

P1. (3 puntos) A partir de las señales que se especifican, diseñar la conexión de un registro de comando y otro de estado, ambos en la dirección 0FCH del mapa de E/S. Dibujar un diagrama de tiempos donde se detalle el momento de la escritura y la lectura en los registros. Utilizar registros LS374 y decodificador LS138.

Señales:

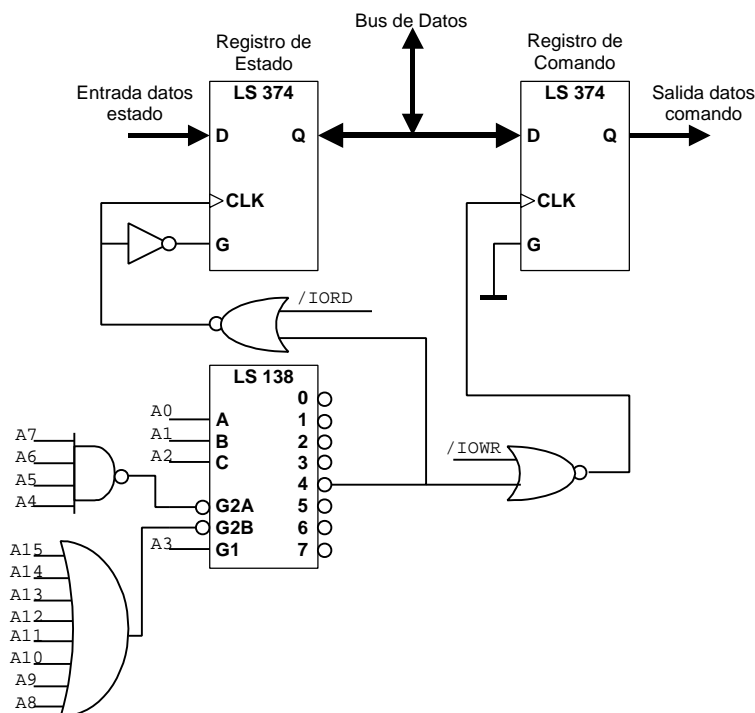
/IOWR: Escritura en E/S
 /IORD: Lectura en E/S
 A15..A0: Líneas 15 a 0 del bus de direcciones.
 D7..D0: Líneas del bus de datos.

SOLUCIÓN:

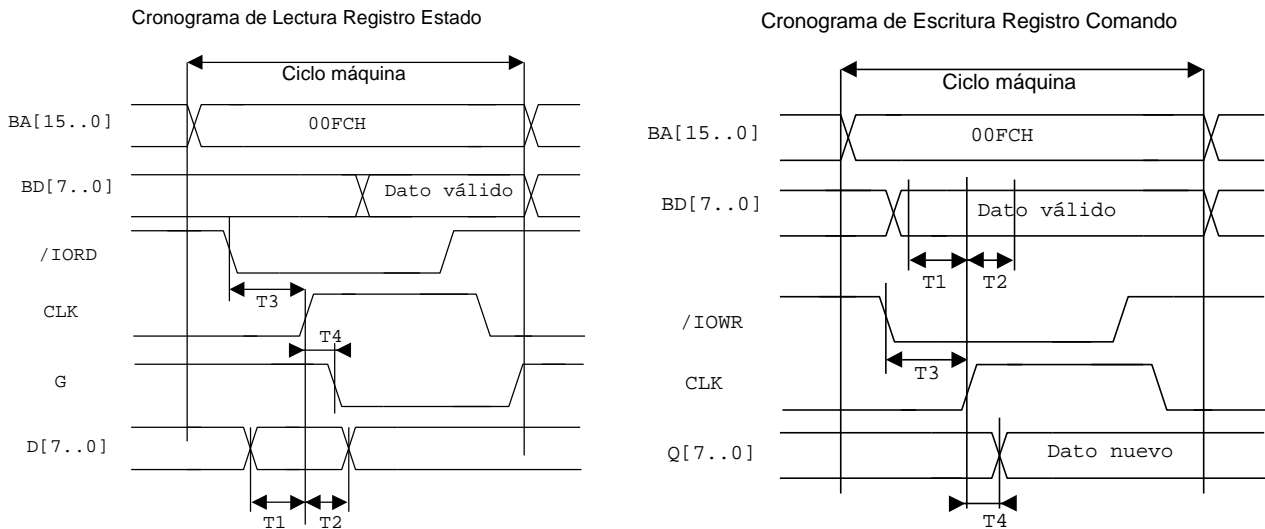
Debemos generar las señales CK y G de cada registro de ocho bits LS374. La señal CK es activa por flanco ascendente, luego los datos se guardarán en ese momento.

El registro de comando será un registro de solo escritura y el de estado de solo lectura y ambos residirán en la misma posición de E/S.

La decodificación se hará para la dirección de 16 bits 00FCH, o sea A15..A8 deben estar todos a 0, A7..A2 todos a 1 y A1 y A0 ambos a 0 lógico. Si empleamos puertas lógicas y el decodificador LS138, el esquema lógico quedaría de la siguiente forma:



Vemos que la señal de reloj y de enable para el registro de estado se extraen de la misma señal, de esta forma teniendo en cuenta los retardos inherentes a las puertas lógicas, se generarán los siguientes cronogramas :



En el cronograma de lectura, los tiempos son:

- T1. Tiempo de setup del registro.
- T2. Tiempo de hold del registro
- T3. Retraso en el decodificador y puerta LS32.
- T4. Retraso en el inversor LS04.

En el cronograma de escritura, los tiempos son:

- T1. Tiempo de setup del registro.
- T2. Tiempo de hold del registro.
- T3. Retraso en el decodificador de direcciones.
- T4. Retraso de habilitación del triestado en el LS374.

P2. (4 puntos) Un sistema basado en un 8086 dispone de un bus de control que contiene todas las líneas de control del procesador en modo mínimo. La frecuencia del reloj es de 5MHz.

La memoria del sistema NO OCUPA todo el mapa de memoria disponible y se supone compuesta por 32K de ROM situada en la parte alta del mapa de memoria y de 128K de SRAM comenzando por la parte baja.

La E/S del sistema se compone de una USART 82050 para el teclado y otra para un canal serie que funciona a 9600 bps con formato asíncrono de 8 bits, sin paridad y 1.5 bits de parada.

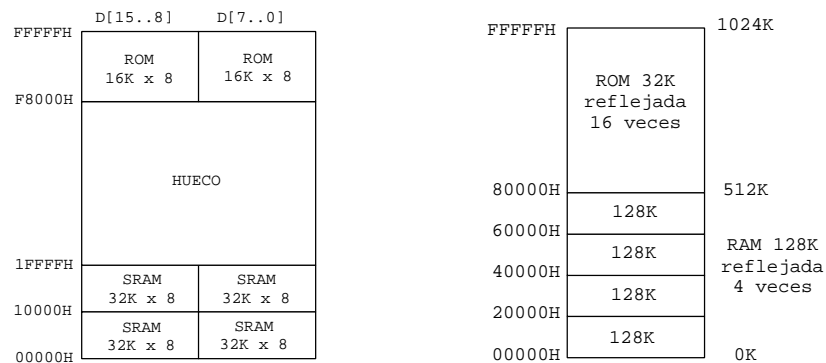
Calcular:

- a) El decodificador necesario para implementar la UCM por medio de memorias RAM de 32Kx8 y ROM de 16Kx8.
- b) El decodificador necesario para situar las dos 82050 en direcciones consecutivas a partir de 0ACH en el mapa de E/S.
- c) Tiempo máximo permitido al software para que la 82050 del canal serie pueda trabajar por interrupciones.

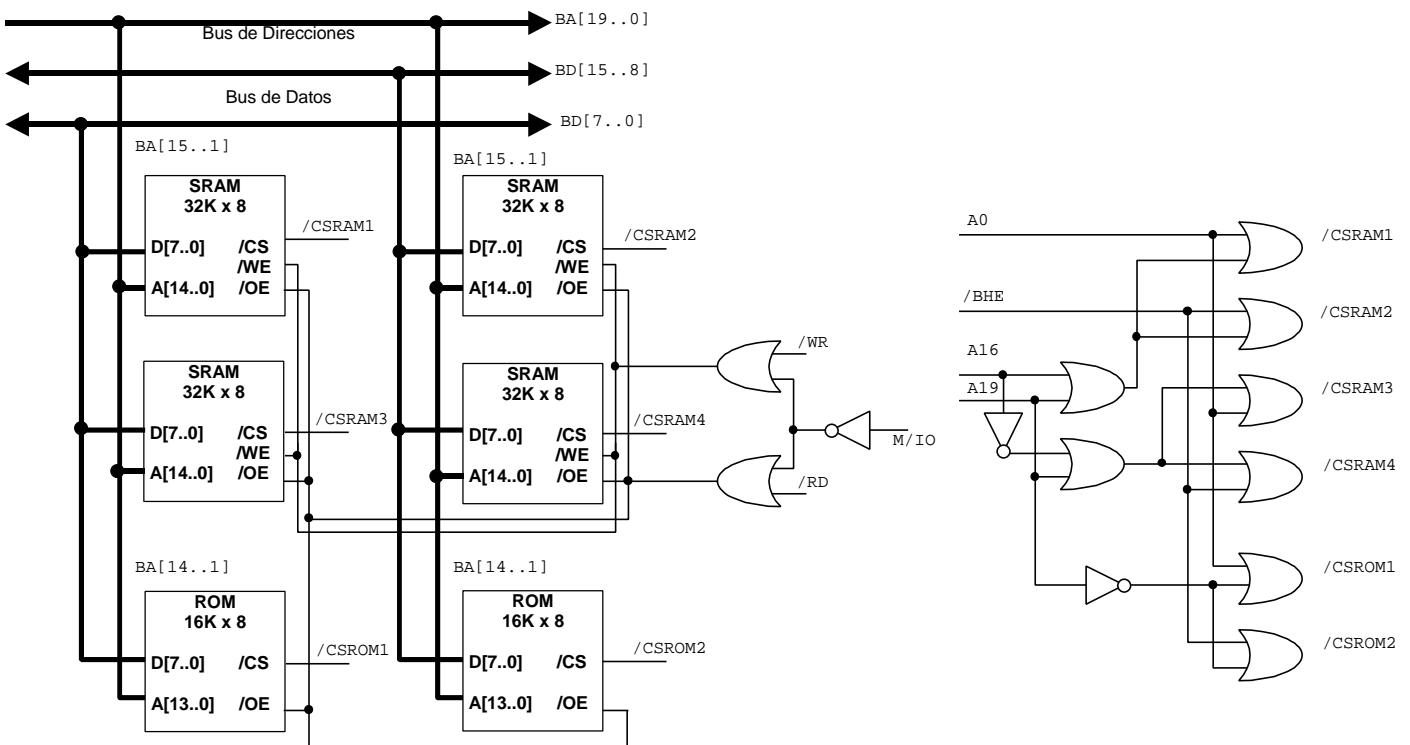
d) Velocidad de la comunicación en caracteres por segundo.

SOLUCIÓN:

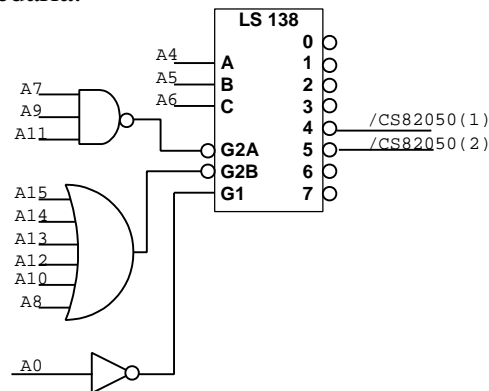
- a) Para entender mejor el problema seria conveniente realizar el mapa de memoria del sistema. Dado que el sistema tiene un hueco de zona de memoria sin utilizar, para realizar el decodificador no es necesario decodificar todas las líneas de direcciones, siempre y cuando no se piense en acceder a direcciones del hueco. Como contrapartida, la RAM y ROM aparecerán reflejadas en otras direcciones del mapa de memoria, con lo cual estas direcciones del hueco no se podrán usar para ampliar el sistema:



En el caso de la RAM el bit que cambia entre bancos es el A16. Se puede usar el bit de mayor peso A19 para distinguir entre RAM y ROM. De esta forma la decodificación queda muy sencilla. El diagrama lógico del decodificador y de los circuitos necesarios para generar todas las señales de las memorias queda:



- b) La forma más sencilla de resolverlo sería conectar ambos controladores al bus de datos bajo y situar las direcciones consecutivas pares, esto consumiría el doble de direcciones necesarias para los puertos, en total 32 bytes, desde 0AC0H a 0AE0H. El circuito lógico quedaría:



- c) El tiempo máximo permitido al software será el mínimo transcurrido entre recepciones de datos consecutivas. Esto es, a 1200 baudios y un tamaño de recepción de 8+1.5+1 bits :

$$\frac{10.5 \text{ bits}}{9600 \text{ bits/segundo}} \approx 1.09 \text{ ms}$$

- d) La velocidad de comunicación en caracteres por segundo será:

$$\frac{9600 \text{ bits/segundo}}{10.5 \text{ bits/caracter}} \approx 914.28 \text{ carac./seg}$$

P3. (3 puntos) Hacer un programa que utilice una rutina. La especificación de la rutina es la siguiente:

- Debe comparar dos tablas de datos (cadenas de caracteres) para comprobar si son iguales.
- La rutina utilizará dos parámetros, la longitud de la cadena y la dirección de una tabla donde se encuentran las direcciones de las dos cadenas a comparar.
- El primer parámetro deberá ser pasado en el acumulador (AX). El segundo parámetro deberá ser pasado en el registro DI.
- Si las dos cadenas son iguales deberá la rutina indicarlo devolviendo el bit de carry a 0 (usar CLC), caso contrario a 1 (usar STC).
- La rutina no debe modificar ningún registro, es decir, los contenidos ANTES de llamar a la rutina y DESPUES de retornar de ésta TIENEN que ser los mismos (excepto el IP y Flags).

La especificación del programa principal es:

- Debe inicializar lo necesario para funcionar con rutinas.
- Debe declarar las cadenas que se comparan y debe hacer como mínimo dos llamadas a la rutina anterior de forma adecuada, es decir, debe preparar los parámetros necesarios y debe evaluar el resultado aportado por la rutina (verificar el estado del bit de carry).
- Si las cadenas son iguales sacar un mensaje por pantalla con la rutina S_OUT, que toma como parámetro en AX la dirección de la cadena de caracteres que se va a mostrar por pantalla.

- El programa debe acabar con la llamada a la rutina FINALIZAR que se supone ya definida.

Hay que entregar el programa completo en lenguaje ensamblador del 8086, empleando las directivas adecuadas del compilador.

SOLUCIÓN:

El programa completo es el siguiente:

```
.model small
.stack 100H
.data
CAD1      DB      "Hola"
CAD2      DB      "Hola"
LONG      DB      4
TABLA1    DW      CAD1,CAD2
TABLA2    DW      CAD2,CAD1
MENS1     DB      "Cadenas son iguales.$"
MENS2     DB      "Cadenas son diferentes.$"

.code

;-----
;  Programa principal. Hace dos llamadas a COMPARA con dos cadenas
;  Entrada:      no tiene
;  Salida:       no tiene
;  Modifica:     Todos
;-----

                MOV     AX,SEG TABLA1
                MOV     DS,AX
                MOV     DI,OFFSET TABLA1
                MOV     AL,LONG
                CALL    COMPARA
                JC      N_IG1
                MOV     AX,OFFSET MENS1
                CALL    S_OUT
                JMP     IG1
N_IG1:          MOV     AX,OFFSET MENS2
                CALL    S_OUT
IG1:            MOV     DI,OFFSET TABLA2
                CALL    COMPARA
                JC      N_IG2
                MOV     AX,OFFSET MENS1
                CALL    S_OUT
                JMP     IG2
N_IG2:          MOV     AX,OFFSET MENS2
                CALL    S_OUT
IG2:            CALL    FINALIZA
```

```

;-----
; Esta rutina compara dos cadenas
; Entrada:  Dirección de punteros de cadenas en DI
;          Tamaño de las cadenas en AX
; Salida:   Si el bit de acarreo esta a cero ambas son iguales
; Modifica: Nada
;-----

COMPARA    PROC
            PUSH    AX
            PUSH    BX
            PUSH    CX
            PUSH    SI
            PUSH    DI                ; Guardamos registros a usar
            MOV     BX,WORD PTR [DI]  ; Puntero a cadena 1
            MOV     CX,WORD PTR [DI+1] ; Puntero a cadena 2
            MOV     SI,BX             ; ptr a cad1 en SI
            MOV     DI,CX             ; ptr a cad2 en DI
            MOV     CX,AX             ; Inicializamos contador
SIG:        MOV     AL,BYTE PTR [SI]
            MOV     AH,BYTE PTR [DI]
            CMP     AL,AH             ; Comparamos bytes de cadenas
            JNE     NO_IGUALES        ; Si no iguales finalizamos
            LOOP    SIG
            CLC                       ; Las cadenas son iguales
            JMP     SALIDA
NO_IGUALES: STC                       ; Las cadenas son diferentes
SALIDA:     POP     DI
            POP     SI
            POP     CX
            POP     BX
            POP     AX
            RET
COMPARA    ENDP
END

```