# Examen de Arquitectura de Computadoras

# 21 de julio de 2023

**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*

Dado un procesador con formato de instrucción de largo fijo de 32 bits con tamaño de memoria 256KB direccionable de a byte. Todas las instrucciones tienen el mismo formato:

- Un campo de código de operación.
- Un campo de modo, para especificar uno de 7 modos de direccionamiento.
- Un campo de registro, para especificar uno de 60 registros.
- Una dirección de memoria.

Especificar el formato de instrucción que maximiza la cantidad de códigos de operación, indicando la cantidad de bits de cada campo.

### Solución:

Para el formato de instrucción indicado, se precisan:

3 bits para el campo modo, para indicar uno de los 7 posibles modos de direccionamiento.
6 bits para especificar el registro (2 ^ 6 = 64)
18 bits para indicar la dirección de memoria (256KB direccionables de a byte)

Por lo tanto, el formato es como se especifica a continuación:

[ opcode | modo(3) | registro (6) | dir(18) ]

Dado que el formato de instrucción es de largo fijo de 32 bits, las definiciones anteriores permiten asignar 5 bits para el campo opcode. Por tanto, este formato de instrucción permite como máximo 32 códigos de operación diferentes.

## *Pregunta 2*

**a)** Nombre los tres componentes principales de una CPU.
**b)** Describa la ALU, incluyendo sus entradas y salidas típicas, e indicando qué tipo de circuito es. Nombre cuatro operaciones habituales.

### Solución:

**a)** Unidad de Control, Banco de Registros y Unidad Aritmético-Lógica (ALU)

**b)** La Unidad Aritmético-Lógica es un conjunto de circuitos (típicamente combinatorios) que implementan un conjunto de operaciones, las cuales habitualmente incluyen suma y resta (en aritmética complemento a 2), operaciones lógicas bit a bit (AND, OR, EXOR, NOT) y operaciones de desplazamiento (shift). Las ALUs más avanzadas incluyen operaciones de multiplicación y división (aunque en este caso se implementan como una máquina secuencial que implementa algún algoritmo para estas operaciones).

Las entradas típicas de una ALU son los operandos a y b (N bits), el código de la operación a realizar, mientras que las salidas típicas son el resultado de la operación (N bits) y las banderas de condición, por ejemplo, cero (Z), negativo (N), overflow (V) y paridad (P).

## *Pregunta 3*

**a)** ¿Qué es el retardo de propagación de una compuerta?

**b)** Explique cómo la cantidad de niveles de compuertas afecta al retardo de propagación del circuito y a la frecuencia de reloj que se puede utilizar.

### Solución:

**a)** El tiempo de retardo es el tiempo que se demora entre que se presenta una valor en una entrada de una compuerta lógica y ésta responde con la salida correspondiente al valor de la entrada y la operación lógica que implementa. Esto se debe a que las compuertas no cambian su salida en forma instantánea.

**b)** La cantidad de niveles de compuertas está directamente relacionado con el retardo de propagación total de un circuito. Los retardos de propagación se acumulan a lo largo del circuito, de forma tal que cuanto mayor es el número de niveles que recorre una entrada hasta incidir en la salida, mayor es el retardo total del circuito.

El retardo de propagación del circuito limita la frecuencia de reloj máxima que se puede utilizar, ya que el período del mismo debe ser suficientemente largo como para permitir la propagación de los valores lógicos correctos en todos los circuitos combinatorios involucrados entre las entradas y las salidas.

## *Pregunta 4*

Escriba una expresión booleana de cuatro variables, que contenga ocho términos canónicos y que no sea minimizable.

### Solución:

Hay varias formas de encontrar ocho términos que no puedan ser simplificados. Una forma posible es partir de los 16 términos canónicos de la función booleana de 4 variables y eliminar los que permitan una simplificación.

Sin embargo, una resolución visual más simple consiste en tomar los siguientes términos en un mapa de karnaugh:

| xy\zw | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 1 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 1 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 1 |

No hay agrupamientos de 2 o más celdas posibles, por lo tanto la función booleana resultante es la siguiente:

f(x,y,z,w) = !x!y!z!w + !x!yzw + !xy!zw + !xyz!w + xy!z!w + xywz + x!y!zw + x!yz!w

## *Ejercicio 2*

El restaurant de sushi ***fish'n'flop*** quiere innovar en sus cadenas con un sistema de entrega de sushi por cinta transportadora. En el restaurant existen 4 cubículos individuales dispuestos uno al lado del otro, donde los comensales realizan los pedidos a través de una tableta disponible en su cubículo. Una vez realizado el pedido, este es preparado en la cocina y luego se envía a través de la cinta transportadora. Cuando el plato de sushi llega al cubículo que la ordenó, la cinta debe detenerse para permitir que el cliente retire el plato de la cinta. Si pasados 20 segundos el cliente no retira el plato, una discreta alarma debe sonar en el cubículo para llamar la atención del cliente.

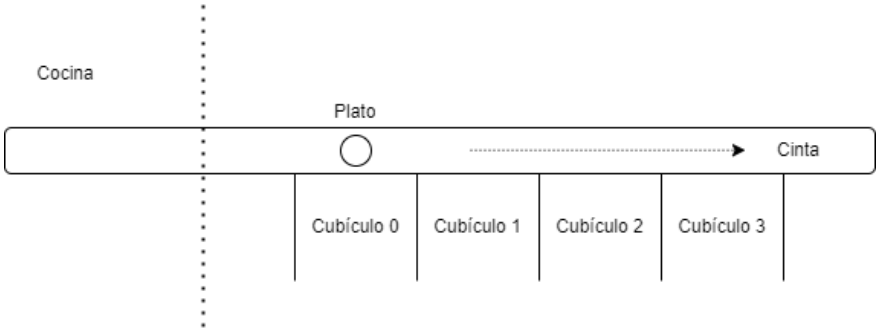


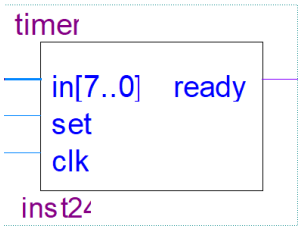

*Figura 1: vista superior*

Los platos que se envían desde la cocina tienen un código en la parte inferior que indica en qué cubículo se debe entregar el plato (cada cubículo es identificado con un código de 2 bits: 00, 01, 10, 11). El sistema cuenta con una entrada **cinta** de 12 bits, la cual indica el estado de la cinta en cada cubículo. El bit 0 indica si hay un plato frente al cubículo 0 (un en la entrada 1 indica que hay un plato), mientras que los bits [2..1] indican, en caso de que exista un plato frente al cubículo, el valor del código bajo el plato. Los bits [5..3] funcionan de forma análoga los bits [2..0] pero para el cubículo 1, y así sucesivamente.

Para avanzar la cinta se debe escribir un 1 en la salida **motor** (un 1 hace avanzar la cinta, mientras que un 0 la detiene). La cinta debe avanzar siempre que exista un plato sobre ella que deba moverse hacia adelante para llegar al cubículo deseado.

Para encender la alarma, el sistema controla con una salida **alarma** de 4 bits, la cual activa la alarma en cada uno de los cubículos: si el bit ***i*** está en 1 suena la alarma del cubículo ***i***. La alarma debe sonar hasta que el cliente retire el plato de la cinta.

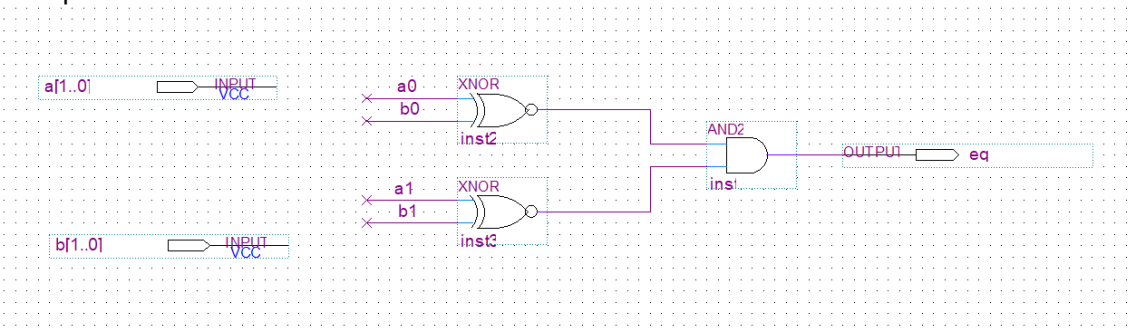
Se dispone de un circuito *timer* con carga paralela. Las entradas *in*[7..0] permiten cargar el tiempo en segundos a contar. Un 1 en *set* carga el valor de la entrada ***in*** en registros internos, momento en el que comienza la cuenta. Una vez pasado el tiempo, la salida ready produce un 1 de **un pulso de duración** (está en 1 durante un ciclo del reloj).




**Se pide:** Diseñar un circuito secuencial que controle las salidas **motor** y **alarma** en base a la entrada **cinta**. Se dispone de flip flops tipo D y compuertas básicas. **NO se debe** utilizar la metodología del curso en la resolución de este problema.
**Sugerencia:** comenzar construyendo un comparador de 2 bits.

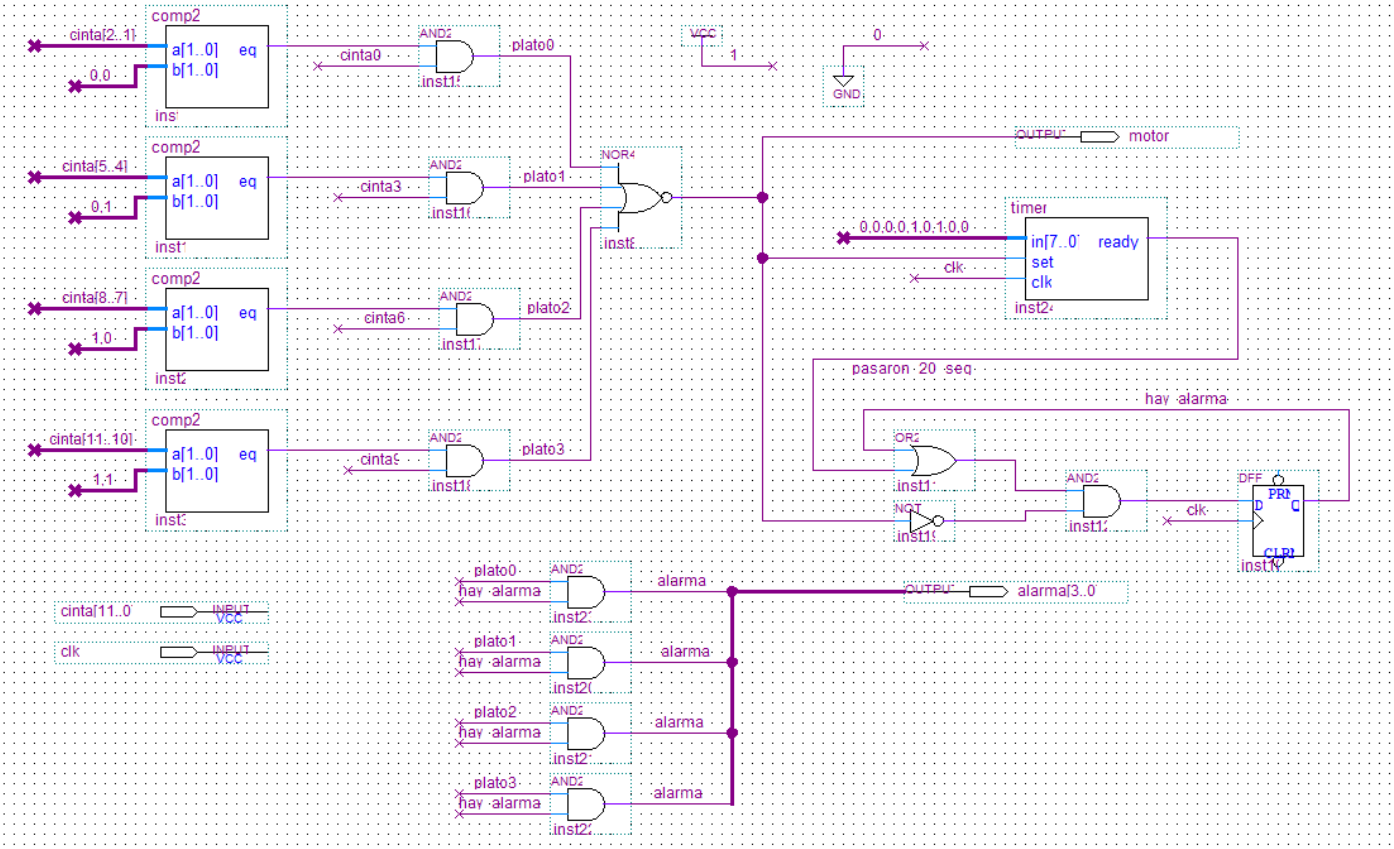
## *Solución:*

Comparador de 2 bits:



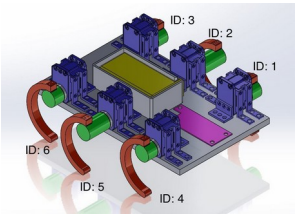

El circuito solución se describe a continuación. Lo primero a generar son las señales plato[3..0], las cuales indican en cada bit si hay un plato frente a un cubículo y que ese plato está destinado a ese cubículo. Tomando como ejemplo el cubículo 0, esa condición se da cuando el bit 0 es 1 (hay plato frente al cubículo 0), y los bits cinta[2..1] son ambos 0 (es decir, que ese plato tiene como destino el cubículo 00). Si alguno de los bits plato[3..0] vale 1, entonces se debe apagar el motor. De forma inversa, si todos esos bits valen 0, entonces se mueve la cinta y además se reinicia la cuenta del timer en 20 segundos (al reiniciarse en cada ciclo, esta jamás llega a cero).

Finalmente, resta explicar el comportamiento de la alarma. Se utiliza un flip flop D para recordar el valor de ready del reset. Dicho flip flop se vuelve a llevar a cero cuando el motor vuelve a encenderse. Por último, cada bit de alarma se prende cuando pasan 20 segundos de cinta apagada y hay un plato para retirar en el cubículo correspondiente.




**Nota**: se acepta también una solución donde se reinicie el FF con lógica combinatoria en la señal CLR del FF.

## *Ejercicio 2*




Se desea implementar un sistema para controlar un robot hexápodo. El hexápodo tiene tres patas a cada lado controladas por un motor cada una. Los motores se identifican del 0 al 5, siendo los motores de 0 a 2 los izquierdos, y los motores 3 a 5 los derechos.

La velocidad de giro de las patas, a la que se deben mover los motores asociados a cada lado del robot para generar las caminatas, son actualizadas constantemente por un dispositivo externo, quedando accesibles en los puertos de E/S de tipo byte, de solo lectura, en las direcciones **LEFT** y **RIGHT**. Estas velocidades están representadas como un entero en complemento a dos de 8 bits. El hexápodo debe mantener la caminata en todo momento salvo cuando ambas velocidades (LEFT y RIGHT) sean cero.

Los motores están acionados por un controlador de E/S. La comunicación con este controlador se realiza a través de mensajes que le son enviados a través del puerto de E/S, de tipo byte, en la dirección **MOTORES**. Un mensaje (secuencia de bytes que codifican comandos a los motores) inicia con un byte que indica la acción a ejecutar (0x1 o 0x2) y luego los parámetros. El formato de los mensajes es el siguiente:

- [0x1] [*idMotor*] [*velocidad*] (3 bytes): indica al motor *idMotor* la *velocidad* de giro. La velocidad de giro debe codificarse en representación valor absoluto y signo.
- [0x2] [*maskMotor*] (2 bytes): indica a todos los motores cuyo bit en *maskMotor* vale 1 que deben ajustar su velocidad en base a la última velocidad indicada. El bit i en *maskMotor* refiere al motor i del hexápodo.

Cada vez que el controlador de los motores recibe un byte enviado a través del puerto **MOTORES,** genera una interrupción asociada al manejador de interrupciones ***ready()*** *indicando que* está listo para recibir otro byte. Si se escribe un nuevo byte en **MOTORES** antes que esté pronto para recibir uno nuevo, el mismo será ignorado.

El controlador de los motores detiene cada motor luego de dar una vuelta completa y coloca un 1 en el bit más significativo del puerto **MOTORES** cuando todos los motores en movimiento lograron dar una vuelta. Pone en 0 automáticamente dicho bit luego que el puerto es leído por el procesador.

**Se pide**: Implementar en un lenguaje de alto nivel, preferentemente C, el sistema propuesto sabiendo que el procesador está dedicado al control del hexápodo y teniendo en cuenta que para que el robot sea estable, una manera de generar caminatas es mover los motores pares, y dejar quietos los motores impares durante una vuelta completa. Luego se deben mover los motores impares, y dejar quietos los pares durante otra vuelta completa, y así sucesivamente.

## *Solución:*

```
unsigned char mensaje[20];
int ind, mensaje_enviado;

void main() {
  char aux_left, aux_right, par = TRUE;
  // instalo manejador interrupción ready()
  mensaje[0] = 0x01; mensaje[1] = 0; mensaje[2] = 0;
  mensaje[3] = 0x01; mensaje[4] = 1; mensaje[5] = 0;
  mensaje[6] = 0x01; mensaje[7] = 2; mensaje[8] = 0;
  mensaje[9] = 0x01; mensaje[10] = 3; mensaje[11] = 0;
  mensaje[12] = 0x01; mensaje[13] = 4; mensaje[14] = 0;
  mensaje[15] = 0x01; mensaje[16] = 5; mensaje[17] = 0;
  mensaje[18] = 0x02; mensaje[19] = 0x3F;  // sincronizo todos los motores
  enable();
  while(TRUE) {
    aux_left = in(LEFT);
    aux_right = in(RIGHT);
    if ((aux_left != 0) || (aux_right != 0)) {
      if (aux_left < 0) {
        aux_left = -aux_left;
        aux_let = aux_left | 0x80;
      }
      if (aux_right < 0) {
        aux_right = -aux_right;
        aux_right = aux_right | 0x80;
      }
      mensaje[2] = par?aux_left:0;
      mensaje[5] = par?0:aux_left;
      mensaje[8] = par?aux_left:0;
      mensaje[11] = par?0:aux_right;
      mensaje[14] = par?aux_right:0;
      mensaje[17] = par?0:aux_right;;
      mensaje_enviado = FALSE;
      ind = 0; out(MOTORES, mensaje[0]);
      while (!mensaje_enviado);
      while (in(MOTORES) & 0x80 != 0x80);
      par = !par;
    }
    else {
      mensaje[2] = 0;
      mensaje[5] = 0;
      mensaje[8] = 0;
      mensaje[11] = 0;
      mensaje[14] = 0;
      mensaje[17] = 0;
      mensaje_enviado = FALSE;
      ind = 0; out(MOTORES, mensaje[0]);
      while (!mensaje_enviado);
    }
  }
}
```

```
void interrupt ready() {
  if (ind >= 20)
    mensaje_enviado = TRUE;
  else
    out(MOTORES, mensaje[ind++]);
}
```