Examen de Arquitectura de Computadoras

13 de diciembre de 2024

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 debe también completar sus datos.**
- **Para aprobar debe contestar correctamente un ejercicio entero y dos preguntas.**

Pregunta 1

Describa cada etapa del ciclo de instrucción e indique en qué etapas intervienen los registros visibles y con qué fin.

Pregunta 2

Construya un Flip Flop JK a partir de un Flip Flop tipo D.

Pregunta 3

Para las representaciones en complemento a 1, complemento a 2 y desplazamiento ($d=2^{n-1}$) de n bits:

a) Indicar el rango de representación
b) Codificar el cero
c) Codificar el menor y el mayor número representable.

Pregunta 4

Considere una CPU de 16 bits (con direcciones de 16 bits), operando con una memoria cache de 1 KByte y un total de 16 líneas, con una correspondencia asociativa de N vías (N>=2 y potencia de 2) y un algoritmo de sustitución FIFO. La memoria se direcciona de a byte.

a) Indique cómo se interpreta la dirección de memoria para acceder a la cache.

b) Asumiendo que se leen secuencialmente elementos de un arreglo de shorts, indicar el índice en que se produce el primer reemplazo. Considere que la caché inicia vacía y que el arreglo se ubica a partir de la dirección 0 de memoria.

```
short arreglo[M]; // donde M es muy grande
```

Problema 1

Considere la siguiente estructura:

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

y la siguiente función:

```
short nodos_mayores (nodo* arbol, short umbral) {
   if (arbol == NULL)
       return 0;
   else if (arbol->valor <= umbral) {
       arbol->diferencia = 0;
       return nodos_mayores(arbol->izq, umbral) + nodos_mayores(arbol->der, umbral);
   else {
       arbol->diferencia = arbol->valor - umbral;
       return 1 + nodos_mayores(arbol->izq, umbral) + nodos_mayores(arbol->der, umbral);
   }
}
```

Parte a)

Compilar la función nodos_mayores en assembler 8086. Considere que los punteros son desplazamientos respecto a ES. Los parámetros se reciben por el stack en el orden indicado en el cabezal de la función y el resultado se retorna por el stack.

Parte b)

Teniendo en cuenta el consumo de la estructura de datos y el consumo del stack, calcular el máximo valor de N (cantidad de nodos) para que la función pueda ejecutar siempre en una máquina 8086. Justifique.

Problema 2

Sea un número x de 9 bits, representado en binario: $x = x_8x_7x_6x_5x_4x_3x_2x_1x_0$

Se define la función $\psi(x)$ que invierte el orden de los bits tal que: $\psi(x) = x_0x_1x_2x_3x_4x_5x_6x_7x_8$

Se desea construir una ROM que tome como entrada un número x de 9 bits, representado en binario y compute: $\psi(x-1)$.

Para ello se dispone de las siguientes ROMs:

- 1 ROM de 256x9
- 2 ROMs de 64x8
- 2 ROMs de 128x5

Se pide:

a. Indicar entradas, salidas, organización y capacidad (tamaño) de la ROM necesaria para resolver este problema. Construirla usando las ROMs disponibles y compuertas básicas.

b. Escribir un programa en C que cargue la ROM construida para computar la función propuesta.

Solución

Pregunta 1

El ciclo de instrucción de 5 etapas presentado en el curso consta de los siguientes pasos:

- Fetch: este paso consiste en leer la próxima instrucción a ejecutarse desde la memoria.
- Decode: en este paso se analiza el código binario de la instrucción para determinar qué se debe realizar (cuál operación, con qué operandos y dónde guardar el resultado).
- Read: en este paso se accede a memoria para traer los operandos.
- Execute: es la ejecución de la operación por parte de la ALU sobre los operandos.
- Write: en el último paso se escribe el resultado en el destino indicado en la instrucción.

En la etapa de read se leen los operandos si estos provienen de registros y en la etapa de write se escribe el resultado si se escribe en un registro. También puede considerarse que la etapa de Read solo se realiza en caso que algún operando esté en memoria (y algo similar aplica para la de Write) y en este caso los registros visibles participan en la etapa de Execute.

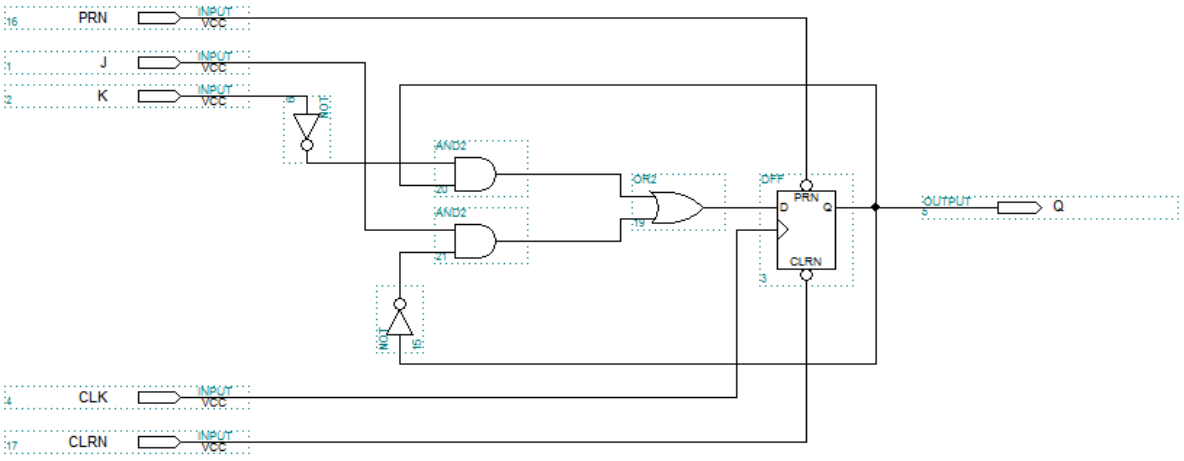
Pregunta 2

Construya un Flip Flop JK a partir de Flip Flop tipo D.

La función característica del Flip Flop tipo D es: $Q_{n+1} = D_n$

y la del Flip Flop tipo JK es: $Q_{n+1} = J.Q_n' + K'.Q_n$

entonces, $D_n = J.Q_n' + K'.Q_n$, lo cual se implementa mediante el siguiente circuito:



Pregunta 3

a) Los rangos de las representaciones son:

- Complemento a 1: $-(2^{n-1} - 1) \leq X \leq 2^{n-1} - 1$
- Complemento a 2: $-2^{n-1} \leq X \leq 2^{n-1} - 1$
- Desplazamiento: $-2^{n-1} \leq X \leq 2^n - 1 - 2^{n-1}$

b) Los representaciones del cero son:

- Complemento a 1: tiene dos representaciones 1) $\overbrace{00...0}^{n}$ y 2) $\overbrace{11...1}^{n}$
- Complemento a 2: tiene una sola representación: $\overbrace{00...0}^{n}$
- Desplazamiento: tiene una sola representación: $1\overbrace{0...0}^{n-1}$

c) Los codificaciones del menor número representable son:

- Complemento a 1: $1\overbrace{0...0}^{n-1}$
- Complemento a 2: $1\overbrace{0...0}^{n-1}$
- Desplazamiento: $\overbrace{0}^{n}0...0$

Los codificaciones del mayor número representable son:

- Complemento a 1: $0\overbrace{1...1}^{n-1}$
- Complemento a 2: $0\overbrace{1...1}^{n-1}$
- Desplazamiento: $\overbrace{1}^{n}1...1$

Pregunta 4

a) Tenemos una cache de 1KB con 16 líneas en total => 1KB / 16 líneas = $2^{10}/2^4 = 2^6$ B/línea (256 bytes por línea) => se utilizan 6 bits para el campo BYTE.

A su vez como cada conjunto tiene N líneas, en total hay 16/N = 2^4/N conjuntos, con N en { 2, 4, 8, 16 }. Notar que si N = 16 el resultado es una función de correspondencia totalmente asociativa.

Dado que hay 2^4/N conjuntos, se precisan $\log_2(2^4/N)$ bits, o sea 4 - $\log_2(N)$ para el campo CONJUNTO.

Por último, sabemos que la dirección es de 16 bits por lo que el campo TAG tiene 16 – 6 – (4 - $\log_2(N)$) = 6 + $\log_2(N)$bits.

b) Dado que los accesos son secuenciales, las direcciones generadas por los accesos varían primero en el campo byte, luego en el campo conjunto y finalmente en el campo tag. Esto quiere decir que independientemente de la cantidad de conjuntos, estos se cargan de forma uniforme. Para que se produzca un reemplazo entonces, se debe llenar la caché y luego cargar un bloque adicional. Para esto, debemos ver cuántos elementos de tipo short (2 bytes) puede almacenar la caché: 1KB / 2 = $2^{10}/2 = 2^9$. El índice i entonces será i = 2^9, o sea 512.

Problema 1

Parte a)

```
nodos_mayores proc
      PUSH BP                  // preservar contexto
      MOV BP, SP
      PUSH AX
      PUSH BX
      PUSH CX
      MOV BX, [BP+6]           // arbol
      MOV AX, [BP+4]           // umbral
      CMP BX, 0                // if (arbol == NULL)
      JE fin
      CMP ES:[BX], AX          // ES:[BX] tiene arbol→valor
      JA else
      MOV word ptr ES:[BX+2],0 // arbol→diferencia = 0
      PUSH ES:[BX+4]           // arbol→izq
```

```
      PUSH AX                  // umbral
      CALL nodos_mayores
      POP CX
      PUSH ES:[BX+6]           // arbol→der
      PUSH AX
      CALL nodos_mayores
      POP BX
      ADD BX, CX
      JMP fin
else:
      MOV CX, ES:[BX]          // CX = arbol→valor
      SUB CX, AX               // le resto umbral
      MOV ES:[BX+2], CX        // escribo ES:[BX+2] (arbol→diferencia) con la resta
`     PUSH ES:[BX+4]
      PUSH AX
      CALL nodos_mayores
      POP CX
      PUSH ES:[BX+6]
      PUSH AX
      CALL nodos_mayores
      POP BX
      ADD BX, CX
      ADD BX, 1                // return 1 + …
fin:
      MOV [BP+6], BX
      MOV BX, [BP+2]
      MOV [BP+4], BX
      POP CX
      POP BX
      POP AX
      POP BP
      ADD SP, 2
      RET
nodos_mayores endp
```

Parte b)

Consumo del stack: El peor caso consiste en un árbol que degenera en una lista, ya que siempre se deben recorrer todos los nodos y en este caso se anidarán todas las llamadas recursivas.

Una llamada recursiva consume:

- 4 bytes de parámetros: 2 bytes por el parámetro árbol y 2 bytes por el parámetro umbral
- 2 bytes para la dirección de retorno (IP)
- 8 bytes para preservación del contexto: se preservan 4 registros (BP, AX, BX y CX) con 2 bytes por cada uno de ellos, resultando en 2*4=8 bytes

En total, se tiene que cada llamada recursiva consume 4+2+8 = 14 bytes.

El paso base consiste en la llamada con el árbol NULL, que consume la misma cantidad de bytes que una llamada recursiva.

Entonces, tenemos que para un árbol de N nodos, se realizarán N llamadas recursivas y una llamada final para el caso base, por lo cual el máximo consumo de stack será 14*(N+1). Para poder ejecutarse debe cumplir que el stack no desborde, es decir, que pueda contenerse en un solo segmento. Es decir: $14*(N+1) \leq 2^{16} = 65536 \Rightarrow N = (65536/14)-1=4680$

Consumo de la estructura de datos: Cada nodo ocupa 8 bytes: dos para el dato (de tipo short), dos para la diferencia (de tipo short) y dos para cada puntero (desplazamiento). Como sólo se utilizan los desplazamientos para identificar al siguiente nodo, entonces todos los nodos deben pertenecer al mismo segmento. Un segmento son $2^{16} = 65536$ bytes. La dirección 0 se utiliza para representar NULL. De esta forma, la cantidad máxima de nodos es el número N que maximiza la siguiente ecuación: $8*N \leq 65536 - 1$. Entonces N = 8191.

Para poder ejecutarse en cualquier caso debe cumplir que la estructura de datos y el stack puedan ser contenidos en un segmento. Por lo que se debe cumplir que N = min(4680, 8191) = 4680, por ende la cantidad de nodos está limitado por la condición más restrictiva que es la del stack.

Problema 2

Parte a)

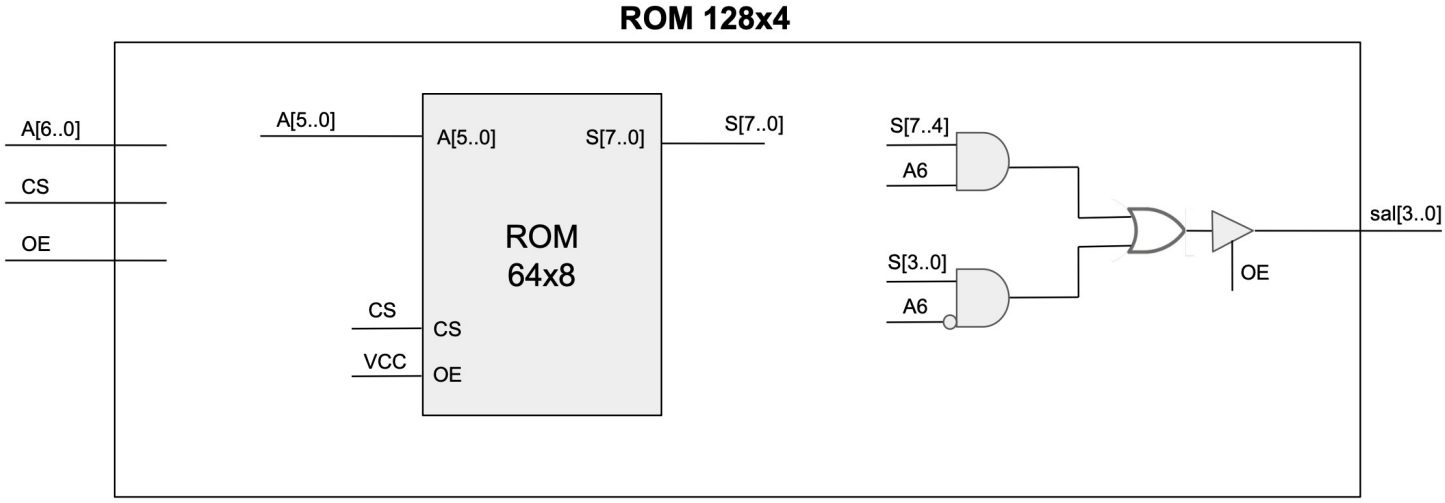
Entrada: es el número x de 9 bits, por lo tanto la entrada es de 9 bits

Salida: la salida es el resultado de la función ψ, que invierte un número de 9 bits, por lo tanto la salida es de 9 bits

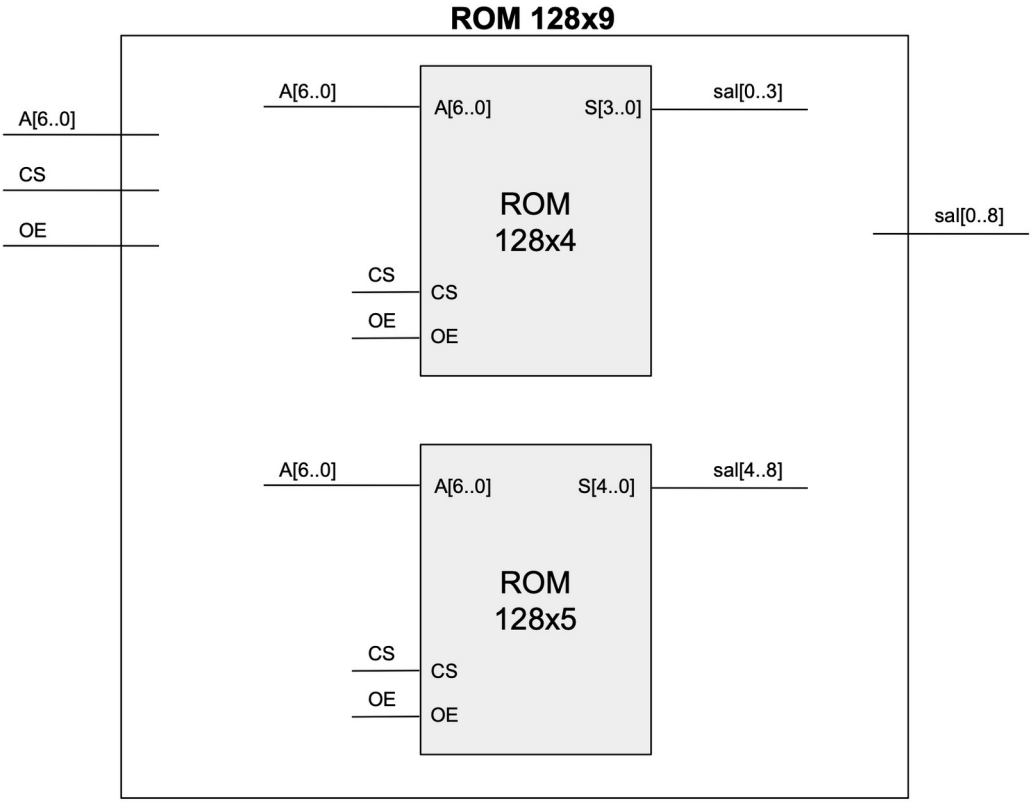
Organización: 2^9 x 9 = 512x9

Tamaño: La ROM es de 512 palabras de 9 bits, 512x9=4608, 4608/8=576, por lo tanto la capacidad es de 576 bytes o 4,5Kbits (4608/1024).

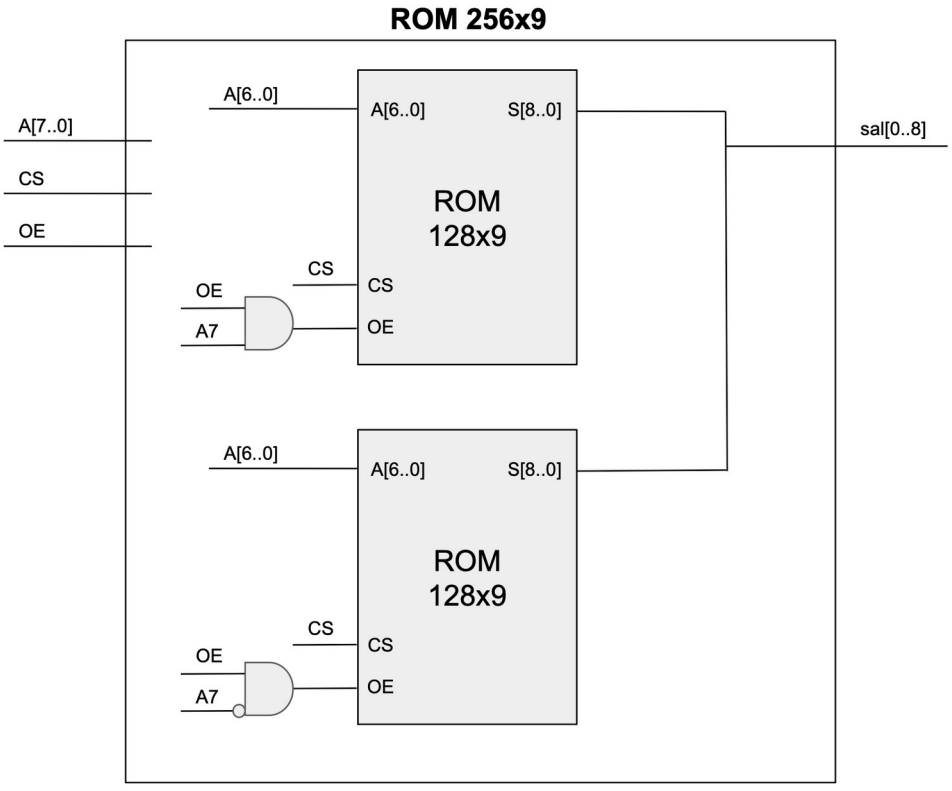
Para la construcción primero usamos una ROM de 64x4 para obtener una ROM de 128x4:



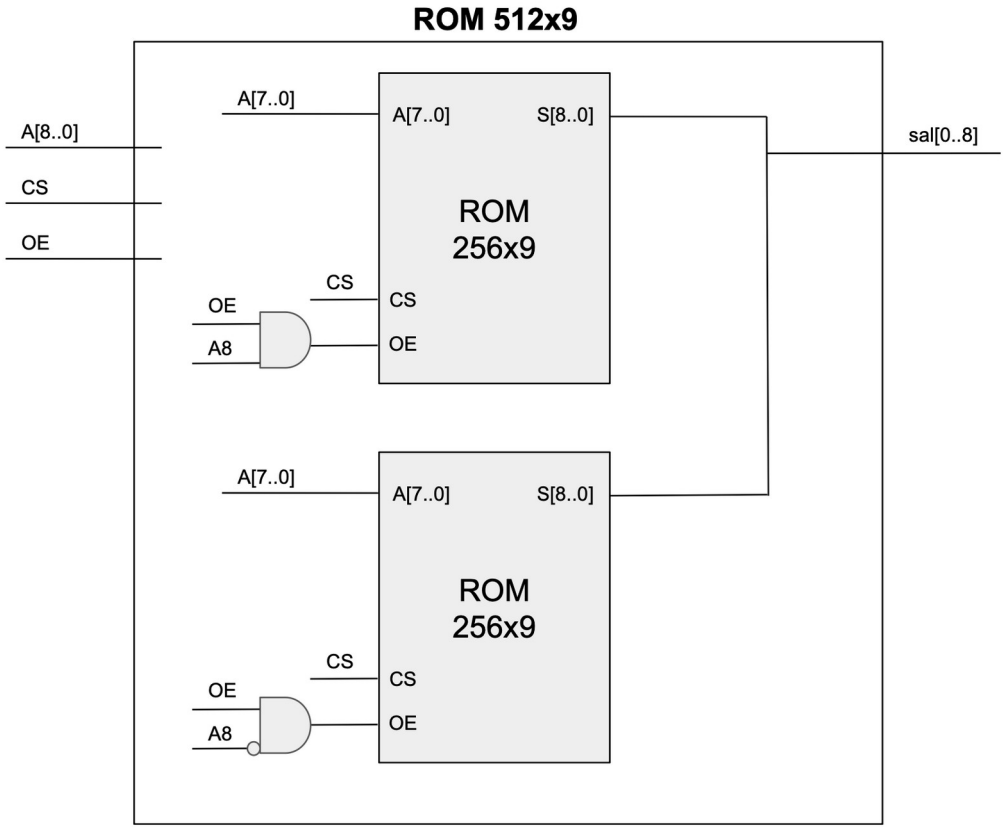
Luego, usamos esta ROM de 128x4 y una de 128x5 para construir una ROM de 128x9:



Luego, usamos dos de estas ROMs de 128x9 para construir una ROM de 256x9:



Por último, usamos dos de estas ROMs de 256x9 para construir una ROM de 512x9:



Parte b)

```
short ROM [512];
void cargar_rom(){
      for (short i = 0; i < 512; i++)
            short invertido = 0;
            short num = i - 1;
            for (char j = 0; j < 9; j++){
                  invertido = invertido | ((num >> j) & 1) << (8-j);
            }
            ROM[i] = invertido & 0x1FF;
      }
}
```