Pregunta 1. Investigue las instrucciones de control de señalizadores (STC, CLC, CMC, CLD, STD, STI, CLI, LAHF, SAHF, PUSHF y POPF), luego mencione la función de cada instrucción. Además, responde las siguientes preguntas:

a) ¿Cuál es la instrucción que almacena el registro de señalizadores o registro de estado en la pila?.

b) ¿Cuál es la instrucción que activa o inactiva la bandera de interrupciones?.

c) ¿Cuál es la instrucción que activa e inactiva la bandera de acarreo?.

a) PUSHF
b) CLI: desactiva las interrupciones al borrar la bandera de interrupciones (IF).
   STI: activa las interrupciones al establecer la bandera de interrupciones (IF).
c) CLC: se utiliza para limpiar (inactivar) la bandera de acarreo, es decir, la pone en 0.
   STC: se utiliza para establecerla (activarla), es decir, la pone en 1.

Instrucción	Función
STC	Set Carry Flag (establece el flag de acarreo)
CLC	Clear Carry Flag (borra el flag de acarreo)
CMC	Complement Carry Flag (complementa el flag de acarreo)
CLD	Clear Direction Flag (borra el flag de dirección)
STD	Set Direction Flag (establece el flag de dirección)
STI	Set Interrupt Flag (establece el flag de interrupción)
CLI	Clear Interrupt Flag (borra el flag de interrupción)
LAHF	Load Status Flags into AH (carga los flags de estado en AH)
SAHF	Store AH into Status Flags (almacena AH en los flags de estado)
PUSHF	Push Flags onto Stack (empuja los flags a la pila)
POPF	Pop Flags from Stack (saca los flags de la pila)

Pregunta 2. Explicite las operaciones en binario para las instrucciones AND, OR, XOR y NOT, con los siguientes operandos 1F y 65, tenga en cuenta que los valores de los operandos están expresados en hexadecimal.

1F en binario es 0001 1111
65 en binario es 0110 0101
Si aplicamos la instrucción AND el resultado es 0000 0101 (05)
Si aplicamos la instrucción OR el resultado es 0111 1111 (7F)
Si aplicamos la instrucción XOR el resultado es 0111 1010 (7A)
Si aplicamos la instrucción NOT el resultado es 1110 0000 (E0)

Pregunta 3. Investigue las instrucciones de salto (JZ, JNZ, JS, JGE y JMP), luego mencione la función de cada instrucción.

- JZ (Jump if Zero): Realiza un salto a una dirección específica si el flag cero (Zero Flag) está activo, lo que indica que el resultado de la última operación fue cero.
- JNZ (Jump if Not Zero): Realiza un salto a una dirección específica si el flag cero no está activo, lo que indica que el resultado de la última operación no fue cero.
- JS (Jump if Sign): Realiza un salto a una dirección específica si el flag de signo (Sign Flag) está activo, indicando un resultado negativo.
- JGE (Jump if Greater or Equal): Realiza un salto a una dirección específica si el resultado de la comparación fue mayor o igual.
- JMP (Jump): Realiza un salto incondicional a una dirección específica, sin evaluar ninguna condición.

Ejercicio 1. Dado un banco de 100 datos ubicados a partir de la posición 0200, realice un programa que recupere el mayor de ellos en la posición 0300.

```
mov si, 0200        ; Dirección de inicio de los datos
mov cx, 64          ; Contador de 100 datos (64 = 100 decimal)
mov ax, [si]        ; Cargar el primer dato en AX
mov bx, ax          ; Inicializar BX con el primer dato
*inc si             ; Incrementar SI para apuntar al siguiente dato
cmp bx, [si]        ; Comparar BX con el siguiente dato
jge siguiente       ; Si BX >= siguiente dato, saltar a 'siguiente'
mov bx, [si]        ; Si BX < siguiente dato, actualizar BX
siguiente:
loop *              ; Decrementa CX y repetir el bucle si CX no es cero
mov si, 0300        ; Dirección para almacenar el valor máximo
mov [si], bx        ; Almacenar el valor máximo en la posición 0300
int 20              ; Finalización del programa
```

Ejercicio 2. Guardar en la posición de memoria 0200 el promedio de diez datos ubicados a partir de la posición 01F6.

```
mov si, 01F6        ; SÍ apunta a la dirección 01F6
mov cx, 10          ; Número de datos
xor ax, ax          ; Inicializar AX (suma) a 0
*  add al, [si]     ; Sumar el dato actual a AL
inc si              ; Mover al siguiente dato
loop *              ; Repetir hasta que CX llegue a 0
mov bl, 10          ; Divisor (número de datos)
xor ah, ah          ; Limpiar AH para división
div bl              ; AX = suma / 10, AL = promedio
mov [0200], al      ; Guardar el resultado en la dirección 0200
int 20              ; Finalización del programa
```

Ejercicio 3. Dado un banco de 100 datos ubicados a partir de la posición 0500, almacenar en la posición 020A la cantidad de datos que posean su 5to bit en 1.

```
mov si, 0500        ; SI apunta a la dirección 0500
mov cx, 100         ; Número de datos
xor ax, ax          ; Inicializar AX (contador) a 0
* mov bl, [si]      ; Cargar el dato en BL
test bl, 20         ; Verificar el 5.º bit (20 = 0010 0000)
jz siguiente        ; Si el 5.º bit es 0, saltar a siguiente
inc ax              ; Incrementar el contador
inc si              ; Mover al siguiente dato
loop*               ; Repetir hasta que CX llegue a 0
mov [020A], al      ; Guardar la cantidad en la dirección 020A
int 20              ; Finalización del programa
```

Ejercicio 4. Realice un programa que tome un dato de la posición de memoria 0200. Enmascare el medio byte superior, convierta a BCD y lo almacene en la posición 0210.

```
mov al, [0200]      ; Cargar el dato de 0200h en AL
and al, 0Fh         ; Enmascarar el medio byte superior (mantener solo los 4 bits inferiores)
mov bl, 0           ; Inicializar BL para contar el número de restas
**cmp al, 10        ; Comparar AL con 10
 jb *               ; Si AL < 10, saltar a guardar el resultado final
sub al, 10          ; Restar 10 de AL
inc bl              ; Incrementar el contador BL
jmp **              ; Repetir el bucle
*mov ah, bl         ; Mover el contador (dígito más significativo) a AH
or ah, al           ; Combinar los dígitos en un solo byte BCD
mov [0210], ah      ; Almacenar el valor BCD en la posición 0210
int 20              ; Finalización del programa
```