Examen – 17 de Febrero de 2012

Instrucciones

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.**
- **Escriba las hojas de un solo lado.**
- **Empiece cada ejercicio en una hoja nueva.**
- **Sólo se contestarán dudas de letra. No se aceptarán dudas los últimos 30 minutos del examen.**
- **Duración: 3 horas.**
- **Requisito para Aprobación: responder al menos la mitad del problema teórico (2 de las 4 preguntas) y resolver un problema de práctico de forma correcta**

Problema 1

Pregunta 1)

a) Con representaciones de 4 bits, escriba el número -6 en: valor absoluto y signo, complemento a 1, complemento a 2 y desplazamiento (d = 7).
b) Para las representaciones indicadas en la parte anterior, escriba el número -8 si es representable o justifique porqué no lo es.

Pregunta 2)

a) Indique la diferencia entre una memoria ROM y una RAM.
b) Utilizando una memoria ROM, se desea implementar el cálculo de código de Hamming de 4 bits de datos. Determinar el tamaño y la organización de la ROM , y construir a partir de memorias ROM de 16 x 1.

Pregunta 3)

a) Describa el principio de localidad, y de que forma se aprovecha en la jerarquía de memoria.
b) ¿De qué forma se logra que el espacio de direccionamiento virtual sea mayor a espacio de direccionamiento físico? Justifique su respuesta.

Pregunta 4)

a) Comente las diferencias entre microinstrucciones verticales y horizontales.
b) Implemente en lenguaje MAL el ciclo de ejecución de la instrucción POP, que retira del stack una palabra y la almacena en el registro acumulador.

Soluciones:

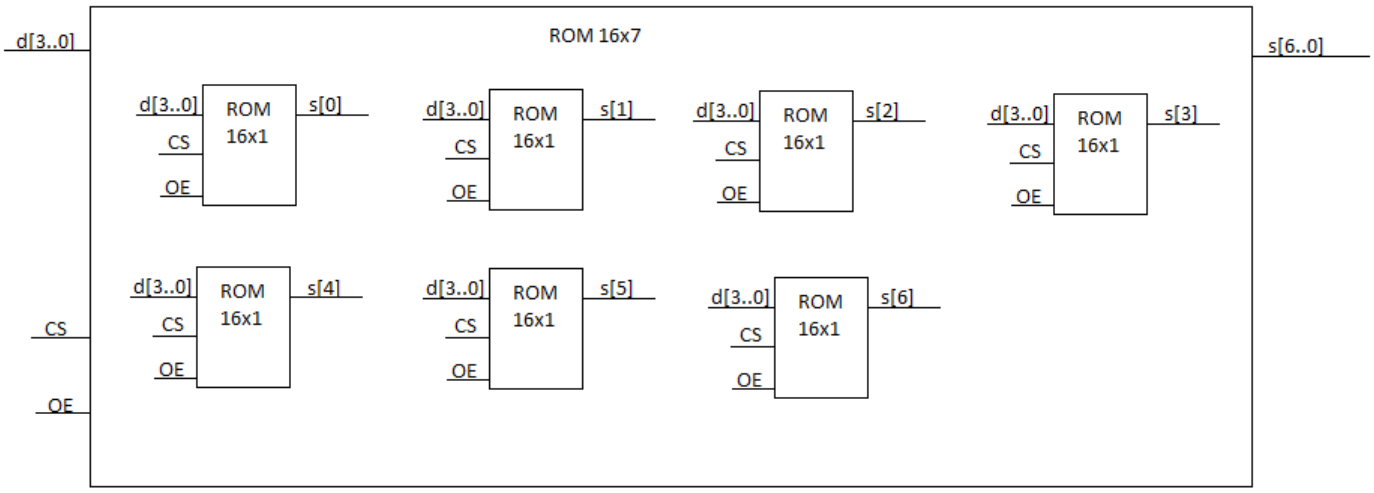
ATENCIÓN: LAS SOLUCIONES PUBLICADAS DE LAS PREGUNTAS TEÓRICAS SON LAS MÍNIMAS ACEPTABLES.

Pregunta 1:
Bit de signo: 1110
Complemento a 1: 1001
Complemento a 2: 1010
Desplazamiento: 0001

Con bit de signo no podemos representar -8, ya que su rango representable es -7 a 7.
Con complemento a 1 tampoco podemos representar -8, ya que el rango representable es -7 a 7.
Con complemento a 2 si podemos representarlo. Es 1000.
Con desplazamiento con d=7 no podemos representarlo ya que el menor número representable en desplazamiento es -d, o sea, -7.

Pregunta 2:

a) La diferencia principal entre una memoria ROM (Read Only Memory) y una memoria RAM (Random Access Memory) es que la ROM es una memoria de solo lectura, mientras que la RAM permite tanto leer como escribir en ella.
b) El código Hamming de 4 bits de datos contiene 3 bits extra de redundancia, por lo que se precisa una memoria de 16x7 (14 bytes). Las entradas corresponden a los 4 bits de datos ($d_1d_2d_3d_4$) y la salida corresponde a la palabra de codigo asociada ($p_1p_2d_1p_3d_2d_3d_4$)



Pregunta 3:
a) El principio de localidad establece que los programas acceden a una porción relativamente reducida del espacio de direcciones en un determinado lapso de tiempo.
Se aprovecha en la jerarquía de memoria manteniendo los datos “recientemente accedidos” en las jerarquías altas, más cerca de la CPU (aprovechando la localidad temporal), y moviendo bloques de memoria contiguos hacia las jerarquías más altas (aprovechando la localidad espacial).

b) Se logra colocando parte de la memoria virtual en almacenamiento secundario (disco) y gestionando cual parte de la memoria virtual se mapea al espacio de direccionamiento físico en determinado momento, logrando así que el espacio de direccionamiento virtual sea mayor que el espacio de direccionamiento fisico.

Pregunta 4:
a) En las microinstrucciones horizontales, el valor de las líneas de control de la UC no está codificado, mientras que en microinstrucciones verticales si. Ej: El valor de las líneas de la ALU está codificado con 2 bits para la arquitectura MIC-1. Sin embargo, las líneas de WR y RD no.
b)

```
mar := sp; sp := sp +1: rd
rd
ac := mbr
```

Problema 2

Se desea implementar un programa en Intel 8086 que calcule la suma de todos los elementos de una matriz cuadrada de 4x4 enteros con signo. A dicho valor le llamaremos el *peso.*
Este programa debe recorrer un arreglo de matrices, calcular para cada una el peso y guardar los resultados en otro arreglo.
Una matriz de 4x4 enteros se dispone en memoria por filas, quedando las 4 filas de forma consecutiva.
Para la implementación, se disponen de las siguientes constantes:

`MATRICES`: Indica el desplazamiento dentro del ES donde comienza el arreglo de matrices.
`PESOS_MATRICES`: Indica el desplazamiento dentro del DS donde comienza el arreglo con los pesos de las matrices.
`CANT_MATRICES`: Indica la cantidad de matrices en el arreglo que comienza en `MATRICES` (es el mismo valor para `PESOS_MATRICES`).

El programa debe ir matriz a matriz y sumar todos los elementos:

```
short pesosMatrices[CANT_MATRICES];
Matriz matrices [CANT_MATRICES ];
void sumaMatrices(){
     for(int i = 0; i < CANT_MATRICES; i++){
          pesosMatrices[i] = calcularPesoMatriz(matrices[i]);
     }
}
```

El sistema cuenta con una memoria caché de correspondencia directa que se utilizará únicamente en la lectura de las las matrices. La memoria es de 8kB distribuida en 1024 líneas.

Se pide:
a) Implemente dicho programa en assembler 8086.
b) Suponga que MATRICES vale 0x0000. ¿Cuál es la tasa de aciertos en la lectura de los elementos de la matriz?

Solución Problema 2

Parte A

```
proc sumaMatrices
      xor DX, DX // DX = 0
inicioForArreglo:
      cmp DX, CANT_MATRICES
      je fin
      mov BX, DX
      mov CL, 5
      shl BX, CL
      xor AX, AX // AX = 0, AX acumulador
      xor CX, CX // CX = 0, CX contador
      add BX, MATRICES
inicioForMatriz:
      cmp CX, 16
      je finForMatriz
      mov SI, CX
      shl SI, 1
      add AX, ES:[BX+SI]
      inc CX
      jmp inicioForMatriz
finForMatriz:
      mov BX, PESOS_MATRICES
      mov SI, DX
      shl SI, 1
      mov [BX+SI], AX
      inc DX
      jmp inicioForArreglo
fin:
      ret
```

Parte B

Con esa distribución de caché, tenemos 8 bytes por línea y la estructura quedaría:

```
TAG 7 bits | LINEA 10 bits | BYTE 3 bits
```

La cantidad de bytes en una línea de caché coincide con la cantidad de bytes que ocupa una fila.
Además, como MATRICES es 0x0000 y el valor del registro de segmento no altera el desplazamiento dentro del bloque de caché (los bits correspondientes a BYTE) todas las filas están alineadas con los bloques.
Por lo tanto, al leer un elemento de una fila, el resto ya queda cargada. Como recorremos por fila, sus cuatro elementos se leen de forma consecutiva. De forma que de las cuatro lecturas, 3 son cache hits. Por lo tanto la tasa de aciertos es ¾.

Problema 3

La empresa **EXECUTISEAT** ha diseñado un nuevo sillón reclinable para el sector "*Business Class"* de los aviones Boeing 777. Para implementar el comando del sillón, que permite accionar los motores para mover las piezas del sillón hasta adoptar la posición deseada, se utilizará un microprocesador dedicado.

El sillón consta básicamente de 3 partes móviles: respaldo, asiento y apoya-pies. Cada parte tiene asociado un motor que permite su movimiento.

El pasajero controla la posición del sillón mediante un teclado especial el cual dispone de cuatro teclas:

- "Relax": mientras se la mantenga apretada lleva el sillón a una posición reclinada
- "Bed": mientras se la mantenga apretada lleva el sillón a un posición prácticamente horizontal
- "Up": al pulsarlo lleva el sillón a la posición "normal" (obligatoria en despegue y aterrizaje). El reacomodo de las partes móviles lleva un cierto tiempo, pero en este caso no es necesario mantener apretada la tecla.
- "Stop": detiene el proceso iniciado por el botón "Up", en caso que aun no se haya completado.

Cada una de las tres configuraciones estándar del sillón ("Relax", "Bed", "Up") se corresponde con posiciones específicas de cada parte móvil. Los motores se pueden accionar en un sentido ú otro en función de si la posición deseada de la parte móvil está antes o después de la actual.

Cada vez que se apreta o suelta una tecla se produce una interrupción que invoca a la reunión **tecla()**. Al ocurrir la interrupción el estado de las teclas (0 = reposo; 1 = apretada) queda accesible en los bits 3, 2, 1 y 0 del byte de E/S de solo lectura en la dirección TECLAS (Relax, Bed, Up y Stop respectivamente, siendo el bit 0 el menos significativo).

Las posiciones actuales de las partes móviles son detectadas por respectivos sensores que codifican cada valor en 4 bits y los deja accesibles en la palabra (16 bits) de E/S en la dirección SENSORES (los bits 11 a 8: respaldo; bits 7 a 4: asiento; bits 3 a 0: apoya-pies).

Las posiciones de las configuraciones estándar son las siguientes:

	Respaldo	Asiento	Apoya-pies
Relax	9	4	9
Bed	15	15	15
Up	0	0	0

Los motores se accionan con las señales: **m** (0 = motor apagado; 1 = motor encendido) y **s** (0 = hacia posiciones decrecientes; 1 = hacia posiciones crecientes) accesibles en el byte de E/S de solo escritura en la dirección MOTORES (bits 5 y 4: respaldo; bits 3 y 2: asiento; bits 1 y 0: apoya-pies). Los motores son paso a paso y mueven un paso cada vez que se escribe un valor nuevo en la pareja de bits correspondiente. El motor demora cierto tiempo en ejecutar un paso. Durante ese tiempo no puede recibir un nuevo comando de paso. Al terminar el movimiento de un paso, se produce una interrupción que invoca a la rutina **finPaso()** quedando en el byte de E/S en la dirección PASO_MOTOR la indicación cual motor produjo la interrupcion (bit 2: respaldo; bit 1: asiento; bit 0: apoya_pies).

Se pide:

Programar en un lenguaje de alto nivel (preferentemente C) todas las rutinas necesarias para implementar el controlador del sillón motorizado descripto.

Notas: si se apreta una nueva tecla mientras está una apretada, se puede ignorar la segunda. Si se aprieta Relax ó Bed se cancela la eventual operación Up que esté pendiente y prevalece la nueva tecla.

Solución Problema 3

```
#define     TRUE        0x01
#define     FALSE       0x00
#define     RELAX       0x08
#define     BED         0x04
#define     UP          0x02
#define     STOP        0x01
#define     RESPALDO    0x04
#define     ASIENTO     0x02
#define     APOYAPIES   0x01

unsigned char hayTecla, teclaUp, teclaStop. teclaRelax, teclaBed, pasoRespaldo, pa-
soAsiento, pasoApoyaPies;

void interrupt tecla() {
      unsigned char leoTecla;

      leoTecla = in(TECLAS);
      if (hayTecla && ((leoTecla && 0x0F) == 0x00)) {
         hayTecla = FALSE;
         teclaRelax = FALSE;
         teclaBed = FALSE;
         teclaUp = FALSE;
         teclaStop = FALSE;
      }
      else {
         if (!(hayTecla)) {
            hayTecla = TRUE;
            if ((leoTecla && RELAX) == RELAX) teclaRelax = TRUE;
            else  if ((leoTecla && BED) == BED) teclaBed = TRUE;
               else  if ((leoTecla && UP) == UP) teclaUp = TRUE;
                  else  teclaStop = TRUE;
         }
      }
}

void interrupt finPaso() {
      unsigned char leoPaso;

leoPaso = in(PASO_MOTOR);
      if ((leoPaso && RESPALDO) == RESPALDO) pasoRespaldo = TRUE;
      if ((leoPaso && ASIENTO) == ASIENTO) pasoAsiento = TRUE;
      if ((leoPaso && APOYAPIES) == RESPALDO) pasoApoyPies = TRUE;
}
```

```
void main() {
   unsigned char leoSensores, upActivo;

      hayTecla = FALSE;
      teclaRelax = FALSE;
      teclaBed = FALSE;
      teclaUp = FALSE;
      teclaStop = FALSE;
      pasoRespaldo = TRUE;
      pasoAsiento = TRUE;
      pasoApoyaPies = TRUE;
      upActivo = FALSE;
      out(MOTORES, 0x00);
      // instalo rutinas de interrupción
      enable();
      while (TRUE) {
         if (hayTecla) {
            if (teclaRelax) {
               upActivo = FALSE;
               while (hayTecla) {
                  leoSensores = in(SENSORES);
                  if ((leoSensores && 0x0F00) != 0x0900) {
                    if (pasoRespaldo) {
                       pasoRespaldo = FALSE;
                       if ((leoSensores && 0x0F00) > 0x0900)
                          out(MOTORES, 0x20);
                       else
                          out(MOTORES, 0x30);
                    }
                  }
                  if ((leoSensores && 0x00F0) != 0x0040) {
                    if (pasoAsiento) {
                       pasoAsiento = FALSE;
                       if ((leoSensores && 0x00F0) > 0x0040)
                          out(MOTORES, 0x08);
                       else
                          out(MOTORES, 0x0C);
                    }
                  }
                  if ((leoSensores && 0x000F) != 0x0009) {
                    if (pasoApoyaPies) {
                       pasoApoyaPies = FALSE;
                       if ((leoSensores && 0x000F) > 0x0900)
                          out(MOTORES, 0x02);
                       else
                          out(MOTORES, 0x03);
                    }
                  }
               }
            }
            if (teclaBed) {
               upActivo = FALSE;
               while (hayTecla) {
                  leoSensores = in(SENSORES);
                  if ((leoSensores && 0x0F00) != 0x0F00) {
                    if (pasoRespaldo) {
                       pasoRespaldo = FALSE;
```

```
                    out(MOTORES, 0x30);
                  }
                }
                if ((leoSensores && 0x00F0) != 0x00F0) {
                  if (pasoAsiento) {
                    pasoAsiento = FALSE;
                    out(MOTORES, 0x0C);
                  }
                }
                if ((leoSensores && 0x000F) != 0x000F) {
                  if (pasoApoyaPies) {
                    pasoApoyaPies = FALSE;
                    out(MOTORES, 0x03);
                  }
                }
              }
            }
            if (teclaUP) upActivo = TRUE;
            if (teclaSTOP) upActivo = FALSE;
          }
          if (upActivo) {
             leoSensores = in(SENSORES);
             if ((leoSensores && 0x0F00) != 0x0000) {
                 if (pasoRespaldo) {
                      pasoRespaldo = FALSE;
                      out(MOTORES, 0x20);
                 }
             }
             if ((leoSensores && 0x00F0) != 0x0000) {
                 if (pasoAsiento) {
                      pasoAsiento = FALSE;
                     out(MOTORES, 0x08);
                }
             }
             if ((leoSensores && 0x000F) != 0x0000) {
                  if (pasoApoyaPies) {
                      pasoApoyaPies = FALSE;
                     out(MOTORES, 0x02);
                  }
             }
          }
      }
}
```