# Examen de Arquitectura de Computadoras

## 23 de diciembre de 2015

**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*

1. Explique los principios de localidad en los que se basa una memoria Cache.
2. Dada una cache con correspondencia asociativa de N vias, indique qué valores debe tomar N para que esta se comporte como una cache con correspondencia directa y una cache con correspondencia totalmente asociativa.

**Solución:**

1. El principio de localidad establece que los programas acceden a una porción relativamente reducida del espacio de direcciones en un determinado lapso de tiempo. El principio tiene dos variantes: a) Localidad temporal: si un elemento es referenciado en determinado momento, es muy probable que vuelva a ser referenciado poco tiempo después. b) Localidad espacial: cuando un elemento es referenciado en determinado momento, es muy probable que los elementos con direcciones “cercanas” también sean accedidos poco tiempo después.

2. En cache de N vías, se puede pensar que cada bloque de memoria tiene “N posibles lugares en la cache”. En el caso de una correspondencia directa, cada bloque tiene una sola posición posible en el cache: por el argumento anterior, por lo que es como una cache de 1 vía (N = 1). Por otra parte, en el caso de una correspondencia totalmente asociativa, cada lugar de memoria tiene tantas posibilidades como lineas en el cache; por lo tanto se corresponde con un cache de L vías (N = L), donde L es la cantidad de líneas (L = tamaño de cache / tamaño de la línea).

## *Pregunta 2*

Represente 125,8125 en:

1. Punto fijo de 9 bits para la parte entera y 7 bits para parte fraccionaria
2. Punto flotante de simple precisión

**Solución:**

1. $125{,}8125_{(10)} = 1111101{,}1101_{(2)}$ , multiplicando por $2^7$, expresando en 16 bits (como es positivo no se precisa hacer el complemento a 2), obtenemos:

0011111011101000

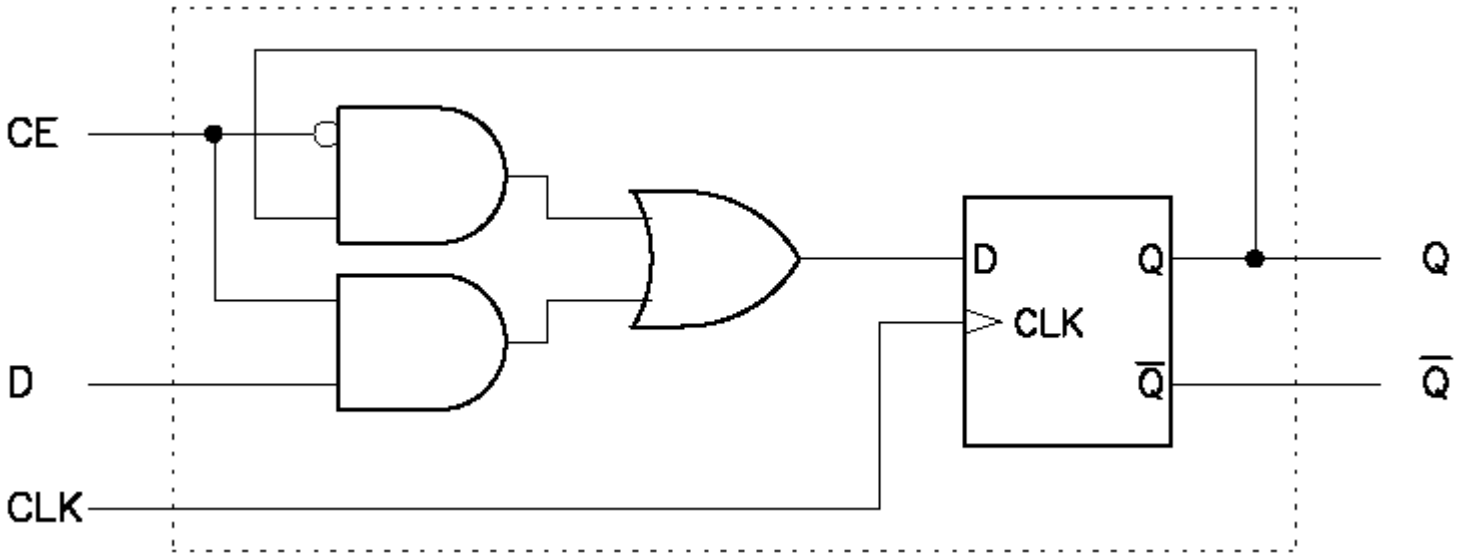
2. En precisión simple usamos 1 bit para el signo, 8 para el exponente y 23 para la mantisa. Para normalizar debemos multiplicar por $2^6$ (1,1111011101 x $2^6$) con lo que la representación queda:

0 10000101 11110111010000000000000.

## *Pregunta 3*

A partir de un flip-flop tipo D que no dispone de entrada de Clock Enable (CE) construya uno que sí la tenga.

**Solución:**




## *Pregunta 4*

Describa la técnica de salto retardado e indique cual es su objetivo.

**Solución:**

La técnica, utilizada como forma de evitar los hazards de control en un pipeline, consiste en ejecutar siempre la instrucción que está a continuación de un salto, con independencia que este se tome o no. La idea es colocar allí una instrucción útil o un NOP en caso de no ser posible. De esta forma, en caso de que el salto se tome, no se incurre en un penalización por salto. Es común en procesadores RISC.

## *Problema 1*

Con la cercanía del verano ***Cabañas Sécheresse*** desea instalar un tanque de agua en cada una de las cabañas del complejo turístico para mejorar la disponibilidad del líquido elemento para los huéspedes, pero al mismo tiempo debe facturar el consumo de agua de cada una de las cabañas de forma independiente.

Para esto se instalará un detector de nivel mínimo de agua (DNM) y una electroválvula (EV) en cada tanque, y un único medidor de caudal (MC) en la entrada general de agua, la que luego llega a los 8 tanques disponibles, junto a un microcontrolador dedicado que comandará estos elementos a los efectos de cumplir el objetivo.

El sistema deberá mantener el agua por encima del nivel mínimo en todos los tanques y contabilizar el consumo de agua de cada uno. Se asume que el caudal de agua que ingresa se reparte en partes iguales entre todos los tanques que tengan la electroválvula abierta.

Ademas, deben controlarse los siguientes aspectos:

- Una vez alcanzado el nivel mínimo de agua en un tanque debe mantenerse abierta la electroválvula durante 20 minutos más.
- Las electroválvulas no pueden permanecer abiertas por más de 2 horas. Si se alcanza este tiempo deben cerrarse y activarse la alarma.

La situación de alarma debe indicarse manteniendo encendida una sirena durante 1 minuto.

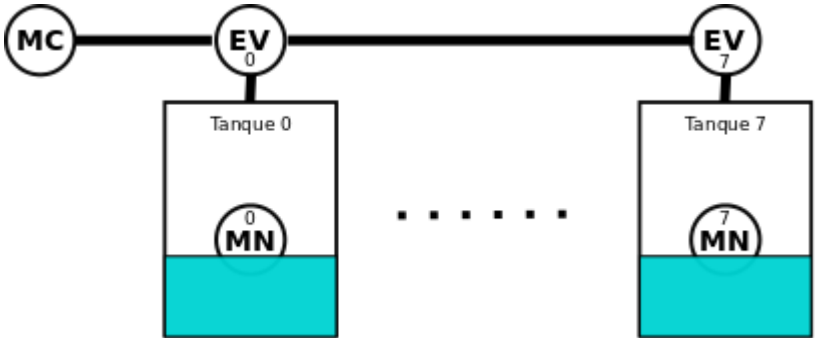
Se dispone de los siguientes puertos de E/S:

- Byte de sólo escritura para abrir (1) o cerrar (0) una EV en la dirección CONTROL_EV. El bit i controla la i-esima EV (numeradas de la 0 a la 7).
- Byte de sólo lectura que permite conocer si el nivel de un DNM está por encima del mínimo (1) o no (0) en la dirección ESTADO_DNM. El bit i consulta el estado del i-esimo DNM (numerados del 0 al 7).
- Byte de sólo escritura en la dirección SIRENA, cuyo bit menos significativo es utilizado para encender (1) o apagar (0) la sirena.
- Palabras de 16 bits de solo escritura donde se debe registrar el valor actual del consumo de agua (en litros) de cada tanque, en las direcciones CONSUMO+i, donde i es el número del tanque (numerados del 0 al 7), es decir las direcciones CONSUMO, CONSUMO+1. CONSUMO+2, etc

Se dispone de un *timer* que interrumpe a una frecuencia de 5Hz invocando a la rutina **tiempo().** El sensor de caudal genera una interrupción por cada litro de agua que pasa por él, que invoca a la rutina **caudal()**.

**Se pide:**

Implementar todas las rutinas necesarias en alto nivel (preferentemente en C).




## *Solución*

Se nos plantea la dificultad de como computar adecuadamente el consumo, ya que si usamos variables enteras que cuenten los litros de consumo de cada cabaña, tendremos problemas al intentar dividir el caudal entre todos los tanques abiertos. Esto es porque si queremos dividir cada litro que entra (en la rutina caudal()) entre las válvulas abiertas nos dará un incremento de 0 (como división entera) para cada tanque.

Hay diversas formas de resolver el problema:

- usando variables “float”: esto resuelve el problema, aunque no es la solución más elegante para un ambiente de microcontrolador.
- computando en unidades menores al litro, por ejemplo centilitros y luego llevando a litros al momento de registrar el consumo en los puertos de E/S.
- dividiendo el caudal cuando hay un cambio en el estado de apertura o cierre de las válvulas. Dado que estamos hablando de tanques de agua para casas, los mismos almacenan centenas o miles de litros. Por otra parte mantenemos abierta las válvulas por 20 minutos luego de llegado al nivel mínimo, lo que seguramente implica el pasaje de muchas decenas de litros de agua. Todo esto lleva a pensar que si hacemos la división cuando abrimos o cerramos alguna válvula, estaremos manejando cifras importantes de litros de agua, con lo que el error en la división será despreciable. Esta solución tiene el inconveniente de que la actualización del consumo se haría “de a saltos”.

Elegimos implementar la segunda variante (acumular en centilitros). Notemos que esta solución equivale a implementar “manualmente” una representación de tipo “punto fijo” decimal. Asumimos que la condición de 2 horas prevalece sobre los 20 minutos.

```
#define SEG 5
#define UNMIN SEG * 60
#define VEINTEMIN UNMIN * 20
#define DOSHORAS 6 * VEINTEMIN
#define CARGANDO 0
#define CARGANDO_PLUS 1
#define SIN_CARGAR 2
#define ALARMADO 3

unsigned int cantValvulasAbiertas;
unsigned int ticsAlarma;

unsigned long int consumoCentiLitros[8];

unsigned int ticsPorTanque[8];
unsigned int ticsAlMinimoPorTanque[8];
unsigned int estadoTanque[8];
```

```
void interrupt caudal(){
  int centiLitrosPorTanque;
  centiLitrosPorTanque = 100 / cantValvulasAbiertas;
  for(int i = 0; i < 8; i++){
    if ((estadoTanque[i] == CARGANDO) || (estadoTanque[i] == CARGANDO_PLUS)){
      consumoCentiLitros[i] += centiLitrosPorTanque;
    }
  }
}

void interrupt tiempo(){
  unsigned char niveles;
  unsigned char nuevoEstadoValvulas;
  short valorOut;

  niveles = in(ESTADO_DNM);
  nuevoEstadoValvulas = 0;      // todas cerradas salvo las que mantengo abiertas
  for(int i = 0; i < 8; i++){
    ticsPorTanque[i]++;
    if ((estadoTanque[i] == CARGANDO) || (estadoTanque[i] == CARGANDO_PLUS)){
      if (ticsPorTanque[i] < DOSHORAS){
         // Mantengo abierta
         nuevoEstadoValvulas = nuevoEstadoValvulas | 1 << i;
      }
      else {
        // Cierro y doy alarma
        ticsAlarma = UNMIN;
        // No dejo que se vuelva a abrir
        estadoTanque[i] = ALARMADO;
        cantValvulasAbiertas--;
      }
    }
    if (niveles & 1 << i){
      // Esta por encima del mínimo
      if (estadoTanque[i] == CARGANDO) {
         // Primera vez encima del mínimo
         estadoTanque[i] = CARGANDO_PLUS;
         ticsAlMinimiPorTanque[i] = ticsPorTanque[i];
      }
      if (ticsPorTanque[i] > (VEINTEMIN + ticsAlMinimiPorTanque[i])){
        // Puedo cerrar
        estadoTanque[i] = SIN_CARGAR;
        cantValvulasAbiertas--;
      } else {
        // No puedo cerrar
        nuevoEstadoValvulas = nuevoEstadoValvulas | 1 << i;
      }
    }
    else if (estadoTanque[i] != ALARMADO) {
      if (estadoTanque[i] == SIN_CARGAR){
        // Primera vez por debajo del mínimo
        estadoTanque[i] = CARGANDO;
        ticsPorTanque[i] = 0;
        nuevoEstadoValvulas = nuevoEstadoValvulas | 1 << i;
        cantValvulasAbiertas++;
      }
    }
```

```
      valorOut = (consumoCentiLitros[i] / 100);
      out(CONSUMO + i, valorOut);
    }
    if(ticsAlarma > 0){
      out(SIRENA, 0x01);
      ticsAlarma--;
    }
    else{
      out(SIRENA, 0x00);
    }
    out(CONTROL_EV, estadoValvulas);
  }

  int main(){

    // Instalo rutinas de interrupción

    for(int i = 0; i < 8; i++){
       consumoCentiLitros[i] = 0;
       ticsPorTanque[8] = 0;
       ticsAlMinimoPorTanque[i] = 0;
       int estadoTanque[i] = CARGANDO;
    }
    // Abro todas las valvulas (estado = CARGANDO)
    out(CONTROL_EV, 0xFF);
    cantValvulasAbiertas = 8;
    ticsAlarma = 0;
    out(SIRENA, 0x00);
    enable();

    while(TRUE){
    }

  }
```

## *Problema 2*

Suponga un procesador 8086, la estructura “unsigned char Darreglo[N] ” (un arreglo ordenado de menor a mayor de naturales de 8 bits) y el siguiente fragmento de código que realiza la búsqueda binaria de “dato” en “Darreglo”, retornando 1 si lo encuentra y -1 en caso contrario:

```
int busquedaB(unsigned char dato, short indice_min, short indice_max){
   char menor,medio,mayor;
   short indice_mid = (indice_min + indice_max) / 2;
   menor = Darreglo[indice_min];
   mayor = Darreglo[indice_max];
   if (menor > mayor)
      return -1;
   else{
      medio = Darreglo[indice_mid];
      if(medio == dato)
         return 1;
      elsif( medio > dato)
         return busquedaB(dato, indice_min, indice_mid - 1);
      else
         return busquedaB(dato, indice_mid + 1, indice_max);
   }
}
```

donde [indice_min, indice_max] es el intervalo del arreglo en el que se está realizando la búsqueda.

### Se pide:

### Parte a)

Compilar el código en assembler 8086. La estructura Darreglo está cargada a partir de la posición 0 del segmento extra y la invocación se realiza de la siguiente manera:

```
mov DX, Valor_a_buscar
mov SI, indice_min
mov DI, indice_max
push DX
push SI
push DI
call busquedaB
pop Resultado
```

Se debe preservar el valor de todos los demás registros.

### Parte b)

Calcular el tamaño mínimo que debería tener el stack para el caso particular en que Darreglo es:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | 8 | 11 | 27 | 59 | 101 | 121 | 233 |

y la llamada es:
“push 9”
“push 0”

“push 7”
call buscarB
pop resultado

**Solución:**

**Parte a)**

```
proc buscarB

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

    mov SI,[BP + 6]        ; indice_min
    mov DI,[BP + 4]        ; indice_max

    mov BX,SI
    add BX,DI
    shr BX,1               ; BX = inice_mid

    mov AL, ES:[SI]        ; menor
    mov AH, ES:[DI]        ; mayor
    cmp AL,AH
    jg noencontre

    xor DX,DX
    mov CX, [BP + 8]       ; dato pasado como argumento
    mov DL, ES:[BX]        ; medio
    cmp DX,CX
    je encontre
    jg iteroIzq

    inc BX
    push CX
    push BX
    push DI
    call buscarB
    pop BX
    jmp fin

iteroIzq:

    dec BX
    push CX
    push BX
    push SI
```

```
    call buscarB
    pop BX
    jmp fin

noencontre:

    mov BX,-1
    jmp fin

encontre:

    mov BX,1

fin:
    pop DX
    pop CX
    pop AX
    pop DI
    pop SI
    mov [BP + 8], BX
    mov BX, [BP + 2]
    mov [BP + 6], BX
    pop BX
    pop BP
    add SP,4

    ret

endp buscarB
```

**Parte b)**

El valor a buscar en el arreglo es 9, que no pertenece.

En cada llamada a la función se utilizan 22 bytes. 6 por los parámetros al llamarla, 2 por el IP y 14 por los registros salvados en el stack.

| indice_min | indice_max | indice_mid | medio | |
|---|---|---|---|---|
| 0 | 7 | 3 | 27 | Medio > dato |
| 0 | 2 | 1 | 8 | Dato > medio |
| 2 | 2 | 2 | 11 | Medio > dato |
| 2 | 1 | - | - | Min > Max ! |

Como se requieren 4 llamadas a la función, el tamaño del stack requerido es de 22*4 = 88 bytes.