 = velocidad excesiva).

Se pide: implementar en un lenguaje de alto nivel, preferentemente C, todas las rutinas necesarias del sistema de radar electrónico. Se dispone de un procesador dedicado para la tarea. Se puede asumir que nunca existirán más de 50 autos en el campo de detección del sistema. Se entiende que se genera una infracción por velocidad si en cualquier momento se detecta una velocidad instantánea mayor o igual a la permitida (72 km/h). Ocurre una infracción de cruce con roja si se detecta un auto en cualquier lugar de la calle que cruza mientras está la luz en rojo.

Problema 2

Considere una CPU de 16 bits con una memoria de 64KB sobre el cual se desea ejecutar el código que se presenta a continuación:

```
          MOV DX, 0x0007
          MOV AX, 0x0000
loop:     DEC DX
          JZ end
          MOV CX, 0xF8FF
          MOV SI, 0xF000
loop2:    CMP CX, SI
          JZ loop
          CMP byte ptr [SI], 0x00
          JNZ cont
          INC AX
cont:     INC SI
          JMP loop2
end:
```

Asuma que se dispone de una memoria cache para datos de tamaño 2KB y líneas de 256 bytes, cuyo *hit time* es de 5 ns, y el *miss penalty* es 7 veces mayor. Al comenzar la ejecución la memoria cache está vacía.

Se pide:

a) Compare el tiempo promedio de acceso a memoria de datos al ejecutar el programa en los siguientes casos:

1. no se utiliza memoria caché
2. se utiliza caché con una función de correspondencia directa

b) Compare el tiempo promedio de acceso a memoria de datos al ejecutar el programa en los siguientes casos:

1. se utiliza caché con una función de correspondencia totalmente asociativa, y cuando se llena se reemplaza la línea que se ha utilizado menos recientemente.
2. se utiliza caché con una función de correspondencia totalmente asociativa, y cuando se llena se reemplaza la línea que se ha utilizado más recientemente.

Nota: en todos los casos debe justificar su respuesta.

Respuesta Pregunta 1

f(x,y,w,z) = Σ(0,2,6,7,10,12,14)

xy\wz	00	01	11	10
00	1			1
01			1	1
11	1			1
10				1

f(x,y,w,z)=w!z + !x!y!z + xy!z+ !xyw

g(x,y,w,z) = π(1,3,4,5,8,9,11,13,15)

xy\wz	00	01	11	10
00	1	0	0	1
01	0	0	1	1
11	1	0	0	1
10	0	0	0	1

g(x,y,w,z)=f(x,y,w,z)=w!z + !x!y!z + xy!z+ !xyw

Respuesta Pregunta 2

La arquitectura 8086 cuenta con ZZ registros de 16 bits:
- Propósito general: AX, BX, CX, DX, SI, DI, SP, BP. De estos AX, BX, CX y DX pueden ser vistos como pares de registros de 8 bits (AH/AL, BH/BL, CH/CL, DH/DL). BX, BP, SI y DI se pueden utilizar para direccionamiento indirecto a memoria. SP junto con SS apuntan al tope de la pila (SS:SP)
- De segmento: CS, DS, ES, SS
- FLAGS: registro de banderas (carry, overflow, signo, negativo, semi acarreo) y estado del CPU (habilitacion de interrupciones, modo de depuración, etc)
- IP: puntero de instrucción

Respuesta Pregunta 3

En punto flotante IEEE 754 de simple precisión, el primer bit (S) representa el signo, los siguientes 8 bits (E) representan el exponente, y los últimos 23 bits (F), forman la mantisa.

- Si E = F = 0, el número representado es 0.
- Si E = 11111111, y F = 0, se interpreta como infinito.
- Si E = 11111111, y F != 0, se interpreta como NaN (Not a Number)
- Si E = 0, y F != 0, el número es desnormalizado, y se obtiene el valor real como:
  - $(-1)^S * 2^{-126} * (0, F)$
- En otro caso, el número está normalizado, y el valor real se obtiene como:
  - $(-1)^S * 2^{E-127} * (1, F)$

Respuesta Pregunta 4

Un hazard de control es un problema que ocurre en una CPU con pipeline, cuando al encontrarse con una instrucción de salto, se deben descartar instrucciones previamente cargadas por no ser parte del flujo lógico del programa.

Salto demorado y predicción de saltos son dos técnicas para mitigar este problema. La técnica de salto demorado consiste en que el pipeline siempre ejecute la instrucción posterior a un salto, y de este modo, se evita descartar instrucciones. Para no alterar el orden lógico del programa, el compilador (o el

programador de bajo nivel) se encarga de colocar una instrucción útil en dichas posiciones de memoria (en caso de no encontrarse una instrucción útil se puede colocar una instrucción NOP). Es decir, esta técnica sí requiere asistencia del compilador (o del programador de bajo nivel).

Predicción de saltos es una técnica en la cual se agrega hardware especializado encargado de intentar predecir si un salto condicional se tomará o no, según diferentes criterios pero usualmente en función de los resultados anteriores de ese salto (si ya se tomó), y/o de otros saltos en el programa. El compilador no interviene en esta técnica ya que se implementa en hardware

Solución Problema 1:

```
#define CHK_LUZ 0x80
#define CHK_VEL 0x40
#define CHK_LUZ_Y_VEL (CHK_LUZ|CHK_VEL)
#define FALSE 0
#define TRUE 1
#define LIMITE 20 // 20 = 72 * 1000 * 10 / 3600 * 0,1
                  // limite decímetros entre tics para velocidad máxima

struct {
      int chapa;
      int posicion;
      char ocupado;
} autos[50];

void main() {
   // instalo rutinas de interrupción
      for (int i = 0; i < 50; i++) autos[i].ocupado = FALSE;
      enable();
      while (TRUE);
}

void interrupt timer() {
  int i, distancia;

  for (i = 0; i < 50; i++) {
    if (autos[i].ocupado) {
      out(LEERDISTANCIA, autos[i].chapa);
      distancia = in(DECIMETROS);
      if ((autos[i].ocupado & CHK_VEL) && (distancia - autos[i].posicion) >= LIMITE)){
          out(FOTO, CHK_VEL);
          autos[i].ocupado &= ~CHK_VEL;
      }
      autos[i].posicion = distancia;
      if ((autos[i].ocupado & CHK_LUZ) && (autos[i].posicion >= 200) &&
             (autos[i].posicion <= 250) && (in(SEMAFORO) & 0x20 == 0x20)){
          out(FOTO, CHK_LUZ);
          autos[i].ocupado &= ~CHK_LUZ;
      }
      if (autos[i].posicion == 0) autos[i].ocupado = FALSE;
    }
  }
}

void interrupt autoDetectado() {
      // encuentro lugar libre y guardo la matrícula
      int i;

      i = 0;
      while (autos[i].ocupado && (i < 50)) i++;
      autos[i].chapa = in(CHAPA);
      autos[i].ocupado = CHK_LUZ_Y_VEL;
      autos[i].posicion = 0;
}
```

Solución Problema 2:

a)

Tiempo promedio de acceso a memoria = hit time + miss rate * miss penalty.

1. Sin memoria cache

Los accesos a memoria son 0xF8FC - 0xF000 = 0x8FF = 2304 (decimal) por iteración, y son seis iteraciones en total. Cuando no hay cache el acceso es siempre “miss penalty”, por lo que el tiempo de cada acceso es de 35ns y el total 35ns x 2304 x 6 = 483840ns.

Tanto para este caso como para las siguientes partes, la cantidad de accesos a memoria es 2304 * 6.

En este caso como no hay cache el tiempo promedio de acceso coincide con el tiempo de acceso, es decir 35ns.

2. Caché con correspondencia directa

La dirección de memoria se interpreta de la siguiente forma:
TAG: 5 bits (bloques = tamaño memoria/tamaño cache = 64KB/2KB = 32 = 2^5)
LINEA: 3 bits (2K/256B = 2^3)
BYTE: 8 bits (256 bytes por línea)

El uso de caché (acceso a datos) es:
Acceso a F0 00 – miss
(cache inicialmente vacía, se carga la línea 0 del bloque con tag 0x1E, la cache pasa a ser 1E 0 00)
[0xF000 = 11110|000|00000000 = 0x1E | 0 | 0]
F0 01 – hit (línea en caché)
F0 02 – hit
(asi sucesivamente)

F0 FF – hit
F1 00 – miss (cambia la línea 1; F100 = 11110|001|00000000 = 1E | 1 | 0, se carga línea 1)
F1 01 – hit

F1 FF – hit
F2 00 – miss (cambia la línea, se carga línea 2)
F2 01 – hit
---
y así (cada ‘bloquecito’ es de FF+1 = 256 accesos) hasta:
F7 00 – miss (se carga la línea 7)

F7 FF – hit
F8 00 – miss, F800 = 1111 1|000|0000000 (se carga la línea 0, cambia el TAG a 1F.
F8 FF – hit
segunda iteración:
F0 00 – miss (línea 0 con tag 1E no está en la caché, se carga la línea 0).
F0 01 – hit

F0 FF – hit
F1 00 – hit, la línea 1 con tag 1E está en la caché

F1 FF – hit

Hasta...
F8 00 – miss, F800 = 1111 1|000|0000000 (se carga la línea 0, cambia el TAG a 1F.

F8 FF – hit
F0 00 – miss (línea 0 con tag 1E no está en la caché, se carga la línea 0).
y así sucesivamente …

Son 9 *miss* en la primera iteración y luego dos miss por iteración (de 1 a 5), o sea un total de 19 *miss*.

Entonces tenemos que de los 2304x6 = 13824 accesos, hay 13810 hits y 19 miss. Esto da un tiempo de 13810 x 5ns + 19 x 35ns = 69050 + 665 = 69715ns

Miss rate = 19 / (2304 * 6) = 0.0013744
AMAT = 5.0481ns

b)

1. Caché con correspondencia totalmente asociativa y política de reemplazo LRU

La dirección de memoria se interpreta de la siguiente forma:
BYTE: 8 bits (256 bytes por línea)
TAG: 8 bits (identifica al bloque)

Recordar que la caché tiene únicamente 8 líneas (2KB / 256B)

El uso de caché (acceso a datos) es:
Acceso a F0 00 – miss (cache inicialmente vacía)
F0 01 – hit
F0 02 – hit

F0 FF – hit
F1 00 – miss (no hay bloque con tag F1, se carga el bloque F1)
F1 01 – hit

F1 FF – hit
F2 00 – miss (no hay bloque con tag F2, se carga el bloque F2)
F2 01 – hit
---
y así (cada 'bloquecito' es de FF+1 = 256 accesos) hasta
F7 01 – miss (y se llena la caché, en el próximo fallo hay que sustituir)
F7 02 – hit

F7 FF – hit
F8 00 – miss (hay que sustituir el bloque menos recientemente utilizado: F0, se carga F8 en lugar de F0)
F8 01 – hit
…
F8 FC – hit
segunda iteración:
F0 00 – miss (bloque con tag F0 no está en la caché)
(hay que sustituir el bloque menos recientemente utilizado: F1, se carga F0 en lugar de F1
F0 01 – hit

F0 FF – hit
F1 00 – miss (bloque con tag F1 no está en la caché)
(hay que sustituir el bloque menos recientemente utilizado: F2, se carga F1 en lugar de F2

F1 FF – hit

Y así continua hasta:
F7 00 – miss (bloque con tag F7 no está en la caché)
(hay que sustituir el bloque menos recientemente utilizado: F8, se carga F7 en lugar de F8)
F7 01 – hit
F7 FF – hit
F8 00 – miss (bloque con tag F7 no está en la caché)
(hay que sustituir el bloque menos recientemente utilizado: F0, se carga F8 en lugar de F0)

F8 FC – hit
F0 00 – miss (bloque con tag F0 no está en la caché)
(hay que sustituir el bloque menos recientemente utilizado: F1, se carga F0 en lugar de F1)

Y así hasta el final de las iteraciones externas, que son 6, entonces se tiene que sobre un total de 2304x6 accesos, hay 9 fallos en cada iteración (para cargar los bloques con cada tag), es decir un total de 54 fallos.

Entonces tenemos que de los 2304x6 = 13824 accesos, hay 13770 hits y 54 miss. Esto da un tiempo de 13770 x 5ns + 54 x 35ns = 69050 + 490 = 79740ns

Miss rate = 54 / (2304 * 6) = 0.003906

AMAT = 5.1367ns

2. Caché con correspondencia totalmente asociativa y política de reemplazo MRU

La dirección de memoria se interpreta de la siguiente forma:

BYTE: 8 bits (256 bytes por línea)

TAG: 8 bits (identifica al bloque)

El uso de caché (acceso a datos) es:

Acceso a F0 00 – miss (cache inicialmente vacía)

F0 01 – hit

F0 02 – hit

F0 FF – hit

F1 00 – miss (no hay bloque con tag F1, se carga el bloque F1)

F1 01 – hit

F1 FF – hit

F2 00 – miss (no hay bloque con tag F2, se carga el bloque F2)

F2 01 – hit

---

y así (cada ‘bloquecito’ es de FF+1 = 256 accesos) hasta

F7 01 – miss (y se llena la caché, en el próximo fallo hay que sustituir)

F7 02 – hit

F7 FF – hit

F8 00 – miss (hay que sustituir el bloque más recientemente utilizado: F7, se carga F8 en lugar de F7)

F8 01 – hit

…

F8 FF – hit

segunda iteración:

F0 00 – hit (bloque con tag F0 está en la caché)

F0 01 – hit

F0 FF – hit

F1 00 – hit (bloque con tag F1 está en la caché)

F1 FF – hit

Y así hasta el primer miss que se da en

F7 00 – miss (el bloque con tag F7 se eliminó de la caché en la sustitución, se carga F7 y se elimina el de tag F6)

F7 01 – hit

F7 FF – hit

F8 00 – hit (08 está cargada)

F8 FF – hit

F0 00 – hit (bloque con tag F0 está en la caché)

Y así hasta el final de las iteraciones externas, que son 6, entonces se tiene que sobre un total de 2304x6 accesos, hay 9 fallos en la primera iteración (para cargar los bloques con cada tag) y luego un fallo por cada iteración subsiguiente (5 fallos): total 14 fallos.

Miss rate = 14 / (2304 * 6) = 0.0010127

AMAT = 5.035445ns

Como curiosidad vemos que en este ejemplo una política de reemplazo MRU es mejor que una LRU, e incluso la totalmente asociativa con LRU es peor que una de correspondencia directa, cuando intuitivamente se tiende a pensar que sería al revés en ambos casos.