Examen de Arquitectura de Computadoras

17 de diciembre de 2022

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 de hojas entregadas.
- Apague su celular. No puede utilizar material ni calculadora.
- La duración del examen es de tres horas, incluyendo el tiempo para completar sus datos.
- Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.

Pregunta 1

Describa el algoritmo para comparar dos números utilizando sus representaciones en Punto Flotante IEEE 754 (normalizados). Justifique el porqué de cada paso del algoritmo.

Pregunta 2

Se desea utilizar una ROM para convertir números representados por cuatro dígitos BCD, sin signo, a representación binaria. Indicar el mínimo número de entradas y salidas de la ROM y escriba el programa en C que genere su contenido para que cumpla dicha función.

Pregunta 3

Se dispone de una CPU operando con una memoria RAM y una caché, de los cuales se conocen los siguientes datos:

- En la caché existen 512 conjuntos.
- El campo 'tag' de la dirección ocupa 16 bits.
- Cuando ocurre un miss en la caché, desde memoria principal se transfieren 128 bytes.
- Se utiliza un algoritmo de sustitución aleatorio (random), y al producirse un reemplazo, cada línea del conjunto tiene un 25% de chances de ser reemplazada.

Indique el largo de la dirección de memoria. Indique la capacidad de la caché y su función de correspondencia. Justifique.

Pregunta 4

Partiendo de la expresión de suma de productos canónicos de una función booleana, explique por qué la compuerta NAND se considera un conjunto logicamente completo. Justifique detalladamente.

Problema 1

Sea la siguiente definición de los números de Jacobsthal escalados:

$$J(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \frac{J(n-1)+2*J(n-2)+1}{2} & \text{if } n > 1 \end{cases}$$

implementado con el siguiente programa de alto nivel:

```
unsigned short jacobshal(unsigned short n)
{
  if(n < 2)
    return n;
  else return (jacobshal(n-1) + 2*jacobshal(n-2) + 1) / 2;
}
```

Se pide:

a) Compilar en 8086 sabiendo que los parámetros se pasan y retiran del stack. Peservar el contenido de los registros

b) Calcular el máximo valor de n para el cual se puede calcular la función

Problema 2

Para aumentar la emoción que provoca el espíritu navideño, **Papá Noél** ha decidido colocar luces de colores en los carros de sus renos, y le ha encomendado a usted -su duende de confianza-, el diseño del circuito secuencial que las controla.

Las luces de las que se dispone cuentan con 4 colores (rojo, amarillo, azul y verde), y se encienden colocando un 1 en el bit de salida **encendido** (un 0 apaga las luces). Para el control del color se utilizan los bits de salida **color[1..0]**, según la siguiente tabla:

azul: 00 rojo: 01 amarillo: 10 verde: 11

Para mantener la seguridad aérea, **Papá Noél** desea que mientras los renos vuelen a una altura muy alta, las luces se comporten como las de un avión, alternando 4 segundos en rojo – 4 segundos apagadas. Para detectar que los renos vuelan a una altura muy elevada, el trineo está equipado con un altímetro binario, un 1 en la entrada **altímetro** indica que el trineo está volando muy alto.

Cuando el trineo se encuentra a menor altura (entrada **altímetro** en 0), las luces deben realizar la siguiente secuencia: 2 segundos azul, 1 segundo amarillo, 2 segundos rojo, 1 segundo en verde, y luego volver a comenzar.

Se pide: Diseñar, siguiendo la metodología del curso, un circuito secuencial en base a flip flops tipo D y compuertas básicas que controle las salidas encendido y color[1..0] en base a la entrada altímetro. Hay un reloj de 1Hz.

Nota: al cambiar el valor de la señal altímetro, es válido que las luces comiencen a funcionar en cualquier punto de la nueva secuencia.

Solución

Respuesta Pregunta 1

En primer lugar se comparan los signos del número. Si un número es positivo y el otro negativo ya se obtiene el resultado de la comparación.

Si los signos son iguales, se compara todo el resto de la tira utilizando un comparador de binario natural. Esto es válido porque el resto de la tira está formada por `| exponente | mantisa |`, donde el exponente está representado en desplazamiento (el cual conserva el orden y por tanto es válido compararlos con un comparador binario). Finalmente, es correcto comparar las tiras enteras, porque por construcción de los números de punto flotante, la parte 1,F siempre es menor que 2. Por lo tanto cualquier diferencia en el exponente es mayor que la mayor diferencia posible entre las mantisas. Finalmente, si los exponentes son iguales, el resultado de la comparación estará guiada por el resultado de la comparación entre las mantisas, resultado ya provisto en la comparación binaria natural de la tira indicada.

Si los números son ambos positivos, el que tenga la tira `| exponente | mantisa |` mayor será el mayor, si son ambos negativos será el menor.

Nota: En esta respuesta se omiten los casos especiales donde los números sean infinito o NaN. Se asume que los números recibidos son normalizados, desnormalizados o cero.

Respuesta Pregunta 2:

La entrada de la ROM son 4 dígitos BCD. Como cada dígito ocupa 4 bits, la entrada de la ROM debe ser de 16 bits. Por otro lado, el máximo número representable con 4 dígitos BCD es 9999. Como $16384 = 2^{14} < 9999 < 2^{13} = 8192$, bastan 14 bits para representarlos. La salida de la ROM tendrá entonces 14 bits.

```
short ROM[65536];

void cargaROM(){
      for (unsigned short i = 0; i < 65536; i++){
            short bin = 0;
            char base = 1;
            for (char j = 0; j <= 12; j+=4){
                  char numN = (i >> j) & 0x0F;
                  bin += base * numN;
                  base = base * 10;
            }
            ROM[i] = bin;
      }
}
```

Respuesta Pregunta 3:

Como al producirse un reemplazo cada línea tiene un 25% de chances de ser reemplazada, se deduce que hay 4 líneas por conjunto, es decir, que la función de correspondencia es asociativa por conjuntos de 4 vías. Por esta razón, la dirección se interpreta como | tag | conjunto | byte |

En la caché existen 512 conjuntos, por lo tanto se requieren 9 bits para direccionarlos. Es decir, el campo conjunto ocupa 9 bits. Por otro lado, como se transfieren 128 bytes desde memoria principal en cada miss, sabemos que los bloques ocupan 128 bytes (y por tanto las líneas también). El campo byte requiere entonces 7 bits para poder direccionar cada byte del bloque. Por último, como por letra el campo tag ocupa 12 bits, la dirección completa ocupa 9 + 7 + 16 = 32 bits.

Finalmente, como en la caché hay 512 conjuntos, cada uno de 4 líneas (512 * 4 = 2048 en total), y cada línea ocupa 128 bytes, la capacidad de la caché es 2048 * 128B = 2K * 128B = 256 KB.

Respuesta Pregunta 4:

Un conjunto lógicamente completo es aquel que permite expresar cualquier función booleana con los operadores de dicho conjunto.
Como cualquier función booleana puedo expresarla como suma de productos canónicos, basta entonces con probar que con los operadores del conjunto pueda expresar los operadores AND, OR y NOT, ya que ellos consituyen un conjunto lógicamente completo..

Uso el NAND ara construir “NOT a”:
a NAND a = NOT(a AND a) x definicion de NAND
= NOT(a) x definición de AND (a AND a = a)

ahora puedo usar NAND y NOT para construir el “a AND b”:
NOT(a NAND b) = NOT(NOT(a AND b)) x definicion de NAND
= a AND b x complemento de complemento

ahora puedo usar NOT y AND para construir “a OR b”:
NOT(NOT(a) AND NOT(b)) = a OR b x De Morgan

Solución Problema 1:

a)

```
jacobsthal proc
  push BP
  mov BP, SP
  push AX
  push BX
  mov AX, [BP+4]
  cmp AX, 2
  jb fin
  dec AX
  push AX
  call jacobsthal
  pop BX  ; BX = jacobsthal(n-1)
  dec AX
  push AX
  call jacobsthal
  pop AX
  shl AX,1 ; AX = 2*jacobsthal(n-2)
  add AX, BX
  inc AX
  shr AX,1
fin:
  mov [BP+4], AX
  pop BX
  pop AX
  pop BP
  ret
jacobsthal endp
```

b)

Se asume que el cálculo de la función no produce problemas de desbordamiento de representación, por lo que la restricción Calculamos el consumo de stack para J(n).

Para los pasos base (n=0 o n=1) se consumen 10 bytes: 2 bytes por el parámetro/resultado, 2 por la dirección de retorno y 6 para preservar el contexto (AX, BX y BP)

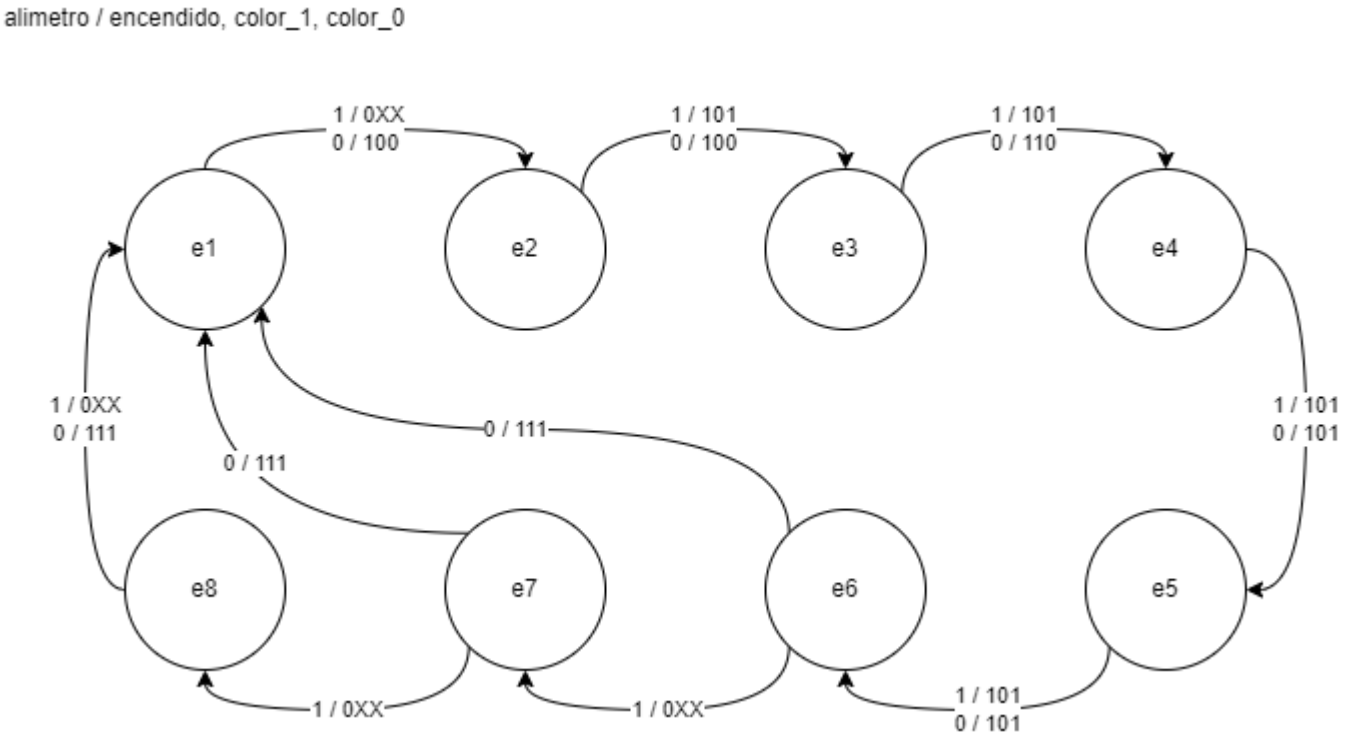
Para cualquiera de las dos llamadas recursivas también se consumen 10 bytes: 2 bytes por el parámetro/resultado, 2 por la dirección de retorno y 6 para preservar el contexto (AX, BX y BP)

A efectos del cálculo del consumo de stack para J(n) se debe tomar en particular la rama que llama a J(n-1) pues tiene mayor anidamiento en profundidad.

El consumo de stack para n > 1 queda C(n) = 10 * (n-1) + 10 = 10 * n; (n-1 pasos recursivos y 1 paso base)

Como el segmento de stack ocupa 65536 bytes (64 KB), se debe cumplir que 10 * n <= 65536, por lo que el mayor n que se puede computar es 6553.

Solución Problema 2:



Como se tienen 8 estados, se precisan 3 bits para codificarlos. Es decir, se precisan 3 FF para guardar el estado.

Codificación de estados (bits q2q1q0):

e1: 001 e5: 101
e2: 010 e6: 110
e3: 011 e7: 111
e4: 100 e8: 000

Tabla de transiciones y salidas:

Estado Actual			Entrada	Próximo Estado			Salidas		
q2(n)	q1(n)	q0(n)	altímetro	q2(n+1)	q1(n+1)	q0(n+1)	encendido	color[1]	color[0]
0	0	0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	0	X	X
0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	1	0	0	X	X
0	1	0	0	0	1	1	1	0	0
0	1	0	1	0	1	1	1	0	1
0	1	1	0	1	0	0	1	1	0
0	1	1	1	1	0	0	1	0	1
1	0	0	0	1	0	1	1	0	1
1	0	0	1	1	0	1	1	0	1
1	0	1	0	1	1	0	1	0	1
1	0	1	1	1	1	0	1	0	1
1	1	0	0	0	0	1	1	1	1
1	1	0	1	1	1	1	0	X	X
1	1	1	0	0	0	1	1	1	1
1	1	1	1	0	0	0	0	X	X

Dado que en los flip flops tipo D, D(n) = Q(n+1), la ecuación de las entradas del FF coinciden con las columnas del próximo estado

Minimización por Karnaugh:

q2q1\ q0a	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	0	1	0	0
10	1	1	1	1

d2 = q2.!q1 + !q0.a.q2 + !q2.q1.q0

q2q1\ q0a	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	1	0	0
10	0	0	1	1

d1 = !q2.q1.!q0 + !q0.a.q1 + !q1.q0

q2q1\ q0a	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	1	1	0	1
10	1	1	0	0

d0 = !q0 + q2.q1.!a

q2q1\ q0a	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	1	0	0	1
10	1	1	1	1

e = q2.!q1 + !q2.q1 + !a

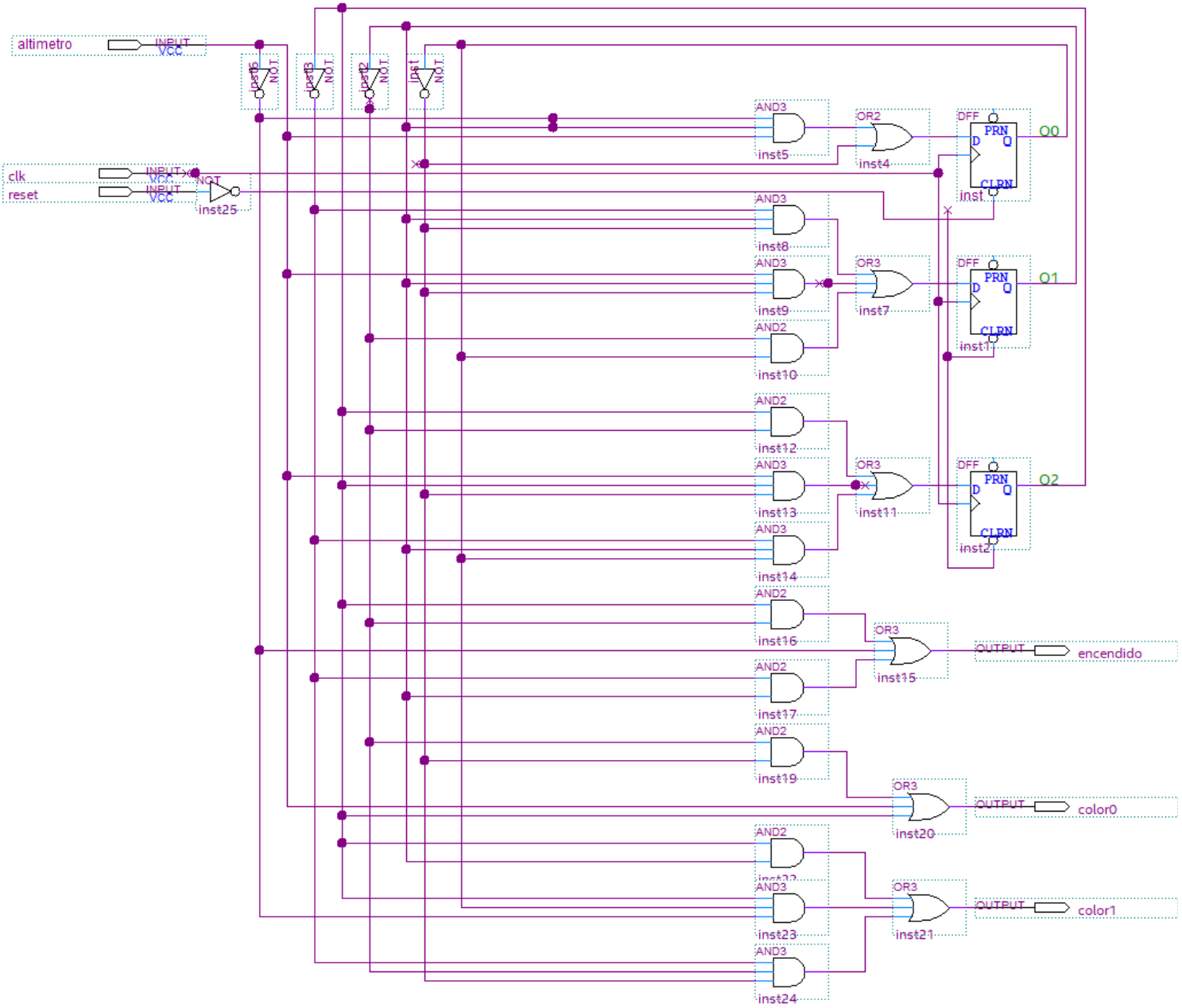
q2q1\ q0a	00	01	11	10
00	1	X	X	0
01	0	0	0	1
11	1	X	X	1
10	0	0	0	0

color1 = q2.q1 + q1.q0.!a + !q2.!q1.!q0

q2q1\ q0a	00	01	11	10
00	1	X	X	0
01	0	1	1	0
11	1	X	X	1
10	1	1	1	1

color0 = a + q2 + !q0.!q1

Circuito:


altimetro
INPUT
VCC
clk
reset
NOT
inst25
AND3
AND2
OR2
OR3
DFF
PRN
D
Q
CLRN
O0
O1
O2
inst
inst1
inst2
inst4
inst5
inst7
inst8
inst9
inst10
inst11
inst12
inst13
inst14
inst15
inst16
inst17
inst19
inst20
inst21
inst23
inst24
OUTPUT
encendido
color0
color1