Examen – 23 de Diciembre de 2013

Instrucciones

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.**
- **Escriba las hojas de un solo lado.**
- **Empiece cada ejercicio en una hoja nueva.**
- **Sólo se contestarán dudas de letra. No se aceptarán dudas los últimos 30 minutos del examen.**
- **Duración: 3 horas.**
- **Requisito para Aprobación: responder al menos la mitad del problema teórico (2 de las 4 preguntas) y resolver un problema de práctico de forma correcta**

Pregunta 1

Describa el algoritmo de resta de números racionales codificados según el estándar de punto flotante IEEE 754.

Respuesta:

Para realizar la resta primeramente se deben alinear los exponentes, llevando el de exponente mas chico al exponente mayor. Para esto se debe desplazar la mantisa del de menor exponente hacia la derecha tantos bits como sea la diferencia de exponentes.
Luego se determina el signo del resultado en función de los signos del minuendo y el sustraendo y de cuál de ellos resultó ser el mayor (en valor absoluto). Por ej: si ambos son positivos y el minuendo es mayor entonces la diferencia es positiva. Si ambos son negativos y se da la misma relación la diferencia es negativa. Si el minuendo es positivo y el sustraendo negativo la diferencia es positiva. Si es al revés es negativa.
Luego se hace la resta o la suma (ambas en binario) de las mantisas, tomando en cuenta el 1 implícito, según que los signos sean el mismo o distintos. Se ajusta el exponente de ser necesario para normalizar el número y finalmente se arma el código binario del resultado con el signo que se determinó y el exponente y la mantisa resultantes.

Pregunta 2

Explique el problema que aparece en sistemas multiprocesador con memoria principal compartida y memorias caches individuales para cada procesador, utilizando una política de escritura Write Back. ¿Persiste el problema si se utiliza Write Through?. ¿Por qué?.

Respuesta:

En un sistema multiprocesador con memoria principal compartida y memorias caches individuales para cada procesador y Write Back ocurre el siguiente problema: si el procesador A realiza una escritura sobre una cierta dirección que está en ese momento en su cache, se actualiza su cache con el valor correspondiente, pero si luego el procesador B realiza una lectura sobre dicha dirección, entonces tanto en caso de Hit o de Miss en su propia cache, se obtendrá un valor erróneo del dato. Este problema es otro caso del problema de la “coherencia del cache” que aparece en sistemas que utilizan DMA (Direct Memory Access), en este caso propio de los sistemas multiprocesador.

Si se utiliza Write Through también ocurre el problema, puesto que si B ya tenía en cache el dato correspondiente a la dirección escrita, por más que esta se actualice en memoria, si B vuelve a acceder a la dirección, dará Hit en su cache, devolviendo un dato incorrecto. La diferencia es que en este caso el problema es igual que el provocado por el DMA.

Pregunta 3

Explique la diferencia entre Arquitectura y Organización de una computadora. Describa tres elementos que permitan diferenciar dos arquitecturas Von Neumann diferentes.

Respuesta:

La Arquitectura de una Computadora, es el conjunto de recursos visibles para el programador (por ejemplo set de instrucciones, set de registros, mecanismo de interrupciones, etc). La Organización, por otro lado, refiere a la forma en que se implementa en hardware una Arquitectura.

Como fue mencionado arriba, tres elementos que diferencian dos arquitecturas Von Neumann entre sí, son: el Set de Instrucciones, el Set de Registros, y el mecanismo de interrupciones utilizado.

Pregunta 4

Explique el propósito de utilizar un predictor de saltos en un procesador con Pipeline. ¿Qué es una Tabla de Historia de Saltos (BHT)?. ¿Cómo funciona?.

Respuesta:

Un predictor de saltos se utiliza como medida para mitigar el impacto de las dependencias de control. Esto se logra, al momento de tomar un salto, tratando de adivinar cuál será el resultado del salto, logrando entonces, si la predicción es correcta, un menor número de vaciados del pipeline.

Una Tabla de Historia de Saltos (Branch History Table – BHT), es una estructura de datos mantenida en el propio procesador, donde se guardan los resultados de saltos para una serie de saltos recientemente evaluados. Al tomar un salto, se accede la tabla según la dirección del salto y de ahí se obtiene la predicción a aplicar. Luego se actualiza la entrada de la tabla según el resultado efectivo del salto.

Problema 1

Un módulo de Laser Range Finder (LRF) es un dispositivo que utiliza un haz láser para determinar la distancia a un objeto. Muchos robots utilizan este tipo de módulos para implementar algoritmos de evasión de obstáculos y la construcción de mapas.

El módulo LRF se coloca en el eje de un motor, de manera de poder apuntarlo en distintas direcciones al modificar la posición angular del motor. El motor puede ser colocado en 16 posiciones angulares predefinidas.

Las interfaces del LRF y del motor son las siguientes:

- Para generar un pulso láser se debe escribir un 1 en el bit más significativo del byte de E/S de solo escritura en la dirección LRF.
- El módulo LRF genera una interrupción al procesador atendida por la rutina lrfFinish() cuando finalizó de calcular la distancia, dejando este resultado en la palabra de E/S de solo lectura en la dirección DISTANCE.
- Para posicionar el motor en la posición *i* debe escribirse un 1 en el bit *i* de la palabra de E/S de solo escritura en la dirección MOTOR_POSITION.
- El motor coloca un 1 en el bit más significativo de la palabra de E/S solo lectura MOTOR_STATE cuando el motor alcanzó la posición indicada.

Se pide:

a) Implementar en un lenguaje de alto nivel, preferentemente C, la función getDistances() y todas las demás rutinas necesarias para el correcto funcionamiento de la misma. La función recibe como parámetro una palabra con las posiciones angulares cuyas distancias correspondientes se desean obtener y devuelve un array con las lecturas en las posiciones indicadas. Debe retornar -1 para las posiciones no consultadas.

b) Compilar a assembler 8086 la solución propuesta en la parte anterior.

Observaciones:

- Puede asumirse que el módulo LRF siempre finaliza una lectura encomendada.
- Cada bit i de la palabra de posiciones se corresponde con la posición i del motor.

Solución:

```
# define MAX_POS 16
# define MOTOR_FINISH 2**15
# define LRF_START 2**7

bool readOK;

void interrupt lrfFinish() {
     readOK = true;
}

int [] getDistances(int readPositions)  {
  int [] result = new int[MAX_POS];

  for (int iterPos=0; iterPos < MAX_POS; iterPos++) {
    if ((readPositions & 2 ** iterPos)) {
      out(MOTOR_POSITION,  2 ** iterPos); // mando posicionar el motor
```

```
      while(!(in(MOTOR_STATE)& MOTOR_FINISH));//espero a que el motor alcance
la posicion
      readOK = false;
      out(LRF,  LRF_START); // genero el pulso laser
      while (!readOK); // espero a que finalice la lectura
      result[iterPos] = in(DISTANCE);
    } else
      result[iterPos] = -1;
  } // fin for
return result;
} // fin getDistances

b)
true equ 1
false equ 0

readOK db 0

proc far lrfFinish
  mov [CS:readOK],  true
  iter
endp

proc getDistances
  push BP
  mov BP, SP
  push AX
  push BX
  push CX
  push DX
  push SI
  push DI

  mov DI, [BP+4]         ; saco del stack las posiciones a leer
  mov BX, offset result  ; desplazamiento del resultado

  xor SI, SI ; uso SI para el iterador
  mov CX, 1 ; uso CX como mascara de posicion actua

for:
  cmp SI, MAX_POS
  je finFor
  mov DX, DI
  and DX, CX
  je else
  mov DX,  MOTOR_POSITION
  mov AX, CX
  out DX, AX
  mov DX,  MOTOR_STATE
waitMotor:
  in AX, DX
  and AX, MOTOR_FINISH
  je waitMotor
  mov [CS:readOK],  false
  mov DX, LRF
```

```
  mov AX, LRF_START
  out DX, AX
waitLRF:
  cmp [CS:readOK],  true
  jne waitLRF
  mov DX,  DISTANCE
  in AX, DX
  mov [CS:BX], AX
  jmp finIf
else:
  mov word ptr [CS:BX], -1
finIf:
  add BX, 2 ; me muevo una palabra en el array de resultados
  sll CX, 1 ; muevo la mascara al siguiente bit
  inc SI ; incremento el iterador
  jmp for
finFor:
  mov [BP+4], offset result ; retorno el desplazamiento de resultado
  pop DI
  pop SI
  pop DX
  pop CX
  pop BX
  pop AX
  pop BP
  ret
endp
```

Problema 2

En el restaurante **Le Bascule** se usa un pequeño montacarga para enviar los platos preparados desde la cocina al salón comedor ubicados en pisos distintos del edificio, y retornar los platos vacíos, al retirar el servicio, hacia la cocina. La cocina se encuentra en el nivel inferior y el salón comedor en el nivel superior.

El montacarga tiene un botón en cada nivel que lo hace subir o bajar dependiendo de dónde esté actualmente la plataforma.. Por ejemplo si el cocinero desea enviar un plato hacia el salón comedor y la plataforma del montacarga está en ese momento en el nivel del salón comedor, primero debe pulsar el botón, esperar que llegue la plataforma hasta su nivel, colocar el plato y luego pulsar de nuevo el botón para que suba.

Los botones de ambos niveles están cableados en paralelo y su estado disponible en la señal ***botón*** (“1” si el botón está apretado).

El montacarga tiene un sensor de presencia de la plataforma en cada nivel. Si la plataforma se encuentra en un nivel el sensor correspondiente se activará y su señal asociada pasará al valor “1”. El sensor del nivel cocina se refleja en la señal ***cocina***, y el del salón comedor en la señal ***comedor***.

El motor del montacarga se enciende poniendo a “1” la señal ***motor***, mientras que el sentido de desplazamiento se establece en la señal ***sentido*** (“1” es subir, “0” es bajar).

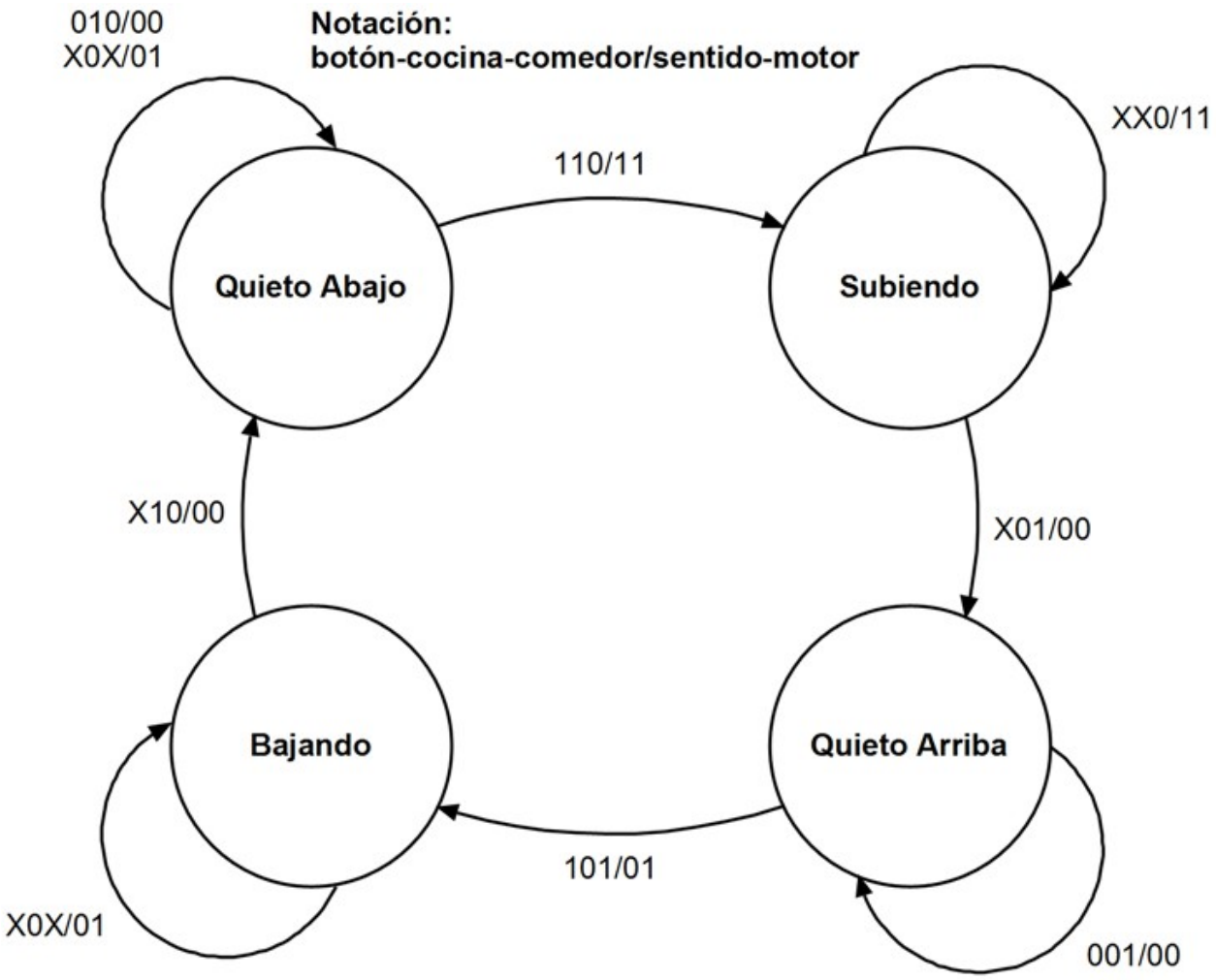
Se pide:

Diseñar, siguiendo la metodología del curso, y dibujar un circuito secuencial que controle las señales ***motor*** y ***sentido*** a partir de los valores de las señales ***botón***, ***cocina*** y ***comedor***. Se dispone de flip-flops tipo D y compuertas básicas.

Solución:

Se identifican cuatro estados diferentes: quieto arriba, quieto abajo, subiendo y bajando. Muchas combinaciones de entradas son imposibles una vez que la máquina esté funcionando, por ejemplo que la plataforma se encuentra en el estado 'Quieto Arriba' y que **cocina = 1**. Sin embargo, como luego de un reset se debe ir a uno de esos estados obligatoriamente (logrando llegar a dicho estado en cualquier posición de la plataforma), se elige el estado 'Quieto Abajo' como estado de reset y se fuerza a bajar la plataforma en caso de llegar por un reset.

El diagrama de estados que resuelve el problema entonces es el siguiente:



La tabla de estados es la siguiente:

Estado Actual	Entrada			Prox. Estado	Salida	
	Botón	Cocina	Contador		Sentido	Motor
Quieto Abajo	0	0	0	Quieto Abajo	0	0
Quieto Abajo	0	0	1	Quieto Abajo	0	1

Quieto Abajo	0	1	0	Quieto Abajo	0	0
Quieto Abajo	0	1	1	X	X	X
Quieto Abajo	1	0	0	Quieto Abajo	0	1
Quieto Abajo	1	0	1	Quieto Abajo	0	1
Quieto Abajo	1	1	0	Subiendo	1	1
Quieto Abajo	1	1	1	X	X	X
Subiendo	0	0	0	Subiendo	1	1
Subiendo	0	0	1	Quieto Arriba	0	0
Subiendo	0	1	0	Subiendo	1	1
Subiendo	0	1	1	X	X	X
Subiendo	1	0	0	Subiendo	1	1
Subiendo	1	0	1	Quieto Arriba	0	0
Subiendo	1	1	0	Subiendo	1	1
Subiendo	1	1	1	X	X	X
Quieto Arriba	0	0	0	X	X	X
Quieto Arriba	0	0	1	Quieto Arriba	0	0
Quieto Arriba	0	1	0	X	X	X
Quieto Arriba	0	1	1	X	X	X
Quieto Arriba	1	0	0	X	X	X
Quieto Arriba	1	0	1	Bajando	0	1
Quieto Arriba	1	1	0	X	X	X
Quieto Arriba	1	1	1	X	X	X
Bajando	0	0	0	Bajando	0	1
Bajando	0	0	1	Bajando	0	1
Bajando	0	1	0	Quieto Abajo	0	0
Bajando	0	1	1	X	X	X
Bajando	1	0	0	Bajando	0	1
Bajando	1	0	1	Bajando	0	1
Bajando	1	1	0	Quieto Abajo	0	0
Bajando	1	1	1	X	X	X

Se necesitan dos bits para codificar los estados. La codificación se realiza de la siguiente manera:

Quieto Abajo = 00

Subiendo = 01

Quieto Arriba = 10

Bajando = 11

Usando flip-flops tipo D (equación Q(t+1) = D(t)), llegamos a la siguiente tabla de verdad.

Estado Actual		Entrada			Prox. Estado		Salida	
Q2	Q1	Botón	Cocina	Contador	D1	D0	Sentido	Motor
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0
0	0	0	1	1	X	X	X	X
0	0	1	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	0	1	1	0	0	1	1	1
0	0	1	1	1	X	X	X	X
0	1	0	0	0	0	1	1	1
0	1	0	0	1	1	0	0	0
0	1	0	1	0	0	1	1	1
0	1	0	1	1	X	X	X	X
0	1	1	0	0	0	1	1	1
0	1	1	0	1	1	0	0	0
0	1	1	1	0	0	1	1	1
0	1	1	1	1	X	X	X	X
1	0	0	0	0	X	X	X	X
1	0	0	0	1	1	0	0	0
1	0	0	1	0	X	X	X	X
1	0	0	1	1	X	X	X	X
1	0	1	0	0	X	X	X	X
1	0	1	0	1	1	1	0	1
1	0	1	1	0	X	X	X	X
1	0	1	1	1	X	X	X	X
1	1	0	0	0	1	1	0	1
1	1	0	0	1	1	1	0	1
1	1	0	1	0	0	0	0	0
1	1	0	1	1	X	X	X	X
1	1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	0	1

1	1	1	1	0	0	0	0	0
1	1	1	1	1	X	X	X	X

Los diagramas de Karnaugh son los siguientes:

	Q2 = 0					Q2 = 1			
CocinaComedor\Q1Boton	00	01	11	10	CocinaComedor\Q1Boton	00	01	11	10
00	0	0	1	1	00	X	X	1	1
01	0	0	0	0	01	0	1	1	1
11	X	X	X	X	11	X	X	X	X
10	0	1	1	1	10	X	X	0	0

D0 = !Q2.Boton.Cocina + !Q2.Q1.!Comedor + Q2.Boton.!Cocina + Q2.Q1.!Cocina

	Q2 = 0					Q2 = 1			
CocinaComedor\Q1Boton	00	01	11	10	CocinaComedor\Q1Boton	00	01	11	10
00	0	0	0	0	00	X	X	1	1
01	0	0	1	1	01	1	1	1	1
11	X	X	X	X	11	X	X	X	X
10	0	0	0	0	10	X	X	O	0

D1 = Comedor.Q1 + !Cocina . Q2

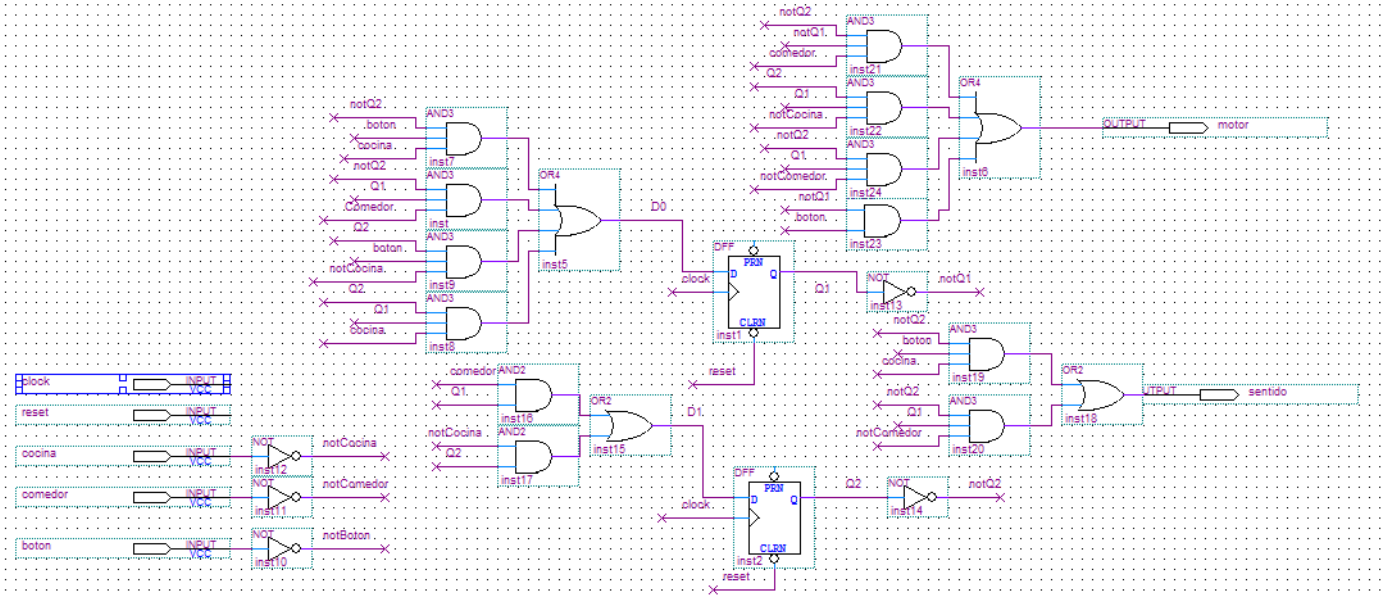
	Q2 = 0					Q2 = 1			
CocinaComedor\Q1Boton	00	01	11	10	CocinaComedor\Q1Boton	00	01	11	10
00	0	0	1	1	00	X	X	0	0
01	0	0	0	0	01	0	0	0	0
11	X	X	X	X	11	X	X	X	X
10	0	1	1	1	10	X	X	0	0

Sentido = !Q2.Boton.Cocina + !Q2.Q1.!Comedor

	Q2 = 0					Q2 = 1			
CocinaComedor\Q1Boton	00	01	11	10	CocinaComedor\Q1Boton	00	01	11	10
00	0	1	1	1	00	X	X	1	1
01	1	1	0	0	01	0	1	1	1
11	X	X	X	X	11	X	X	X	X
10	0	1	1	1	10	X	X	0	0

Motor = !Q2.!Q1.Comedor + !Q1.Boton + Q2.Q1.!Cocina + !Q2.Q1.!Comedor

El circuito resultante es el siguiente:


block
reset
cocina
comedor
boton
INPUT
VCC
NOT
inst12
inst11
inst10
notCocina
notComedor
notBoton
notQ2
notQ1
comedor
Q1
Q2
cocina
boton
AND3
AND2
inst7
inst
inst9
inst8
inst16
inst17
OR4
OR2
inst5
inst15
D0
D1
DFF
PRN
D
Q
CLRN
inst1
inst2
Q1
Q2
notQ1
notQ2
inst13
inst14
inst21
inst22
inst24
inst23
inst19
inst20
inst6
inst18
OUTPUT
motor
sentido