Examen de Arquitectura de Computadoras

20 de diciembre 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

Se desea construir un multiplexor de N entradas de control utilizando únicamente una memoria ROM.

a. Indique tamaño y organización de la ROM requerida, indicando el significado de entradas y salidas

b. Escriba la tabla de verdad de dicha ROM para N=1

Pregunta 2

a. Presente la tabla de verdad de un Flip-Flop (FF) tipo JK y de un FF tipo T.

b. Construya un FF tipo T a partir de un FF tipo JK.

Pregunta3

Considere el siguiente sistema de codificación de números punto fijo:

La parte entera del número se codifica en desplazamiento de 8 bits.
La parte fraccionaria se codifica en binario natural de 4 bits.

Codifique el siguiente número: -45,128

Pregunta 4

En el contexto de una CPU con pipeline, explique qué es un hazard de datos. Explique dos posibles técnicas para resolverlos.

Problema 1

La empresa ***Neversleep*** se encuentra desarrollando un sistema de control de una máquina expendedora de café. La máquina cuenta con un microprocesador dedicado y dispone de 2 tipos de café: café extra fuerte y capuccino.

El sistema consta de los siguientes elementos:

- Un detector de fichas que genera un pedido de interrupción cada vez que ingresa una ficha a la máquina. La rutina que se invoca para atender la interrupción se llama **ficha()**.

- Una puerta de devolución de fichas, que devuelve una ficha al escribir un 1 en el bit menos significativo del puerto de E/S de un byte de sólo escritura en la dirección **DEVOLVER**.
- Un par de botones que al presionarse generan una interrupción invocando a la rutina **boton()**. El botón presionado queda accesible codificado en los dos bits menos significativos del puerto de E/S de solo lectura en la dirección **ID_BOTON** (el contenido del puerto se pierde una vez leído), según la siguiente tabla:

ID Botón	Tipo bebida
1	café extra fuerte
2	capuccino

- Una válvula dispensadora de café y otra de leche. Para controlar las válvulas se utiliza el puerto de E/S (byte de lectura y escritura) en la dirección **DISPENSADOR**. El bit 1 controla la válvula dispensadora de café y el bit 2 la válvula de la leche (1=abierta y 0=cerrada).
- Un timer que interrumpe con una frecuencia de 10 Hz invocando a la rutina **tiempo()**.
- Un sensor que permite detectar la presencia de un vaso bajo las válvulas dispensadoras. Este sensor es accesible en el bit 2 del puerto de E/S (byte de solo lectura) en la dirección **SENSOR** (1=vaso detectado, 0=vaso no detectado).

El sistema debe funcionar de la siguiente manera:

En usuario, luego de ingresar una ficha, puede seleccionar presionando un botón el tipo de bebida que desea consumir.

La máquina debe aceptar una sola ficha por pedido (si se ingresan más, deberán ser devueltas). Si durante 5 minutos sin seleccionar bebida deberá cancelar la operación y devolver la ficha.

La preparación comienza luego de que el usuario selecciona el tipo de bebida (siempre que antes haya sido ingresada una ficha) y siempre y cuando se detecte la presencia de un vaso.

La preparación debe realizarse de acuerdo al tipo de bebida:

- Si es un café extra fuerte, se debe activar la válvula del café durante 4 segundos.
- Si es capuccino, se debe activar la válvula del café durante 2 segundos y luego la válvula de la leche durante 2 segundos.

Una vez iniciado el proceso de preparación, éste no puede ser detenido, por lo que si se introduce una ficha debe ser devuelta y si se presiona un botón se debe ignorar.

Se pide:

Escribir en un lenguaje de alto nivel (preferentemente C) todas las rutinas necesarias para implementar el sistema de control descripto.

Nota: se asume que el usuario coloca el vaso y lo retira luego de finalizada la preparación.

Problema 2

Considere el siguiente código que implementa el algoritmo ***Quicksort*** para ordenar secuencias de enteros de 8 bits:

```
int partition(char* vector, int ini, int fin){
    int i = ini;
    char pivot = vector[fin];
    for (int j = ini; j < fin; j++){
            if (vector[j] < pivot){
                    if (i != j){
                            char temp = vector[i];
                            vector[i] = vector[j];
                            vector[j] = temp;
                    }
                    i++;
            }
    }
    char temp = vector[i];
    vector[i] = vector[fin];
    vector[fin] = temp;
    return i;
}

void quicksort(char* vector, int ini, int fin){
    if (ini < fin){
            int pivot = partition(vector, ini, fin);
            quicksort(vector, ini, pivot - 1);
            quicksort(vector, pivot + 1, fin);
    }
}
```

Se pide:

a) Compilar las funciones **partition** y **quicksort** en assembler 8086. Considere que, para ambas funciones, los punteros se representan como desplazamientos dentro del segmento ES, que los parámetros se reciben por stack y que el resultado se retorna en el stack. Es necesario preservar el valor de los registros.

b) Calcular el máximo consumo de stack cuando se invoca a **quicksort** para ordenar el arreglo:

char vector_prueba[3] = {1, 2, 3};

Respuesta Pregunta 1

a) Un multiplexor consta de 2^N entradas de datos, N entradas de control y 1 salida. Si numeramos las 2^N entradas de datos con N bits, las N entradas de control vistas como un número binario indican el índice de la entrada que debe presentarse en la salida del multiplexor.
Por lo tanto para construir un multiplexor de N entradas de control requerimos un ROM 2^(2^N+N)x1 con un tamaño de 2^(2^N+N) bits.
Los N bits más significativos de la dirección serán las entradas de control y los 2^N bits menos significativos entrada serán las entradas de datos. La salida será el valor de la entrada de datos asociada a cada valor de la entrada de control.

b) Para N=1 la tabla queda (x es la entrada de control y a y b las entradas de datos, siendo b la entrada 0 y a la entrada 1):

dirección(xab)	contenido
000	0
001	1
010	0
011	1
100	0
101	0
110	1
111	1

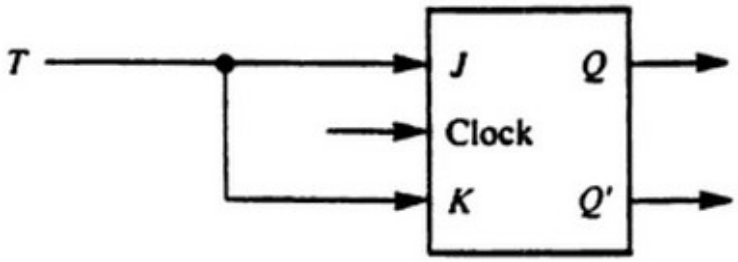
Respuesta Pregunta 2

a.

T	Q_{n+1}
0	Qn
1	Q'n

J	K	Q_{n+1}
0	0	Qn
0	1	0
1	0	1
1	1	Q'n

b.



Respuesta Pregunta 3

D = 2 ^ 7 = 128
-45 se codifica como - 45 + 128 = 83 en binario natural.

-45 + D = 01010011b

0,128 * 2 = 0,256
0,256 * 2 = 0,512
0,512 * 2 = 1,024
0.024 * 2 = 0,048

Por lo tanto -45,128 se codifica como 01010011 0010

Respuesta Pregunta 4

Un hazard u obstáculo es un problema o una situación que impide alcanzar el rendimiento óptimo de un pipeline. Los hazards de datos se producen cuando diferentes instrucciones que están en el pipeline utilizan datos en común, de forma que interfieren entre ellas. Un ejemplo de esto es el hazard RAW, que ocurre cuando una instrucción del pipeline requiere un cierto operando para una instrucción, pero el valor de dicho operando es calculado por una instrucción anterior, y esta aún no ha escrito el resultado en registros o memoria. En el curso se mencionan soluciones para este tipo de hazards, entre ellas: i) detectar el hazard a través de lógica combinatoria en el pipeline, y generar una señal de detención (stall), introduciendo una 'burbuja' en el pipeline; ii) aplicar la técnica de register forwarding, que consiste en propagar hacia las etapas tempranas del pipeline los resultados apenas estén disponibles, sin necesidad de esperar a que culmine la etapa de escritura, para que la etapa de lectura pueda leer el operando desde el registro indicado ó desde la salida de la ALU directamente, en función de la lógica de control que implemente esta funcionalidad.

Solución Problema 1:

```
#define DOS_SEG 20
#define CUATRO_SEG 40
#define CINCO_MIN 10*60*5
#define ESPERANDO_FICHA 0
#define ESPERANDO_BOTON 1
#define CAFE 2
#define CAPUCCINO_CAFE 3
#define CAPUCCINO_LECHE 4
#define FALSE 0
#define TRUE 1

int tics, estado, hayFicha, hayVaso, preparando;


void main(){
	// Instalar interrupciones
	tics = 0;
	hayFicha = FALSE;
	hayVaso = FALSE;
	preparando = FALSE;
	estado = ESPERANDO_FICHA;
	// Habilitar interrupciones

	while (TRUE){

		if ( (IN(SENSOR) & 0x04) && !preparando ){
			if ((estado == CAFE || estado == CAPUCCINO_CAFE)){
				OUT(DISPENSADOR, IN(DISPENSADOR) | 0x02); //Abro válvula del
café
			}
			else if (estado == CAPUCCINO_LECHE){
				OUT(DISPENSADOR, IN(DISPENSADOR) | 0x04); //Abro válvula de la
leche
			}
			tics = 0;
			preparando = TRUE;
		}
	}
}


	void interrupt timer(){

		tics++;
		if ((estado == ESPERANDO_BOTON) && (tics >= CINCO_MIN)){
			OUT(DEVOLVER, 0x01);
			estado = ESPERANDO_FICHA;
			hayFicha = FALSE;
			preparando = FALSE;
		}


		if ((estado == CAFE) && (tics >= CUATRO_SEG)){
			OUT(DISPENSADOR, IN(DISPENSADOR) & 0xFD); //Cierro válvula del café
			estado = ESPERANDO_FICHA;
			hayFicha = FALSE;
			preparando = FALSE;
		}
```

```
            else if ((estado == CAPUCCINO_CAFE) && (tics >= DOS_SEG)){

                  OUT(DISPENSADOR, IN(DISPENSADOR) & 0xFD); //Cierro válvula del cafe
                  estado = CAPUCCINO_LECHE;
                  preparando = FALSE;
            }
            else if ((estado == CAPUCCINO_LECHE) && (tics >= DOS_SEG)){
                  OUT(DISPENSADOR, IN(DISPENSADOR) & 0xFB); //Cierro válvula de la
leche
                  estado = ESPERANDO_FICHA;
                  hayFicha = FALSE;
                  preparando = FALSE;
            }

      }


      void interrupt ficha(){

            if (hayFicha){
                  OUT(DEVOLVER, 0x01);
            }
            else{
                  hayFicha = TRUE;
                  estado = ESPERANDO_BOTON;
                  tics = 0;
            }
      }


      void interrupt boton(){

            if (estado == ESPERANDO_BOTON){

                  if (IN(ID_BOTON) & 0x01){
                        estado = CAFE;
                  }
                  else {
                        estado = CAPUCCINO_CAFE;
                  }
            }
      }
```

Solución Problema 2:

a)

```
quicksort proc
        push BP                 ;  respaldar registros y ajustar BP para acceder a al stack
        mov BP, SP
        push AX
        push BX
        push CX

        mov AX, [BP+6]      ;  cargo ini en AX
        mov BX, [BP+4]      ; cargo fin en BX
        cmp AX, BX
        jl fin
        push [BP+8]     ;  parametro vector
        push AX                 ; ini
        push BX                 ; fin
        call partition
        pop CX                  ; obtengo pivot del stack
        dec CX                  ; pivot - 1
        push [BP+8]     ; parametro vector
        push AX                 ; ini
        push CX                 ; pivot - 1
        call quicksort
        add CX, 2
        push [BP+8]     ; parametro vector
        push CX                 ; pivot + 1
        push BX                 ; fin
        call quicksort
fin:
        mov AX, [BP + 2]
        mov [BP+8],  AX
        pop AX
        pop BX
        pop CX
        pop BP
        add SP, 6
        ret
endp

partition proc
        push BP                 ;  respaldar registros y ajustar BP para acceder a al stack
        mov BP, SP
        push AX
        push BX
        push SI
        push CX
        push DI
        push DX

        mov DI, [BP+6]      ; cargo ini en DI (i)
        mov BX, [BP+8]      ; cargo vector en BX
        mov SI, [BP+4]      ; cargo fin en SI
        mov CL, ES:[BX+SI] ; pivot = vector[fin]
```

```
        mov SI, DI              ; j = ini
loop:
        cmp SI, [BP+4]          ; j < fin (control del for loop)
        jge fin
        cmp ES:[BX+SI], CL ; vector[j] < pivot
        jge finloop_1
        cmp DI, SI              ; i != j
        je finloop_2
        mov DL, ES:[BX+DI] ; temp = vector[i]
        mov AL, ES:[BX+SI] ; temp_2 = vector[j]
        mov ES:[BX+DI], AL ; vector[j] = temp
        mov ES:[BX+SI], DL ; vector[i] = temp_2
finloop_2:
        inc DI                  ; i++
finloop_1:
        inc SI                  ; j++
        jmp loop
fin:
        mov DL, ES:[BX+DI] ; temp = vector[i]
        mov SI, [BP+4]          ; cargo fin en SI
        mov AL, ES:[BX+SI] ; temp_2 = vector[fin]
        mov ES:[BX+DI], AL ;; vector[fin] = temp
        mov ES:[BX+SI], DL ; vector[i] = temp_2

        mov [BP+8], DI          ; coloco el parámetro i que voy a devolver
        mov DI, [BP+2]
        mov [BP+6], DI          ; acomodo IP
        pop DX
        pop DI
        pop CX
        pop SI
        pop BX
        pop AX
        pop BP
        add SP, 4
        ret
endp
```

b)

char vector_prueba[3] = {1, 2, 3};

La función se invoca con quicksort(&vector_prueba, 0, 2)

Se invoca a la función partition una vez, la cual no realiza intercambios por estar el vector ordenado y devuelve 2 como pivot. Las siguientes llamadas recursivas son quicksort(vector_prueba, 0, 1) y quicksort(vector_prueba, 3, 2). La segunda llamada culmina en el paso base, mientras que la primera llama a partition, quien nuevamente no realiza cambios y devuelve pivot = 1. Las últimas dos llamadas recursivas entonces son quicksort(vector_prueba, 0, 0) y quicksort(vector_prueba, 2, 1), las cuales son ambos pasos base.

El árbol de recursiones queda de la siguiente forma:

quicksort – partition

- **quicksort – partition – peor caso (ver más adelante)**
  - quicksort (base)
  - quicksort (base)
- quicksort (base)

Quicksort consume 2 bytes por IP, 6 bytes por parámetros y 8 bytes por contexto, tanto en el paso base como en el recursivo. A su vez, partition consume 2 bytes por IP, 6 bytes por parámetros y 14 bytes por contexto.

Por esta razón, el peor caso es la secuencia marcada en negrita en el árbol de recursión brindado, y el consumo de stack es: 2 * consumo_quicksort + consumo_partition = 2 * 16 + 22 = 54bytes.