SISTEMAS BASADOS EN MICROPROCESADORES

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Escuela Politécnica Superior – UAM

COLECCIÓN DE PROBLEMAS DE LOS TEMAS 5.5 A 6.5

P1. Escribir una rutina de ensamblador que se quede **bloqueada en un bucle hasta que el bit de petición de interrupción (IRQF) del RTC se ponga a uno**. Se valorará la eficiencia del código.

```
Esperar_IRQF PROC FAR

   push ax

   itera:  mov al, 0Ch
           out 70h, al          ; Pide leer registro C del RTC
           in al, 71h           ; Lee el registro C del RTC
           test al, 10000000b   ; IRQF es bit de más peso
           jz itera             ; Itera mientras IRQF sea cero
   pop ax
   ret

Esperar_IRQF ENDP
```

P2. Escribir una rutina de ensamblador que **lea en el registro DX la dirección base del puerto paralelo LPT2**. Se valorará la eficiencia del código.

```
Leer_LPT2 PROC FAR

   push es

   xor dx, dx
   mov es, dx
   mov dx, es:[040Ah]      ; Lee la dirección base de LPT2 en la BIOS

   pop es
   ret

Leer_LPT2 ENDP
```

P3. Usando la BIOS (se adjunta un extracto de la documentación de Ralph Brown), escribir en ensamblador el código de la función de C **`char Keystroke()`**, que **retorna un 1 si se ha pulsado alguna tecla o un 0 si no se ha pulsado**. Se valorará la eficiencia del código.

```
_Keystroke PROC FAR

   mov ah, 1
   int 16h
```

```
        jnz retorna1

        ; No hay tecla
        mov ax, 0
        ret

retorna1:  ; Hay tecla
           mov ax, 1
           ret

    _Keystroke ENDP
```

KEYBOARD - CHECK FOR KEYSTROKE

```
AH = 01h

Return:
ZF set if no keystroke available
ZF clear if keystroke available
AH = BIOS scan code
AL = ASCII character
```

Category: Bios - Int 16h - K

P4. Escribir en ensamblador el código que sea necesario para ejecutar un EOI en los manejadores de las siguientes interrupciones:

76h (Disco Duro)	*02h (NMI)*	*0Ch (COM1)*

`mov al, 20h`
`out 20h, al`
`out A0h, al`
	EOI innecesario	`mov al, 20h`
		`out 20h, al`

P5. Escribir una subrutina de ensamblador que **programe la oscilación del RTC a 512 interrupciones por segundo**. Se valorará la eficiencia del código.

```
    RTC_512 PROC FAR

       push ax
       mov al, 0Ah
       out 70h, al
       mov al, 00100111b      ; DV = 010 (32768 Hz), RS = 0111 (512 Hz)
       out 71h, al
       pop ax
       ret

    RTC_512 ENDP
```

P6. Usando la BIOS (se adjunta un extracto de la documentación de Ralph Brown), escribir en ensamblador el código de la función de C **`void Imprimir_Letra( char ascii )`**, que **imprime en la posición actual del cursor la letra de código ASCII indicado, usando color rojo intenso sobre fondo verde parpadeante**. La controladora de vídeo está configurada en modo CGA de texto. Todas las direcciones son cercanas. Se valorará la eficiencia del código.

```
_Imprimir_Letra PROC NEAR

   push bp
   mov bp, sp
   push ax
   push bx
   push cx

   mov al, 4[bp]
   mov bl, 10101100b
```

VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION

```
AH = 09h
AL = character to display
BH = page number
BL = attribute (text mode)
CX = number of times to write character

Return:
Nothing
```

Category: Video - Int 10h - V

```
   xor bh, bh
   mov cx, 1

   mov ah, 09h
   int 10h

   pop cx
   pop bx
   pop ax
   pop bp
   ret

_Imprimir_Letra ENDP
```

P7. Implementar en ensamblador de 8086 la subrutina lejana `imprimir_ASCIIZ`, que ha de imprimir por el puerto paralelo LPT1 una cadena ASCIIZ cuya dirección recibe en el registro BX. La impresión debe realizarse siguiendo el protocolo BUSY. Se valorará la eficiencia del código.

```
imprimir_ASCIIZ PROC FAR

   ; Imprime por el LPT1 con protocolo BUSY la
   ; cadena ASCIIZ cuya dirección recibe en BX.

   push ax bx dx es

   mov ax, 0
   mov es, ax
   mov dx, es:[0408h]          ; Lee en dx la dirección base del LPT1

continuar:
   mov al, [bx]
   cmp al, 0
   jz final                    ; Acaba por haber llegado a final de cadena

   call espera_libre           ; Espera a que impresora no esté ocupada
   call imprimir_caracter      ; Imprime carácter contenido en al

   inc bx
   jmp continuar

final:
   pop es dx bx ax
   ret

imprimir_ASCIIZ ENDP


espera_libre PROC FAR
   ; Espera activa mientras señal #BUSY = 0. Recibe
   ; en DX la dirección base del puerto paralelo

   push ax dx

   inc dx          ; Obtiene dirección del registro de estado

ocupada:
```

```
   in al, dx            ; Lee registro de estado
   test al, 10000000b   ; Comprueba bit 7 (#BUSY)
   jz ocupada           ; Impresora ocupada si #BUSY = 0

   ; Impresora ya está libre

   pop dx ax
   ret

espera_libre ENDP

espera_ocupada PROC FAR
   ; Espera activa mientras señal #BUSY = 1. Recibe
   ; en DX la dirección base del puerto paralelo

   push ax dx

   inc dx         ; Obtiene dirección del registro de estado

libre:
   in al, dx            ; Lee registro de estado
   test al, 10000000b   ; Comprueba bit 7 (#BUSY)
   jnz libre            ; Impresora libre si #BUSY = 1

   ; Impresora ya está ocupada

   pop dx ax
   ret

espera_ocupada ENDP



imprimir_caracter PROC FAR
   ; Recibe en AL el código ASCII del carácter que
   ; se debe enviar y en DX la dirección base del
   ; puerto paralelo. Debe generar un
   ; pulso positivo de STROBE.

   push ax dx

   out dx, al     ; Envía carácter a registro de datos

   ; Genera pulso positivo de STROBE

   add dx, 2      ; Obtiene dirección de registro de control
   in al, dx      ; Lee registro de control

   or al, 00000001b
   out dx, al     ; STROBE = 1

   sub dx, 2      ; Obtiene dirección base
   call espera_ocupada  ; Espera activa hasta que la impresora pasa
                        ; a estar ocupada.
   add dx, 2      ; Obtiene dirección de registro de control

   and al, 11111110b
```

```
    out dx, al      ; STROBE = 0

    pop dx ax
    ret

imprimir_caracter ENDP
```

P8. Escribir en ensamblador de 80x86 el código necesario para que el puerto paralelo LPT2 genere interrupciones. Se valorará la eficiencia del código.

```
mov ax, 0
mov es, ax
mov dx, es:[040Ah]     ; Lee dirección base de LPT2 desde BIOS
add dx, 2              ; Calcula dirección de registro de control
in al, dx              ; Lee registro de control
or al, 00010000b       ; Activa bit 4 (IRQEN)
out dx, al             ; Modifica registro de control

; A partir de aquí, la señal #ACK del LPT2 activa IRQ5 de
; PIC maestro.
; Para que estas interrupciones lleguen a la CPU es necesario
; que el bit 5 del registro de máscara IMR del PIC maestro
; esté a cero, que es su valor por defecto tras la inicialización
; del PIC.
```

P9. Escribir en ensamblador de 80x86 una subrutina que inicie la emisión a través del altavoz del PC de un sonido de frecuencia lo más próxima posible a **440 Hz usando el temporizador** (*timer*). Se valorará la eficiencia del código.

```
tocar_440Hz PROC FAR
    push ax

    ; Genera palabra de control
    ; SC = 10 (contador 2), RW = 11 (byte bajo + byte alto)
    ; M = 011 (modo 3: onda cuadrada); BCD = 0 (binario)

    mov al, 10110110b       ; Palabra de control SC|RW|M|BCD
    out 43h, al             ; Configura timer

    ; Valor inicial = 1193182 / 440 = 2711,77 ≈ 2712
    ; Frecuencia real = 1193182 / 2712 = 439,96 Hz
    ; 2712 = 10 * 256 + 152

    mov ax, 2712
    out 42h, al       ; Envía byte bajo de valor inicial (152)
    mov al, ah
    out 42h, al       ; Envía byte alto de valor inicial (10)

    ; Activa salida del altavoz y puerta del contador 2

    in al, 61h        ; Lee registro de control
    or al, 00000011b  ; Activa bit 0 (puerta) y bit 1 (salida)
    out 61h, al
```

```
            ; Altavoz empieza a sonar

            pop ax
            ret

        tocar_440Hz ENDP
```

P10. Usando la BIOS (se adjunta un extracto de la documentación de Ralph Brown), escribir en ensamblador de 80x86 una subrutina que posicione el cursor de modo texto en la fila y columna de la página cero especificadas en el registro AX **(AH = columna, AL = fila)**. Se valorará la eficiencia del código.

```
set_cursor PROC FAR
  push ax bx dx

  mov dl, ah    ; Define columna
  mov dh, al    ; Define fila
  mov bh, 0     ; Define página 0
  mov ah, 2     ; Código de operación
  int 10h       ; BIOS vídeo

  pop dx bx ax
  ret
set_cursor ENDP
```

VIDEO - SET CURSOR POSITION

```
AH = 02h
BH = page number
0-3 in modes 2&3
0-7 in modes 0&1
0 in graphics modes
DH = row (00h is top)
DL = column (00h is left)

Return: Nothing
```

P11. Escribir en ensamblador de 80x86 una **rutina de servicio a la interrupción** del reloj de tiempo real (*RTC*), que llame a la subrutina `_Actualizar` cada vez que se reciba una **interrupción de actualización de la hora/fecha**. Se supone que el RTC tiene habilitadas todas sus interrupciones. Se valorará la eficiencia del código.

```
        rsi_RTC PROC FAR

            sti                 ; Activa interrupciones
            push ax

            ; Lee registro C del RTC (banderas de interrupción)

            mov al, 0Ch
            out 70h, al
            in al, 71h

            test al, 00010000b      ; Comprueba bandera UF (Update Flag)
            jz final                ; UF = 0 <==> no hay actualización

            call _Actualizar

final:      mov al, 20h
            out 20h, al         ; EOI al maestro
            out 0A0h, al        ; EOI al esclavo

            pop ax
            iret

        rsi_RTC ENDP
```

P12. Escribir en ensamblador de 80x86 el código que sea necesario para ejecutar un EOI en los manejadores de las siguientes interrupciones:

1Ch (Tic temporizador)	*70h (RTC)*	*16h (E/S Teclado)*

EOI innecesario	`mov al, 20h`
	`out 20h, al`
	`out A0h, al`
		EOI innecesario

P13. Escribir en ensamblador de 80x86 una subrutina que **ponga a 0 el bit IRQEN del primer puerto paralelo (LPT1), dejando intactos los demás bits.** Se valorará la eficiencia del código.

```
IRQEN0_LPT1 PROC FAR
  push ax es dx

  mov ax, 0
  mov es, ax
  mov dx, es:[408h]     ; Lee dirección base de LPT1
  add dx, 2             ; Obtiene dirección de registro control
  in al, dx             ; Lee registro de control
  and al, 11101111b     ; Pone a 0 bit IRQEN
  out dx, al            ; Escribe registro de control

  pop dx es ax
  ret
IRQEN0_LPT1 ENDP
```

P14. Usando la BIOS (se adjunta un extracto de la documentación de Ralph Brown), escribir en ensamblador de 80x86 el código de la función de C `int GetASCIICursor()`, que **retorna en un entero el código ASCII del carácter que se encuentra en la posición actual del cursor en la página 0** de la memoria de vídeo en modo texto. Se valorará la eficiencia del código.

```
_GetASCIICursor PROC FAR
  push bx
  mov ah, 8
  mov bh, 0
  int 10h
  mov ah, 0
  pop bx
  ret
_GetASCIICursor ENDP
```

VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION

```
AH = 08h
BH = page number

Return:
AH = character's attribute
AL = character (ASCII)
```

Category: Video - Int 10h - V

P15. Escribir en ensamblador de 80x86 las **rutinas lejanas** `Sound_500Hz` y `Sound_1000Hz`, que han de activar el sonido del altavoz interno del PC a las **frecuencias de 500Hz y 1000Hz** respectivamente. Se supone que el contador 2 del temporizador 8254 (*timer*) ya ha sido previamente inicializado en modo 3, **codificación binaria** y **RW=11b**. Las dos rutinas solicitadas

han de encargarse también de **activar los bits de menor peso del puerto 61h**. Se valorará la eficiencia del código.

```
Sound_500Hz PROC FAR
      push ax
      mov ax, 2386    ; 1193182 / 500
      out 42h, al
      mov al, ah
      out 42h, al     ; Valor inicial
      in al, 61h
      or al, 00000011b
      out 61h, al     ; Activa altavoz
      pop ax
      ret
Sound_500Hz ENDP


Sound_1000Hz PROC FAR
      push ax
      mov ax, 1193    ; 1193182 / 1000
      out 42h, al
      mov al, ah
      out 42h, al     ; Valor inicial
      in al, 61h
      or al, 00000011b
      out 61h, al     ; Activa altavoz
      pop ax
      ret
Sound_1000Hz ENDP
```

P16. Escribir en ensamblador de 80x86 una rutina de servicio de la interrupción 1Ch (tic del temporizador), que llame de **forma alternada** a las rutinas lejanas **`Sound_500Hz`** y **`Sound_1000Hz`** desarrolladas en el problema anterior, dejando un tiempo entre llamadas de medio segundo aproximadamente, de forma que se obtenga un sonido tipo sirena.

```
rsi_1Ch   PROC FAR          ; Cambia cada 9 * 55 ms = 0,495s
          sti               ; Activar interrupciones
          jmp ini
          cont db 18        ; Variable de control iteración
ini:      dec cs:cont
          jnz nocero
          call Sound_500Hz  ; Primeras 9 iteraciones a 500Hz
          mov cs:cont, 18
          jmp fin
nocero:   cmp cs:cont, 9    ; Últimas 9 iteraciones a 1000Hz
          jne fin
          call Sound_1000Hz
fin:      iret
rsi_1Ch ENDP
```

P17. Escribir en ensamblador de 80x86 llamando a la **interrupción 10h** de la BIOS (se adjunta un extracto de la documentación de Ralph Brown) una subrutina lejana que pinte un **rectángulo sin rellenar** mediante una línea continua de **color negro** (color = 0) y de **un píxel de grosor**. Se supone que la controladora de vídeo ya está configurada en modo gráfico VGA de resolución 320x200 y 256 colores. La esquina superior izquierda del rectángulo está en las coordenadas

(10,10) y la esquina inferior derecha en las coordenadas **(310,190)**. Se valorará la eficiencia del código.

VIDEO - WRITE GRAPHICS PIXEL

AH = 0Ch
BH = page number
AL = pixel color
CX = column
DX = row

Return:
Nothing

Category: Video - Int 10h - V

```
rect PROC FAR
        push ax bx cx dx
        ; AH=0Ch, AL = color negro
        mov ax, 0C00h
        mov bh, 0       ; página 0
        ; Pintar líneas horizontales
        mov cx, 10      ; col = 10
horiz:  mov dx, 10      ; fil = 10
        int 10h
        mov dx, 190     ; fil = 190
        int 10h
        inc cx          ; col++
        cmp cx, 310
        jle horiz       ; col <= 310
        ; Pintar líneas verticales (dx = 190)
vert:   mov cx, 10      ; col = 10
        int 10h
        mov cx, 310     ; col = 310
        int 10h
        dec dx          ; fil--
        cmp dx, 10
        jge vert        ; fil >= 10
        pop dx cx bx ax
        ret
rect ENDP
```

P18. Escribir en ensamblador de 80x86 una subrutina lejana que programe las interrupciones periódicas del reloj de tiempo real (RTC) a una frecuencia aproximada de 2 MHz. Se valorará la eficiencia del código.

```
RTC2MHz PROC FAR
    push ax
    mov al, 0Ah
    out 70h, al
    mov al, 00000010b ; DV = 000 (4,193MHz), RS = 0010
    out 71h, al
    pop ax
    ret
RTC2MHz ENDP
```

P19. Escribir en ensamblador de 80x86 una rutina lejana que configure la frecuencia de las interrupciones periódicas del RTC a 4 Hz y habilite dichas interrupciones en el RTC. Se valorará la eficiencia del código.

```
confRTC PROC FAR
      push ax
      mov al, 0Ah
      out 70h, al       ; Accede a registro 0Ah
      mov al, 00101110b ; DV=010b, RS=1110b (14 == 4 Hz)
      out 71h, al       ; Escribe registro 0Ah
```

```
        mov al, 0Bh
        out 70h, al         ; Accede a registro 0Bh
        in al, 71h          ; Lee registro 0Bh
        mov ah, al
        or ah, 01000000b    ; Activa PIE
        mov al, 0Bh
        out 70h, al         ; Accede a registro 0Bh
        mov al, ah
        out 71h, al         ; Escribe registro 0Bh

        pop ax
        ret
confRTC ENDP
```

P20. Escribir en ensamblador de 80x86 una rutina de servicio de la interrupción periódica del RTC, que cada vez que se ejecute modifique la frecuencia actual del contador 2 del temporizador 8254 (*timer*). En concreto, aumentará en 100 Hz la frecuencia actual del *timer* si ésta es menor que 1000 Hz. No podrán usarse variables globales. El valor 1193182 equivale a 001234DE en hexadecimal. Se supone que el contador 2 del *timer* ha sido previamente inicializado en modo 3, codificación binaria y RW=11b. Se valorará la eficiencia del código.

```
RTC_rsi PROC FAR
        sti
        push ax bx dx

        mov al, 0Ch
        out 70h, al         ; Accede a registro 0Ch de RTC
        in al, 71h          ; Lee registro 0Ch de RTC (reset flags)

        mov bx, 0FFFFh      ; bx == Contador actual

        ; Determina contador inicial (valor máximo de conteo)

findHz:
        mov dx, bx          ; dx == Cont. anterior := cont. actual
        mov al, 10000000b   ; SC | RW | M | BCD
        out 43h, al         ; Memoriza contador 2 actual
        in al, 42h
        mov bl, al
        in al, 42h
        mov bh, al          ; bx == Contador actual

        cmp bx, dx
        jbe   findHz        ; actual <= anterior (decrementando)

        ; actual > anterior => actual (bx) == contador inicial

        ; Calcula frecuencia actual en ax

        mov dx, 0012h
        mov ax, 34DEh       ; dx:ax = 001234DEh = 1193182
        div bx              ; ax (frec) = 1193182 / contador inicial

        cmp ax, 1000        ; frec < 1000?
```

```
        jae final           ; Acaba si frec >= 1000

        add ax, 100         ; frec += 100
        mov bx, ax          ; bx == nueva frecuencia

        ; Calcula nuevo contador inicial en ax
        mov dx, 0012h
        mov ax, 34DEh       ; dx:ax = 001234DEh = 1193182
        div bx              ; ax (inicial) = 1193182 / frec

        ; Envía nuevo contador inicial
        out 42h, al         ; Envía byte bajo
        mov al, ah
        out 42h, al         ; Envía byte alto

final:  ; Envía EOIs (RTC)
        mov al, 20h
        out 20h, al         ; Master PIC
        out 0A0h, al        ; Slave PIC

        pop dx bx ax
        iret

RTC_rsi ENDP
```

P21. Escribir en ensamblador de 80x86 una rutina de servicio de la interrupción periódica del RTC que copie el valor actual de la entrada SLCT del puerto paralelo **LPT1** a la salida STROBE del puerto paralelo **LPT1**. Se valorará la eficiencia del código.

```
rsiRTC PROC FAR
        sti
        push ax dx es

        mov al, 0Ch
        out 70h, al
        in al, 71h  ; Leer registro C de RTC (reset flags)

        mov ax, 0
        mov es, ax
        mov dx, es:[408h]       ; dx == @Datos LPT1
        inc dx                  ; dx == @Estado LPT1
        in al, dx
        mov ah, al              ; ah == Estado LPT1

        inc dx                  ; dx == @Control LPT1
        in al, dx               ; al == Control LPT1

        test ah, 00010000b      ; SLCT?
        jz SLCT0                ; SLCT == 0

        ; SLCT 1
        and al, 11111110b       ; STROBE := 1 (invertido)
        jmp final

SLCT0: or al, 00000001b         ; STROBE := 0 (invertido)
```

```
final: out dx, al              ; Cambiar STROBE

       mov al, 20h
       out 20h, al             ; EOI maestro
       out 0A0h, al            ; EOI esclavo

       pop es dx ax
       iret
rsiRTC ENDP
```

P22. Escribir en ensamblador de 80x86 el código que sea necesario para ejecutar un EOI en los manejadores de las siguientes interrupciones

04h (Overflow)	*1Ch (Timer tic)*	*70h (RTC)*

EOI Innecesario	**EOI Innecesario**	`mov al, 20h`
		`out 20h, al`
		`out A0h, al`

P23. Los pines 2 a 9 del LPT1 controlan una hilera de ocho LEDs, de modo que cuando uno de esos pines está activo a +5V, su LED correspondiente está encendido. Escribir en ensamblador de 80x86 la función cercana `VUmeter`, que recibe en AH el número de LEDs consecutivos que deben ser encendidos (entre 0 y 8). Por ejemplo, si AH=2, se deben encender solamente los LEDs correspondientes a los pines 2 y 3. Se valorará la eficiencia del código

```
VUmeter PROC NEAR
       push ax cx dx es
       mov cl, ah              ; cl == Unos consecutivos
       mov ax, 0FF00h
       rol ax, cl              ; al == Patrón de bits con unos consecutivos

       mov cx, 0
       mov es, cx
       mov dx, es:[0408h]      ; dx == Dirección base de LPT1

       out dx, al        ; Escribe patrón de bits en registro de datos LPT1

       pop es dx cx ax
       ret
VUmeter ENDP
```

P24. Escribir en ensamblador de 80x86 el procedimiento cercano `Carrillon`, que recibe en **CL un número de hora entre 0 y 23** y, mediante el *timer*, debe emitir pitidos intermitentes de **200 Hz**

para indicar la hora de forma sonora. Si la hora es 0 debe emitir 12 pitidos. Si la hora es entre 1 y 12, debe emitir tantos pitidos como indica la hora. Si la hora es entre 13 y 23, debe emitir entre 1 y 11 pitidos respectivamente. Cada pitido consta de un intervalo sonoro de un segundo seguido de un silencio de un segundo. Se dispone de la función cercana `espera1seg`, que realiza un retardo activo de un segundo. **El procedimiento debe configurar e inicializar el *timer***. Se valorará la eficiencia del código.

```
    Carrillon PROC NEAR
            push ax cx

            cmp cl, 0           ; cl == hora 0..23
            jne nocero
            mov cl, 24          ; 0 horas equivale a 24 (12 pitidos)
nocero:     cmp cl, 12          ; cl == hora 1..24
            jbe tocar           ; hora <= 12
            sub cl, 12          ; Convierte hora 13..24 a 1..12

tocar:      mov al, 10110110b   ; Palabra de control SC|RW|M|BCD
            out 43h, al         ; Configura timer 2

            mov ax, 5966        ; Valor inicial = 1193182Hz / 200Hz == 5966
            out 42h, al         ; Envía byte bajo de valor inicial (78,4Eh)
            mov al, ah
            out 42h, al         ; Envía byte alto de valor inicial (23,17h)

            in al, 61h          ; Lee registro de control

bucle:      or al, 00000011b    ; Activa bit 0 (puerta) y bit 1 (salida)
            out 61h, al         ; Inicia pitido
            call espera1seg     ; Retardo 1 segundo

            and al, 11111100b ; Desactiva bit 0 (puerta) y bit 1 (salida)
            out 61h, al         ; Finaliza pitido
            call espera1seg     ; Retardo 1 segundo

            dec cl
            jnz bucle           ; hora > 0

            pop cx ax
            ret
    Carrillon ENDP
```

P25. Escribir en ensamblador de 80x86 una rutina de servicio (*driver*) de la interrupción del RTC, que cada vez que se ejecute **lea un byte del puerto 1234h y lo almacene en un *buffer* circular de 512 bytes**. Los bytes leídos en ejecuciones sucesivas del *driver* deben almacenarse secuencialmente en el *buffer* en posiciones sucesivas. Como el *buffer* es circular, las escrituras continúan desde el principio cuando se llega al final. Tanto el *buffer* como las variables globales necesarias para realizar esta rutina de servicio deben **declararse dentro del código del *driver***. Se valorará la eficiencia del código.

```
rsi_RTC PROC FAR
      sti                   ; Activa interrupciones
      jmp ini
```

```
        ; Declara buffer de 512 bytes e índice de escritura
        buffer db 512 dup (?)
        index dw 0

ini:    push ax dx di

        ; Lee registro C del RTC (banderas de interrupción)
        mov al, 0Ch
        out 70h, al
        in al, 71h

        mov dx, 1234h
        in al, dx               ; Lee byte de puerto 1234h

        mov di, cs:index
        mov cs:buffer[di], al   ; Almacena byte en posición escritura
        inc di                  ; Incrementa índice de escritura
        cmp di, 512             ; índice == 512?
        jne fin                 ; buffer no está lleno
        mov di, 0               ; buffer lleno: se reescribe desde principio

fin:    mov cs:index, di        ; Actualiza índice de escritura

        mov al, 20h
        out 20h, al             ; EOI al maestro
        out 0A0h, al            ; EOI al esclavo

        pop di dx ax
        iret

rsi_RTC ENDP
```

P26. Escribir en ensamblador de 80x86 llamando a la **interrupción 10h** de la BIOS (se adjunta un extracto de la documentación de Ralph Brown) una subrutina lejana que imprima en la **columna 40 de la página 0 una línea vertical de 25 caracteres '=' de color amarillo parpadeante con fondo rojo**. Se supone que la controladora de vídeo ya está configurada en modo CGA de texto de resolución 80x25. Se valorará la eficiencia del código.

VIDEO - SET CURSOR POSITION

```
AH = 02h
BH = page number
DH = row (00h is top)
DL = column (00h is left)

Return: Nothing
```

VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION

```
AH = 09h
AL = ASCII character to display
BH = page number
BL = attribute
CX = number of times to write character

Return: Nothing
```

Category: Video - Int 10h - V

```
Linea PROC FAR

        push ax bx cx dx
        mov bl, 11001110b  ; Atributo: amarillo parpadeante en fondo rojo
        mov bh, 0       ; Página 0
        mov al, '='     ; código ASCII
        mov dl, 40      ; Columna
        mov dh, 24      ; Fila inicial
        mov cx, 1       ; Carácter se escribe una sola vez

bucle:  mov ah, 2
        int 10h         ; Sitúa cursor en fila dh y columna dl
        mov ah, 9
        int 10h         ; Imprime carácter
        dec dh          ; Decrementa fila
        jns bucle       ; fila >= 0

        pop dx cx bx ax
        ret

Linea ENDP
```

P27. Usando la BIOS (se adjunta un extracto de la documentación de Ralph Brown), escribir en ensamblador de 80x86 el código de la rutina de C **`void GetASCIIRow( char row, char *buffer)`**, que reposicionando el cursor de texto, rellena los 80 bytes del buffer indicado con los códigos ASCII de los 80 caracteres que se encuentran en la fila de pantalla indicada, suponiendo la página 0. Se supone que la controladora de vídeo ya está configurada en modo CGA de texto de resolución 80x25. El programa en C está compilado en **modelo pequeño (*small*)**. Se valorará la eficiencia del código.

```
_GetASCIIRow PROC NEAR

   push bp
   mov bp, sp
   push ax bx dx di

   mov dh, [bp+4] ; dh == row
   mov di, [bp+6] ; di == buffer

   mov dl, 0      ; dl == col
   mov bh, 0      ; bh == página 0

bucle:
   mov ah, 2
   int 10h    ; Posiciona cursor
   mov ah, 8
   int 10h    ; Lee ASCII en al
```

VIDEO - SET CURSOR POSITION

```
AH = 02h
BH = page number
DH = row (00h is top)
DL = column (00h is left)

Return: Nothing
```

VIDEO - READ CHARACTER AND ATTRIBUTE AT CURSOR POSITION

```
AH = 08h
BH = page number

Return:
AH = character's attribute
AL = character (ASCII)
```

Category: Video - Int 10h - V

```
    mov [di], al   ; Escribe ASCII en buffer
    inc di         ; Avanza buffer
    inc dl         ; col++

    cmp dl, 80
    jne bucle

    pop di dx bx ax
    pop bp
    ret

_GetASCIIRow ENDP
```

P28. Escribir en ensamblador de 80x86 mediante llamadas sucesivas a la **interrupción 1Ah** de la BIOS (se adjunta un extracto de la documentación de Ralph Brown) una rutina lejana con un bucle infinito que llame a la función de C externa `Tick( int hour, int minutes, int seconds)` cada vez que haya transcurrido un segundo en el reloj de tiempo real (RTC). Después de cada llamada a la interrupción, primero se comprobará si hay error (acarreo igual a uno) o no. Si hay error se llamará a la interrupción de nuevo. Si no hay error, se deberá comprobar si ha transcurrido un segundo desde la llamada anterior. En ese caso se llamará a la función externa. Finalmente se iterará de nuevo. Se valorará la eficiencia del código.

```
RTC PROC FAR

        mov bh, 0
        mov al, 0 ; al == old seconds
        mov ah, 2

iterate: int 1Ah
        jc iterate ; CF==1 => Error
        cmp al, dh ; old seconds == seconds?
        je iterate ; old seconds == seconds

        ; New second
        mov al, dh ; old seconds := seconds

        mov bl, dh
        push bx ; push seconds
        mov bl, cl
        push bx  ; push minutes
        mov bl, ch
        push bx  ; push hour
        call _Tick
        add sp, 6 ; Balance stack
        jmp iterate

RTC ENDP
```

TIME - GET REAL-TIME CLOCK TIME

```
AH = 02h

Return:
CF clear if successful
CH = hour (BCD)
CL = minutes (BCD)
DH = seconds (BCD)
DL = daylight savings flag (00h standard
time, 01h daylight time)
CF set on error (i.e. clock not running
or in middle of update)
```

Category: - Int 1Ah - T

P29. Escribir en ensamblador de 80x86 una rutina lejana que **recibe en AH un número binario de 4 bits**, lo convierte a **código Gray de 4 bits** y escribe ese código resultante en los **4 bits de más peso del registro de datos del puerto paralelo 2** (LPT2). **Los 4 bits de menos peso del registro de datos no se pueden cambiar**. La conversión de código binario a código Gray se realiza llamando a una función externa de C cuyo prototipo se muestra en el siguiente recuadro. Esa función está compilada en **modelo largo** (*large*). Se valorará la eficiencia del código.

```
unsigned int BinaryToGray( unsigned int n );
```

```
WriteGrayLPT2 PROC FAR

        push ax cx dx es

        mov al, ah
        mov ah, 0
        push ax                 ; Apila código binario
        call _BinaryToGray      ; ax := código Gray
        add sp, 2               ; Equilibra pila

        mov ah, al              ; ah := Código Gray


        mov dx, 0
        mov es, dx
        mov dx, es:[040Ah]      ; Dirección base de LPT2
        in al, dx               ; al := Registro datos LPT2

        mov cl, 4
        shl ah, cl              ; Mueve código Gray a 4 bits altos de ah
        and al, 00001111b       ; Pone a cero 4 bits altos de al
        or al, ah               ; Escribe Gray en 4 bits altos de al

        out dx, al              ; Envía al puerto LPT2

        pop es dx cx ax
        ret

WriteGrayLPT2 ENDP
```

P30. Completar los recuadros del programa de ensamblador de 80x86 mostrado a continuación usando instrucciones básicas (sin instrucciones de cadena ni de bucle) y sin variables auxiliares. Este programa **implementa la rutina de servicio de la interrupción (RSI) del RTC** y transmite una secuencia Morse codificada según el problema P3. En concreto, cuando la variable global `Transmit` vale 0, la RSI termina. Si `Transmit` vale 1, lee el carácter ('1' o '0') almacenado en la posición `Index` de la cadena `Buffer` y activa (si '1') o desactiva (si '0') el altavoz interno del PC. Si el carácter leído es un cero binario, resetea las variables `Transmit` e `Index`. El **número de segmento** físico donde están las tres **variables globales** está almacenado en el **registro DS**. **No hay ningún *assume* activo**. Se supone que el RTC y el Timer están ya configurados a las frecuencias correctas, y que las interrupciones periódicas del RTC están habilitadas. Se valorará la eficiencia del código.

```
rsi70h PROC FAR
        sti
        push ax si
```

```
        mov al, 0Ch                 ; Leer Registro C
        out 70h, al
        in al, 71h
        cmp ds:_Transmit, 1         ; Transmit == 1?
        jne endrsi                  ; Transmit == 0
        mov si, ds:_Index           ; si := Index
        mov ah, ds:_Buffer[si]      ; ah := Buffer[Index]
        cmp ah, 0                   ; Buffer[Index] == 0?
        je end_trans                ; Buffer[Index] == 0
        cmp ah, '1'                 ; Buffer[Index] == '1'?
        jne zero                    ; Buffer[Index] != '1'
        in al, 61h                  ; Activar altavoz
        or al, 00000011b
        out 61h, al
        jmp next
zero:
        in al, 61h                  ; Desactivar altavoz
        and al, 11111100b
        out 61h, al
next:
        inc ds:_Index               ; Index++
        jmp endrsi
end_trans:
        mov ds:_Index, 0            ; Index := 0
        mov ds:_Transmit, 0         ; Transmit := 0
endrsi:
        mov al, 20h                 ; EOI y finalizar
        out 20h, al
        out 0A0h, al
        pop si ax
        iret
rsi70h ENDP
```

P31. Completar los recuadros del programa de ensamblador de 80x86 mostrado a continuación usando etiquetas o instrucciones básicas (sin instrucciones de cadena ni de bucle) y sin variables auxiliares. Este programa **implementa la rutina de servicio de la interrupción (RSI) del Puerto Paralelo 2 (LPT2)**. Esta rutina lee un **byte de datos del puerto bidireccional**, le aplica un **procesamiento externo** llamando a la función de C con signatura: **`char`** `Process(`**`char`** `port,` **`char`** `data)` y envía el resultado de esa función al puerto paralelo mediante el **protocolo ACK**. El primer parámetro de la función externa es el número de puerto paralelo (2 en este caso). Se supone que las interrupciones del LPT2 están habilitadas. Se valorará la eficiencia del código.

```
rsi0Dh PROC FAR
    sti
    push ax dx es
    mov ax, 0                   ; dx := Dirección base LPT2
    mov es, ax
    mov dx, es:[040Ah]
    add dx, 2                   ; dx := Dirección reg. control
    in al, dx                   ; Activar BIDIR
    or al, 00100000b
    out dx, al
    sub dx, 2                   ; Leer reg. datos
    in al, dx
    push ax                     ; Pasar parámetros a función
    mov ax, 2
    push ax
    call _Process               ; Llamar a función
    add sp, 4                   ; Equilibrar pila
    out dx, al                  ; reg. datos := resultado función
    add dx, 2                   ; Activar STROBE
    in al, dx
    or al, 00000001b
    out dx, al
    dec dx                      ; dx := Dirección reg. estado
espera:
    in al, dx                   ; Espera activa mientras BUSY == 1
    test al, 10000000b
    jnz espera
    inc dx                      ; Desactivar STROBE
    in al, dx
    and al, 11111110b
    out dx, al
    mov al, 20h                 ; Finalizar RSI
    out 20h, al
    pop es dx ax
    iret
rsi0Dh ENDP
```