Examen de Arquitectura de Computadoras

23 de diciembre de 2014

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. Indique qué es y cuál es la función del chip select (CS) y el output enable (OE).
2. Construya una ROM de 32x8 en base a dos ROMs de 16x8.

Pregunta 2

1. Indique para cada uno de los siguientes sistemas si son capaces de corregir errores:
   - Código de Hamming
   - Paridad
   - Entrelazado 2 de 5
   - Paridad horizontal+vertical
2. Codifique en el sistema de Hamming la siguiente tira : 1001

Pregunta 3

1. Enumere y describa los componentes de la arquitectura de Von Neumann.
2. Enumere y describa las etapas del ciclo de instrucción.

Pregunta 4

En la dirección de memoria ES:[DI] está guardado el entero -15.

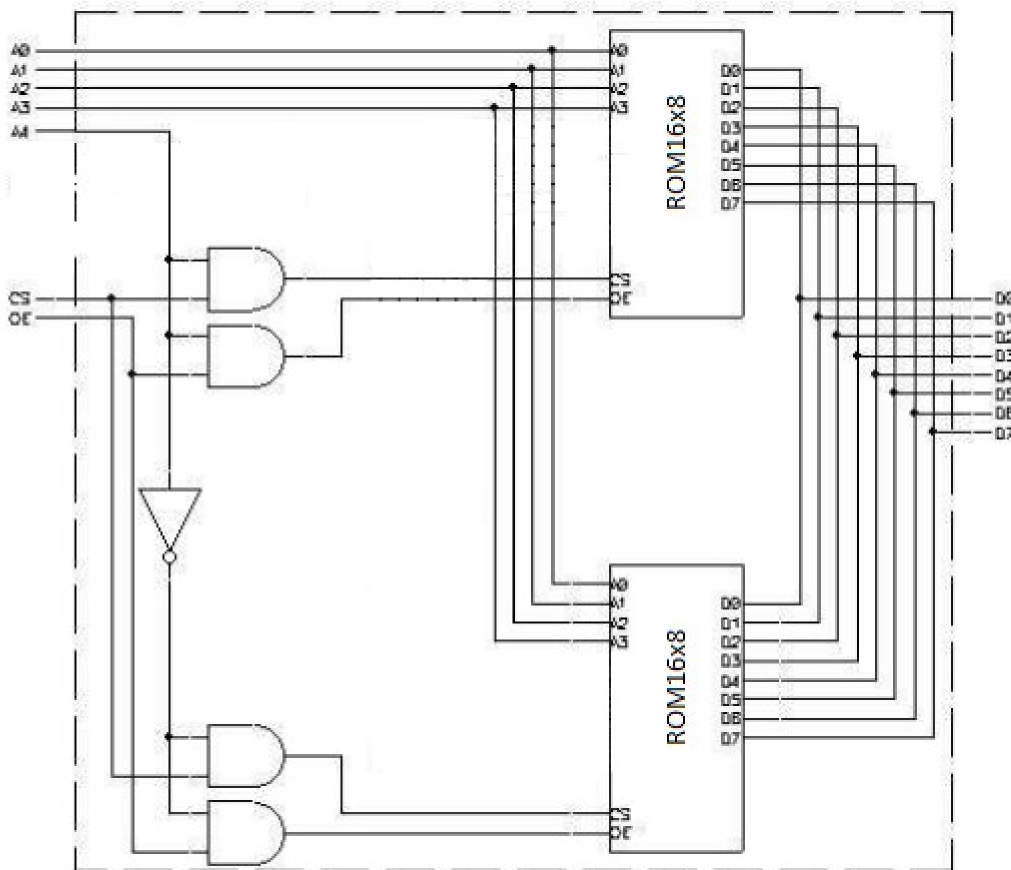
- Sabiendo que el valor de ES es 0x1030 y el de DI 0x22AA, ¿en qué posición de memoria física está guardado el entero?
- Especifique el valor de DS para que el acceso a DS:[0x1A] coincida con el entero guardado.
- Explique por qué no es posible especificar un valor para DS de forma tal que DS:[0x5] coincida con la dirección física del dato guardado.

Solución Preguntas:

Pregunta 1

Chip select: Si CS = 0 todas las salidas de la ROM están en 0, con independencia de las entradas de dirección y del "valor" almacenado en la "posición" de la ROM indicada por dicha dirección y si CS = 1 las salidas presentan el contenido de la ROM en la posición señalada por la dirección. Esto nos permite ahorrar en la implementación la estructura de ANDs que deberíamos colocar a la salida de las ROMs a la hora de trabajar con varias ROMs y elegir la salida de la ROM general.

Output enable: La entrada Output enable tiene como cometido habilitar o no la salidas de una ROM. Opera de la siguiente manera: cuando dicha entrada de control está en 0, la salida pasa al "estado de alta impedancia" y cuando la entrada de control está en 1, la salida está en estado lógico 0 ó 1. Esto nos permite utilizar compuertas ORs "cableadas" (sin necesidad de utilizar compuertas). Cuando OE = 0, sin importar el valor de CS la salida estará en el "tercer estado" o "estado de alta impedancia".



Página 11 y 12 de las notas del curso capítulo "Memorias ROM"

Pregunta 2

- Código de Hamming: SI
- Paridad: NO
- Entrelazado 2 de 5: NO
- Paridad horizontal+vertical: SI

Utilizando código de Hamming 1001 queda de la siguiente forma: 1001100.

Notas de teórico sobre Código y errores

Pregunta 3

a)
- **CPU**: se encarga de ejecutar los programas.
  Memoria: almacena el programa (conjunto de instrucciones ordenado lógicamente) y los datos (operadores y resultados de la ejecución de las instrucciones).
- **Entrada/Salida**: comunica el computador con el mundo exterior, permitiendo la interacción con los usuarios y con otras computadoras.

Notas de teórico sobre la arquitectura de Von Neumann página 2.

b)
Un ciclo de instrucción típico tiene 5 pasos característicos:
- **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 donde 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.

Notas de teórico sobre Organización página 3

Pregunta 4

La dirección segmentada 1030h:22AAh corresponde a la dirección efectiva 1030h*16 + 22AAh = 0x125AA

Para que el acceso a DS:[0x1A] coincida con el entero guardado, DS*16+0x1A = 0x125AA, por lo tanto DS = 0x1259

No es posible especificar un valor para DS de forma tal que DS:[0x5] coincida con la dirección física del dato guardado porque la dirección efectiva del dato está ubicada en la posición 10 dentro del parágrafo de memoria asignado, y la dirección segmentada dada se ubica en la posición 5 del parágrafo, independientemente del valor de DS.

Problema 1

Se desea construir un cuadricóptero, el *SantaCóptero,* con su mecanismo de nivelación basado en un sistema **no dedicado**.
El *SantaCóptero* posee cuatro motores que se encuentran en los extremos de sus brazos. Cada brazo posee un sensor de altura relativa, que permite indicar si está muy bajo o muy alto respecto al centro del *SantaCóptero*. Si está muy bajo, su motor deberá aumentar su velocidad y viceversa. Además, el *SantaCóptero* cuenta con un sensor de golpes que permite reaccionar rápido frente a cambios imprevistos.
El sistema cuenta con un reloj de 20Hz que interrumpe ejecutando la rutina **tiempo()**.
Los motores se controlan simultáneamente escribiendo en el puerto de E/S, de solo escritura de 16 bits, en la dirección **MOTORES**. Cada motor se controla con 4 bits, el motor 0 ocupa los bits menos significativos, el motor 1 los siguientes 4 bits, etc.
Los sensores de altura relativa ocupan 2 bits cada uno y están accesibles en el puerto de solo lectura de 8 bits en la dirección **ESTADO**. El sensor 0 ocupa los dos bits menos significativos, el sensor 1 los siguientes dos bits, etc. En cada sensor el bit más significativo indica si está fuera de posición y el menos indica si está más alto (1) o más bajo (0) de lo que debería.
Si está más alto se debe disminuir en uno la velocidad actual de ese motor, si está más bajo se debe aumentar en uno y si está en posición se debe dejar su velocidad actual. Debe tener en cuenta que si un motor ya está en su velocidad máxima, no debe incrementar su velocidad. Análogamente para la velocidad mínima.
En caso de dispararse el sensor de golpes, se ejecuta la rutina de interrupción **golpe()**. En este caso se deben poner al máximo o al mínimo por 500ms los motores cuyos sensores de altura relativa estén fuera de posición. Luego deben volver a su valor anterior. Si durante estos 500ms vuelve a dispararse la interrupción golpe(), ésta se debe ignorar.

Se pide: Implementar todas las rutinas del sistema necesarias para la **máquina no dedicada**.

Nota: la velocidad inicial para los motores es de 8.

Solución Problema 1

```
#define VMAX 0xF
#define VMIN 0x0
```

```
unsigned char velocidades[4], velocidades_backup[4];
bool estoyEnTilt;
int ticsTilt;

unsigned short [] MASK_SENSOR = {0x02, 0x08, 0x20, 0x80}
unsigned short [] MASK_ARRIBA = {0x01, 0x04, 0x10, 0x40}

void interrupt golpe(){
    ticsTilt = 0;
    if(!estoyEnTilt) {
        unsigned short sensores = in(ESTADO);
        unsigned short velocidades = 0;
        for(int i = 0; i < 4; i++) {
            velocidades_backup[i] = velocidades[i];
            if (sensores & MASK_SENSOR[i])
                velocidades[i]=(sensores & MASK_ARRIBA[i])?VMIN:VMAX;
            velocidades = velocidades | velocidades[i] << (4*i);
        } // fin del for
        out(MOTORES, velocidades);
        estoyEnTilt = true;
    }
}

void interrupt tiempo(){
    unsigned short  sensores = in(ESTADO);
    if(estoyEnTilt) {
        if (ticsTilt == 10){
            estoyEnTilt = false;
            for(int i = 0; i < 4; i++) velocidades[i] = velocidades_backup[i];
        } else
            ticsTilt++;
    } else {
        for(int i=0; i < 4; i++){
            if (sensores & MASK_SENSOR[i]) {
                if (sensores & MASK_ARRIBA[i] {
                    if(velocidades[i]!=0xF) velocidades[i]++;
                } else if(velocidades[i])
                    velocidades[i]--;
            }
        }
    }
    unsigned short velocidades = 0;
    for(int i=0; i < 4; i++) velocidades = velocidades | velocidades[i] << (4*i);
    out(MOTORES, velocidades);
}

int main(){
    disable();
    estoyEnTilt = false;
    for(int i=0; i < 4; i++) velocidades[i] = 8;
    // Instalar rutinas
    enable();
}
```

Problema 2

Se considera la siguiente estructura de árbol binario:

```
struct
{
     unsigned char hijoIzq;
     unsigned char hijoDer;
     unsigned short dato;
} nodo;
nodo arbol[256]
```

En donde hijoIzq e hijoDer son los índices a los subárboles izquierdo y derecho respectivamente para un nodo dado.
El valor **0** para un índice indica que no existe nodo asociado. El árbol tiene por lo menos un elemento y no hay nodos con **dato** repetido en el mismo.

Se pide:

a) Escribir en alto nivel un procedimiento recursivo que retorna el dato mayor que se encuentra en el árbol y su índice asociado en el arreglo.
El procedimiento tiene la siguiente firma:

```
void buscoMayor(unsigned char indice, out short mayor, out char indiceMayor)
```

Compilar la rutina en assembler 8086 sabiendo que la variable **arbol** se encuentra a partir de la dirección segmentada **ES:0000**. El parámetro se recibe en el stack y el resultado también se debe retornar por el stack. La rutina debe conservar todos los registros de propósito general.

b) Calcular el tamaño mínimo que debe tener el stack para que la función pueda ser ejecutada en todos los casos, cualquiera sea el tamaño y topología del árbol.

Solución

Parte a)

```
void buscoMayor(unsigned char indice, out short mayor, out char indiceMayor)
{
  unsigned short mayor1, mayor2;
  unsigned char indice1, indice2;
  if(indice == 0)
  {
    mayor1 = 0;
    indiceMayor1 = 0;
  }else
  {
    buscoMayor(arbol[indice].hijoIzq, mayor1, indice1);
    buscoMayor(arbol[indice].hijoDer, mayor2, indice2);
    if(mayor2 >= mayor1)
```

```
    {
      indice1 = indice2;
      mayor1 = mayor2;
    }
    if(arbol[indice].dato >= mayor1)
    {
      indice1 = indice;
      mayor1 = arbol[indice].dato;
    }
  }
  indice = indice1;
  mayor = mayor1;
}

buscoMayor proc
  sub SP, 2  ; Reservo espacio para parametros de retorno
  push BP
  mov BP, SP
  push DI
  push AX
  push BX
  push DX
  cmp word ptr [BP+6], 0  ; if (indice == 0)
  jne else
  xor DX, DX  ; mayor1 = 0
  xor DI, DI  ; indiceMayor1=0
  jmp fin
else:
  mov BX, [BP+6]
  shl BX, 1
  shl BX, 1   ; ES:[BX] apunta a base del nodo
  xor AH, AH
  mov AL, ES:[BX]
  push AX
  call buscoMayor  ; buscoMayor(arbol[indice].hijoIzq, mayor1, indice1);
  pop DI ; indice1
  pop DX ; mayor1
  mov AL, ES:[BX+1]
  push AX
  call buscoMayor  ; buscoMayor(arbol[indice].hijoDer, mayor2, indice2);
  cmp [BP-10], CX   ; if(mayor2 >= mayor1)
  jb menor
  pop DI
  pop DX
  jmp sigo
menor:
  add SP,4
sigo:
  cmp ES:[BX+2], DX
  jb fin
  mov DI,[BP+6]          ; indice1 = indice;
  mov DX, ES:[BX+2]
fin:
  mov AX, [BP+4]
  mov [BP+2], AX ; actualizo dir ret
```

```
  mov [BP+6], DX ; mayor
  mov [BP+4], DI ; indiceMayor
  pop DX
  pop BX
  pop AX
  pop DI
  pop BP
  ret
buscoMayor endproc
```

Parte b)

Analizándo el código vemos que la funcion discrimina dos casos (paso base, donde el índice es 0 y paso recursivo, donde el índice es distinto de 0).

El consumo para el paso base es de 2 bytes para el índice de entrada, 2 bytes para la dirección de retorno, 2 bytes para el espacio reservado a parámetros de retorno y 10 bytes para guardar el contexto (BP, DI, AX, BX, DX).
Por lo tanto, **C(0) = 16 bytes**
Para las llamadas recursivas vemos que el consumo es el mismo pues luego de cada llamada recursiva se quitan los parametros de retorno del stack.

Considerando que el árbol más grande que se puede construir tiene 255 nodos, éste requerirá el consumo máximo de stack cuando se completen los 255 nodos y el mismo degenere en una lista.

El paso recursivo, considerando N como la altura del árbol descripto tiene como consumo: 2 bytes para el índice de entrada, 2 bytes para la dirección de retorno, 2 bytes para el espacio reservado a parámetros de retorno, 10 bytes para guardar el contexto (BP, DI, AX, BX, DX) .

Por lo tanto, $C(N) = 16 + C(N-1)$ para $N > 0$ ==> $\mathbf{C(N) = 16 * (N+1)}$

==> Tamaño minimo de stack = C(255) = 16 * 256 = 4096 bytes