

A

**Instrucciones generales para la realización de este examen**  
La respuesta debe escribirse en el hueco existente a continuación de cada pregunta **con letra clara**.

Cada respuesta correcta suma un punto Cada respuesta incorrecta, ilegible o vacía no suma ni resta. El total de puntos se dividirá entre el total de preguntas y se multiplicará por 10 para obtener la nota del examen.

Se va a lanzar la versión 2.0 del sistema con capacidad para mostrar gráficos en la pantalla/marcador de un estadio deportivo. Se pretende que el operador del marcador puede elegir además de las figuras básicas (equis, asterisco, espiral y triángulo) otras que parten de estas mismas pero sufren ciertas transformaciones matemáticas. El marcador tiene las mismas características que el periférico **Pantalla** visto en la asignatura. El operario maneja el marcador a través de un periférico de control similar al periférico **Luces** visto en la asignatura. El operario controlará con los interruptores de los 2 bits menos significativos la figura base (equis, asterisco, espiral ó triángulo) que quiere mostrar en el marcador. Con los siguientes 2 bits la transformación matemática a aplicar a la figura base a representar, con la combinación: 00->figura base, 01->NOT de la figura base y 10->NEG de la figura base. Con los interruptores de los 6 bits menos significativos del byte alto se controlará los atributos de color de texto y de fondo que presentará el gráfico. En la figura siguiente se resumen esta información:

Registro datos periférico Luces

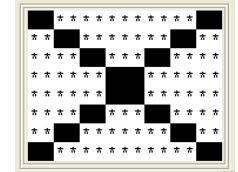
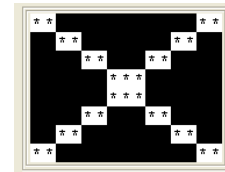
XX	RGB-RGB	XXXX	2 bits	2bits
Si n us o	Atributos fondo- tinta	Sin uso	Transfor- mación	Figura a dibujar

Para comprobar el funcionamiento del dispositivo se ha escrito el siguiente código:

- El programa principal:** Establece el vector de interrupción **7 (siete)** asociado al periférico **Luces** y su rutina.
- rutina\_luces:** Rutina que se ejecutará cuando el periférico **Luces** solicite una interrupción. Llamará al procedimiento *“aplica\_transformacion”* que con la figura seleccionada y la transformación elegida escribirá la nueva figura en una zona de memoria etiquetada como **figura**. Después llamará al procedimiento *“dibuja\_figura”* para mostrar en pantalla la figura transformada por el anterior procedimiento.
- dibuja\_figura:** Procedimiento para pintar en la pantalla la figura seleccionada. La figura se formará escribiendo **\*** en las posiciones de pantalla que se indiquen. El asterisco tendrá los atributos de fondo y color que se den como parámetro. Se pasarán a través de la pila y en el orden indicado: dirección del vector donde se encuentra codificada la figura a mostrar, y los atributos en el byte más significativo de la palabra.
- aplica\_transformacion:** Procedimiento para dada una figura base, se le aplicará la transformación indicada por los bits 3º y 4º del periférico luces y escribiendo dicha transformación en una zona de memoria fija. Se pasarán a través de la pila y en el orden

indicado: dirección del vector donde se encuentra codificada la figura a transformar, y el registro de datos del periférico luces.

Para codificar la figura que se quiere mostrar en pantalla se usará un vector de 8 elementos. Cada elemento se corresponde con la codificación de una línea de la pantalla y será un número binario de 15 bits (un bit para cada columna de la pantalla) Cada bit indica si hay que escribir en el píxel de esa columna. En la figura de la izquierda se muestra el aspecto de la pantalla al dibujar la equis codificada en la zona de datos, y a la derecha el aspecto al aplicarle la transformación 01->NOT.



```

ORIGEN 0500h
INICIO primera
.PILA XXXX
.DATOS
    dir_pantalla VALOR 0ff00h
    dir_luces VALOR 0ffd0h
    dir_ctrl_pantalla VALOR 0ff78h

p_de_dos_inverso VALOR 16384, 8192
    VALOR 4096, 2048, 1024, 512, 256, 128, 64
    VALOR 32, 16, 8, 4, 2, 1

figura VALOR 8 VECES 0
equis VALOR 110000000000011b
        VALOR 001100000001100b
        VALOR 000011000110000b
        VALOR 000000111000000b
        VALOR 000000111000000b
        VALOR 000011000110000b
        VALOR 001100000001100b
        VALOR 110000000000011b
;::::::::::::::::::::::::::::::::::
; Datos omitidos intencionadamente.
;
; Son codificaciones de diferentes
    
```



```
;figuras.
;
.CODIGO
PROCEDIMIENTO rutina_luces

    PUSH R0 ;Dirección de inicio figura.
    PUSH R1 ;Atributo.
    PUSH R2 ;Nº figura a mostrar.
    PUSH R3 ;Auxiliar.
    PUSH R4 ;Estado luces.
    PUSH R5 ;Auxiliar.

;Leemos el estado del periférico luces en R4.
    ----- HUECO 1 -----
    MOVH R0,BYTEALTO DIRECCION dir_luces
    MOVL R0,BYTEBAJO DIRECCION dir_luces
    MOV R1,[R0]
    MOV R4,[R1]
;R4 contiene el atributo y la figura a mostrar.
;En R1 ponemos el atributo.
    XOR R3,R3,R3
    MOVH R3,3Fh
    AND R1,R3,R4 <----- 2a

;Ponemos en R2 el numero de la figura se tiene que
; mostrar.
    XOR R3,R3,R3
    MOVL R3,3
    AND R2,R3,R4 <----- 2b

;Ponemos en R0 la dirección de la primera figura.
    MOVL R0,BYTEBAJO DIRECCION equis
    MOVH R0,BYTEALTO DIRECCION equis

;Ponemos en R3 nº filas de la pantalla.
    XOR R3,R3,R3
    MOVL R3,8

;Comprobamos si es la primera figura la que
; tenemos que mostrar.
    XOR R5,R5,R5;
    COMP R2,R5
    BRZ es_figura_cero
;En caso contrario avanzamos tantas veces el nº
; columnas hasta llegar al comienzo de la figura
; a mostrar.
seguimos:
    ADD R0,R0,R3
    DEC R2
    BRNZ seguimos
es_figura_cero:

;Borramos la pantalla.
    MOVH R3,BYTEALTO DIRECCION dir_ctrl_pantalla
    MOVL R3,BYTEBAJO DIRECCION dir_ctrl_pantalla
    MOV R5,[R3]
    XOR R3,R3,R3
    MOVL R3,3
    MOV [R5],R3
```

```
;Llamamos al procedimiento para realizar la
; transformación

    ----- HUECO 3 -----

    PUSH R0
    PUSH R4
    CALL aplica_transformacion
    INC R7
    INC R7

;Llamamos al procedimiento para dibujar la figura.
    MOVL R5 ,BYTEBAJO DIRECCION figura
    MOVH R5 ,BYTEALTO DIRECCION figura

    PUSH R5
    PUSH R1
    CALL dibuja_figura
    INC R7
    INC R7
    POP R5
    POP R4
    POP R3
    POP R2
    POP R1
    POP R0

IRET
FINP

PROCEDIMIENTO aplica_transformacion
    PUSH R6
    MOV R6,R7
    PUSH R0 ;Tendrá estado luces y Auxiliar
    PUSH R1 ;Tendrá direcc. figura a transformar
    PUSH R2 ;Tendrá direcc. figura transformada
    PUSH R3 ;Auxiliar y codificación transformación

;recuperamos segundo param., estado luces, en R0

    ----- HUECO 4 -----

    INC R6
    INC R6
    MOV R0,[R6]

;recuperamos primer param., direcc. figura a
; transformar, en R1
    INC R6
    MOV R1,[R6]

;La figura transformada estará en la variable
;global figura. Ponemos la dirección de
;la variable en R2

    ----- HUECO 5 -----

    MOVL R2 ,BYTEBAJO DIRECCION figura
    MOVH R2 ,BYTEALTO DIRECCION figura

;la transformación que queremos hacer está
;codificada en los bit 3º y 4º, hacemos una
;mascara para quedarnos con esa información
    XOR R3,R3,R3
    MOVL R3,0Ch
```

```
    AND R3,R0,R3

;si la codificación es cero no hacemos
; transformación, copiamos sin más
    BRZ transforma_nada

;si la codificación es 4 hacemos transformación NOT
    XOR R0,R0,R0
    MOVL R0,4
    COMP R0,R3
    BRZ transforma_NOT

;si la codificación es 8 hacemos transformación NEG
    XOR R0,R0,R0
    MOVL R0,8
    COMP R0,R3

    ----- HUECO 6 -----

    BRZ transforma_NEG
    JMP salida

transforma_NOT:
    XOR R0,R0,R0
    MOVL R0,8
salta_transforma_NOT:
    MOV R3,[R1] <----- 7a
    NOT R3
    MOV [R2],R3 <----- 7b
    INC R2
    INC R1
    DEC R0
    BRNZ salta_transforma_NOT
    JMP salida
transforma_NEG:

;
; Código omitido intencionadamente similar al
; etiquetado como "transforma_NOT"
;

transforma_nada:
;
; Código omitido intencionadamente similar al
; etiquetado como "transforma_NOT"
;

salida:
    POP R3
    POP R2
    POP R1
    POP R0
    POP R6

    ----- HUECO 8 -----

RET
FINP

PROCEDIMIENTO dibuja_figura
;
; Código omitido intencionadamente
```



```
; Tenemos 8 instrucciones PUSH
;
FINP

primera:

;Instalamos la rutina del perif. Luces vector 7.
MOVL R1, BYTEBAJO DIRECCION rutina_luces
MOVH R1, BYTEALTO DIRECCION rutina_luces
XOR R0, R0, R0
MOVL R0,7

----- HUECO 9 -----

MOV [R0], R1
STI
JMP -1
FIN
```

– ¿Qué instrucción o instrucciones faltan en HUECO 1? (Usa R1 como registro auxiliar)

```
MOVH R0,BYTEALTO DIRECCION
dir_luces

MOVL R0,BYTEBAJO DIRECCION
dir_luces

MOV R1,[R0]
```

– Completa las instrucciones marcadas como 2a y 2b?

```
2a: AND R1,R3,R4

2b: AND R2,R3,R4
```

– ¿Qué instrucción o instrucciones faltan en HUECO 3?

```
PUSH R0
PUSH R4
CALL aplica_transformacion
```

```
INC R7

INC R7
```

– ¿Qué instrucción o instrucciones faltan en HUECO 4?

```
INC R6

INC R6

MOV R0,[R6]
```

– ¿Qué instrucción o instrucciones faltan en HUECO 5?

```
MOVL R2 ,BYTEBAJO DIRECCION
figura

MOVH R2 ,BYTEALTO DIRECCION
figura
```

– ¿Qué instrucción o instrucciones faltan en HUECO 6?

```
BRZ transforma_NEG

JMP salida
```

– Completa las instrucciones marcadas como 7a y 7b?

```
7a: R3,[R1]

7b: [R2],R3
```

– ¿Qué instrucción o instrucciones faltan en HUECO 8?

```
RET

FINP
```

– ¿Qué instrucción o instrucciones faltan en HUECO 9?

```
MOV [R0], R1

STI

JMP -1
```

– ¿Cuál es el tamaño mínimo de pila que debemos reservar en la directiva .PILA para que el programa funcione correctamente? (Expresar en decimal)

```
19
```

— Completa los caracteres que faltan en el interior de las casillas, teniendo en cuenta los caracteres que ya hay y sus códigos ASCII.

44h	64h	6Bh	4Ah	38h	35h
D	d	k	J	8	5

– La cantidad 80140000h representa un número en formato coma flotante IEEE 754 en precisión simple. ¿Con qué cantidad decimal se corresponde? (Expresar en potencias de dos)

```
-(2-129+2-131)
```

Se sabe que en la CPU teórica un ciclo dura 0,25 milisegundos. Teniendo en cuenta el siguiente



fragmento de código:

```
XOR R1, R1, R1
XOR R0, R0, R0
INC R1
INC R0
DEC R0
COMP R0, R1
BRNZ final
MOVL R3, FFh
final: MOV R5, R2
XOR R1,R1,R1
```

– ¿Cuántos milisegundos empleará en ejecutar dicho código?

12

— En una UC microprogramada para la CPU teórica, las palabras de control se interpretan como se indica en la figura (solamente se muestra los 11 bits menos significativos de la palabra de control).

..	R3-IB	IB-R3	R0-IB	IB-R0	IB-TMPE	TMPE-CLR	SUB	TMPS-IB	IB-PC	ALU-SR	FIN
...											

Para una determinada instrucción se generan las señales de control para las siguientes palabras de control:

Paso 4: 001 0100 0000

Paso 5: 100 0001 0011

¿Cuál es la codificación de la instrucción que se ejecuta con esas palabras de control?  
Expresar el resultado en hexadecimal

6860h (COMP R0,R3)

En un computador basado en la CPU elemental se tiene el siguiente mapa de memoria:

- Dispositivos de memoria RAM a partir de la posición de memoria 0000h y tamaño 36K
- Dispositivos de memoria ROM a partir de la posición de memoria C800h y tamaño 14K

¿Qué tamaño del espacio de direcciones queda libre para poder mapear dispositivos de Entrada/Salida? ¿A partir de qué dirección (en hexadecimal)?

Tamaño: 14K

Dirección comienzo: 9000h



En el espacio de direcciones descrito anteriormente la memoria RAM está formada por 9 dispositivos de memoria cuya organización es del tipo 4Kx16. y cada uno está formado por 8 bancos de 8 chips

– ¿Cuál es la organización interna de los Chips? (contestar en el formato NxM; por ejemplo 32Kx1)

512x2

– ¿Cuántas líneas de dirección entrarán a cada dispositivo 4Kx16 anterior? ¿Y cuántas de esas líneas de dirección se conectarán al decodificador de cada dispositivo 4Kx16?

Nº líneas al dispositivo: 12

Nº líneas al decodificador: 3

– ¿Cuál o cuáles de las siguientes afirmaciones son CIERTAS (puedes responder “ninguna” o “todas” si así lo consideras)?

A).- El valor de un vector de interrupción indica la prioridad de la interrupción asociada a ese vector.

B).- Al final de la rutina de servicio de una interrupción **no** se tiene que ejecutar la instrucción **STI** para activar el flag de interrupción que se había puesto a cero durante la fase de aceptación.

C),. La línea **INTA** nunca se puede activar si no se ha activado la línea **INT** previamente.

D).- La rutina de servicio de una interrupción puede recibir parámetros a través de la pila.

B y C

— Con objeto de mejorar la funcionalidad del juego de instrucciones de la CPU elemental se implementa una versión para la operación ADD que hace uso de direccionamiento indirecto a memoria La nueva versión: **ADD [R0],R1,R2** almacena en la dirección de memoria apuntada por R0 la suma del contenido de los registros R2 y R1. Completa acciones de control necesarias para llevar a cabo dicha instrucción .

Pasos	Acciones de Control
4	R1-IB, IB-TMPE
5	R2-IB, ADD, ALU_SR, ALU_TMPS
6	TMPS_IB, IB-MDR
7	R0-IB, IB-MAR, WRITE
8	FIN
9	

Se sabe que las entradas de la ALU de 16 bits vista en clase tienen los siguientes valores:

- En el **operando A** se encuentra la combinación de bits que representa el +9 en formato exceso a Z con 16 bits y exceso 120.
- En el **operando B** se encuentran el número positivo más grande que se puede codificar en complemento a 2
- **CarryIn=0, Resta=0, Op1=1 y Op0=1.**

Se recuerda que las ALUs elementales que componen la ALU de 16 bits tienen conectadas las salidas de una puerta AND, de una puerta

OR, de una puerta XOR y el resultado de un sumador elemental, a las entradas e0, e1, e2 y e3 del multiplexador, respectivamente.

– ¿Cuál es el resultado obtenido en la salida? Expresar el resultado con **cuatro dígitos hexadecimales**.

8080h