Examen de Arquitectura de Computadoras

13 de febrero de 2025

Instrucciones:

- **Indique su nombre, apellido y número de cédula en todas las hojas que entregue.**
- **Las hojas deben estar numeradas y en la primer hoja debe escribirse el total.**
- **Apague su celular. No puede utilizar material ni calculadora.**
- **La duración del examen es de tres horas. En dicho tiempo también debe completar sus datos.**
- **Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.**

Pregunta 1

Implementar una función que transforme un short a BCD. El encabezado de la función es:

```
int short2bcd(short number)
```

***Nota**: El código 1100 (C hexadecimal) indica el fin de un positivo, mientras que el 1101 (D) indica el fin de un negativo, y se ubica en la parte menos significativa del resultado.*

Pregunta 2

Enumere y describa los tipos de “hazards” que se pueden dar en una CPU con pipeline. Para uno de ellos describa una manera de mitigarlo.

Pregunta 3

Mencione tres diferencias entre CISC y RISC. Además, compare ambas filosofías de diseño en términos de cantidad de instrucciones de un mismo programa.

Pregunta 4

Minimice utilizando Karnaugh la función de 4 variables Σ(0, 1, 2, 7, 8, 9, 10, 13, 14, 15).

Problema 1

Considere la siguiente estructura de árboles binarios y la función iguales, que calcula la cantidad de nodos iguales entre dos árboles dados.

```
typedef struct {
	short valor;
	nodo* izq, der;
} nodo;

short iguales(nodo* arb1, nodo* arb2) {
	if (arb1 == NULL || arb2 == NULL)
		return 0;
	if (arb1 → valor == arb2→valor){
		return 1 + iguales(arb1 → izq, arb2→izq) + iguales(arb1→der, arb2→der);
	} else {
		return iguales(arb1→izq, arb2→izq) + iguales(arb1→der, arb2→der);
	}
}
```

a) Compile el programa en 8086 teniendo en cuenta que los parámetros y el retorno se pasan por stack y que los árboles se compilan como desplazamientos con respecto al segmento ES.

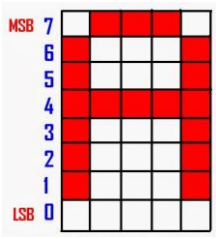
b) Calcule la cantidad máxima de nodos N1 y N2 de dos árboles arbitrarios para que el programa funcione correctamente. Justifique.

Problema 2

La empresa ***carteLED*** nos ha solicitado la programación del controlador dedicado para su nuevo letrero de anuncios basado en una matriz de LEDs monocolor de 8 filas y 128 columnas.

Los letreros basados en LEDs son un ingenioso dispositivo que utiliza la persistencia del ojo para “engañar” a nuestro cerebro, en forma similar a como lo hacen las pantallas de los televisores y monitores, a los efectos de desplegar textos o imágenes. De esta manera se van activando las columnas de a una por vez y de la columna activa se encienden los LEDs en las filas que correspondan según lo que debe estar encendido en esa columna en cada momento. Si esto se hace periodicamente para cada columna de manera secuencial a una velocidad suficientemente alta, el ojo ve los LEDs como si estuvieran encendidos en forma fija y no percibe que en realidad está parpadeando (en el caso de este letrero un LED que se ve como encendido estaría realmente encendido solo la 1/128 parte del tiempo).

En este letrero el objetivo es desplegar textos, para lo que se utilizan dibujos matriciales de 8x5 puntos de los distintos caracteres, dejando una columna apagada entre dos caracteres consecutivos.
Por ejemplo la letra A se dibuja así:



Los textos a desplegar tienen 32 caracteres (es decir tiene más caracteres que los que se pueden desplegar en la matriz de LEDs) por lo que se pretende que el texto se mueva de derecha a izquierda (de la columna 127 a la 0) a razón de una columna cada 250 ms en forma circular y cíclica.

Los textos se ingresan a través de un teclado. Cuando se hayan ingresado 32 caracteres, el nuevo texto reemplazará al anterior.

Tener en cuenta que:
- Se dispone de un ***decodificador*** de 7 bits para seleccionar la columna activa, accesible en los bits menos significativos del byte de E/S de solo escritura en la dirección **COLUMNA**.
- La activación de los LEDs se hace escribiendo en el byte de E/S de solo escritura en la dirección **LEDS** (el bit en 1 enciende el LED correspondiente).
- El teclado genera una interrupción por cada tecla presionada que invoca a la rutina de interrupción **teclado()**, quedando disponible el código ASCII de la tecla en el byte de E/S de solo lectura en la dirección **CARACTER**.
- Se dispone de un ***timer*** que interrumpe con una frecuencia de 400 Khz, invocando a la rutina **tiempo()**.

Se pide:

Escribir en un lenguaje de alto nivel, preferentemente C, todas las rutinas para controlar el funcionamiento del display giratorio, teniendo en cuenta que:

Se dispone de un arreglo bidimensional de bytes de nombre ***matriz8x5[ascii][columna]*** precargado con los dibujos de los caracteres ASCII en matrices de 8x5 puntos. Cada elemento del arreglo es una columna de la matriz de puntos.

Solución

Pregunta 1

```
int short2BCD(short num){
  int abs = num; // se usa int para no tener problemas de módulo cuando num vale 1000…0000
  int bcd = 0xC;
  if(abs < 0){
    abs = -abs; // Convertimos el numero positivo para procesarlo
    bcd++; // fin BCD negativo
  }
  for(short shift = 4; abs > 0; shift += 4){
    bcd += ((abs % 10)<<shift); // Colocamos el ultimo digito decimal de num en el resultado
    abs /= 10;  // Eliminamos el ultimo digito
  }
  return bcd;
}
```

Pregunta 2

Hay posibles problemas y situaciones que generan problemas para el funcionamiento óptimo del pipeline. En general estas situaciones se denomina en inglés *hazards*. Estos se clasifican en tres grandes grupos:

- Hazards estructurales: hay un conflicto de hardware entre dos o más etapas del pipeline para alguna combinación de instrucciones. Una solución general al problema de la falta de un recurso de hardware es esperar. No es una solución óptima pero es la más sencilla, en el sentido que no agrega hardware. Una forma es agregar, artificialmente, operaciones del tipo “NOP”, que no requieren de recursos. Está técnica también se denomina burbuja de pipeline. Es habitual encontrar una combinación de soluciones: se agrega hardware para los casos más comunes (ej: sumador en la etapa de Fetch) y se implementan esperas para los casos menos comunes.
- Hazards de datos: existen dependencias de datos entre instrucciones. En este caos, el problema aparece porque hay datos en común entre instrucciones que están en el pipeline, ya sea una instrucción siguiente que utiliza un resultado aún no calculado como operando ó altera un operando antes que sea leído como operando. Una forma de mitigarlo es hacer register forwarding. Esta técnica consiste en propagar hacia las etapas tempranas del pipeline los resultados apenas estén disponibles, sin necesidad de esperar a que culmine la etapa de Write. En este caso la etapa de Read podrá leer el operando desde el registro indicado ó desde la salida de la ALU directamente, en función de la lógica de control que implementa esta funcionalidad.
- Hazards de control: causados por instrucciones de salto u otras modificaciones del registro IP. La etapa de Fetch asume, por defecto, que la próxima instrucción a ejecutarse es la que está a continuación en memoria, lo que nunca es correcto si la instrucción es un salto incondicional ó no es siempre correcto si es un salto condicional. Una técnica de mitigación es el prefetch de destino, que consiste en realizar simultáneamente el fetch de la próxima instrucción en memoria y de la instrucción apuntada por la dirección del destino del salto. Cuando el salto se efectiviza la instrucción que no corresponde a la secuencia lógica se descarta.

Pregunta 3

Con respecto al set de instrucciones, las arquitecturas CISC suelen tener un conjunto mucho más amplio de instrucciones que las arquitecturas RISC (que como su sigla lo sugiere tienen un conjunto reducido de instrucciones). A nivel de formato de instrucción, las arquitecturas CISC suelen implementar instrucciones de largo variable, mientras que las arquitecturas RISC utilizan una codificación de largo fijo. Por último, una tercer diferencia se puede encontrar en el set de registros, donde las arquitecturas CISC muchas veces ofrecen registros con propósitos específicos, mientras que las RISC disponen de registros uniformes, 'sin personalidad' y, muchas veces, un número mayor.

En términos de la cantidad de instrucciones en un mismo programa, una tarea en CISC generalmente requiere menos instrucciones, ya que estas son más complejas y pueden realizar múltiples operaciones a la vez. En cambio, en RISC, las instrucciones son más simples, por lo que se suelen necesitar más instrucciones para el mismo programa. Un ejemplo de esto sería una arquitectura CISC que incluye una instrucción de multiplicación, mientras

que en una arquitectura RISC, esta operación podría no estar disponible y requerir múltiples instrucciones más simples para obtener el mismo resultado.

Pregunta 4

Primero realizamos la tabla de verdad:

a	b	c	d	**f**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Y luego el Karnaugh correspondiente:

cd \ ab	00	01	11	10
00	1			1
01	1		1	1
11		1	1	
10	1		1	1

Resultando en la expresión minimizada:

f (a,b,c,d) = b.d + b.c + b.c.d + a.b.d + a.b.c

Problema 1

a)

```
iguales proc
  push BP
  mov BP, SP
  push SI
  push DI
  push AX
  xor AX, AX ; AX =resultado
  mov SI, [BP+6]  ; SI=arbol1
  cmp SI, 0
  je fin
  mov DI, [BP+4]  ; DI=arbol2
  cmp DI, 0
  je fin
  mov AX, ES:[SI] ; accedo al dato de los arboles
  cmp AX, ES:[DI]
  mov AX, 0
  jne else
  inc AX
else
  push ES:[SI+2]  ; coloco parametros para llamada recursiva con hijos izquierdos
  push ES:[DI+2]
  call iguales
  pop BX
  add AX, BX
  push ES:[SI+4]  ; idem hijos derechos
  push ES:[DI+4]
  call iguales
  pop BX
  add AX, BX
fin:
  mov [BP+6], AX  ; guardo resultado
  mov AX, [BP+2]
  mob [BP+4], AX  ; acomodo dir ret
  pop AX          ; recupero contexto local
  pop DI
  pop SI
  pop BP
  add SP, 2       ; ajusto stack para retorno
  ret
iguales endp
```

b)

Primero calculemos la cantidad máxima de nodos que se pueden almacenar en un segmento. El segmento ES tiene 64Kb y cada nodo consume 6 bytes (2 para el dato y 4 para los punteros). Considerando que un nodo no puede comenzar en el desplazamiento 0 (NULL) tenemos que la cantidad máxima de nodos es 65535/6, es decir 10922 nodos.

Calculemos ahora el consumo de stack del algoritmo.

Para el paso base tenemos que se consumen 4 bytes para los parámetros y 2 para la dirección de retorno. El contexto preservado son 8 bytes (BP, SI, DI y AX). Por lo tanto el consumo de stack del paso base son 14 bytes.

Para los pasos recursivos tanto para el caso en que el valor es igual como distinto se tienen también 4 bytes de parámetros, 2 de dirección de retorno y 8 bytes de contexto (BP, SI, DI y AX).

Para cada rama se realizan 2 llamadas recursivas pero no se almacena contexto adicional entre ambas.

Por lo tanto el consumo del paso recursivo también es de 14 bytes.

Ahora se debe tener en cuenta la estructura de los árboles para determinar el peor caso de consumo de stack. El mismo se va a dar cuando los árboles coincidan en estructura en profundidad por una única rama.

Como el segmento de stack tiene 64KB se tiene que se pueden realizar 65536/14 invocaciones anidadas, es decir 4681 invocaciones. Por lo tanto el mayor consumo de stack es 65534 bytes.

Por lo tanto la cantidad máxima de nodos N1 y N2 para que el algoritmo funcione en cualquier caso debe ser de N1=4680 nodos y N2=10922-4680 nodos (N2=6242) o simétricamente N2=4680 y N1=6241.

Problema 2

```
#define TICS_CAMBIO 100000
#define MAX_CHARS 32
#define MAX_COLS 192
#define MAX_COLS_DISP 128

int pos_msg = 0;
char n_msg[MAX_CHARS], msg[MAX_COLS];
int tics_cambio=0;
int col=0;

void main(){
      // instalar interrupciones
      for(int i = 0; i < MAX_COLS; i++)
        msg[pos]=0;
      enable();
      while(1){}
}

void interrupt teclado(){
      n_msg[pos_msg++] = in(CARACTER);
      if(pos_msg == MAX_CHARS){
            int pos = 0;
            for(int i = 0; i < MAX_CHARS; i++){
                  char car = n_msg[i];
                  for(int j = 0; j < 5; j++)
                        msg[pos++] = matriz8x5[car][j];
                  msg[pos++] = 0; // columna vacia
            }
            pos_msg = 0;
            tics_cambio = 0;
            col_inicio = 0;
            col = 0;
      }
}
```

```
void interrupt tiempo(){
      tics_cambio++;
      if(tics_cambio == TICS_CAMBIO){
        col_inicio++;
        col_inicio %= MAX_COLS;
        tics_cambio = 0;
      }
      col++;
      col = col % MAX_COLS_DISP;
      out(COLUMNA, col);
      out(LEDS, msg[(col_inicio + col) % MAX_COLS]);
}
```