1. De acuerdo al código de prueba https://onecompiler.com/assembly/443pg5uuh responde y desarrolla lo siguiente:

a. Agrega comentarios en el código explicando su funcionamiento.

```
section .data
num1 db 5           ; Primer número (byte)
num2 db 11          ; Segundo número (byte)
result db 0         ; Variable para almacenar el resultado
msg db 'Resultado: ', 0   ; Cadena a imprimir

section .bss
buffer resb 4       ; Buffer para guardar el carácter convertido a ASCII

section .text
global _start

_start:
    mov al, [num1]      ; Carga num1 en AL
    add al, [num2]      ; Suma num2 a AL
    mov [result], al    ; Guarda el resultado en memoria

    ; Convertir el resultado numérico a su código ASCII
    movzx eax, byte [result] ; Extiende el byte a 32 bits sin signo
    add eax, 48         ; Suma 48 ('0' en ASCII) para obtener el carácter ASCII del número

    mov [buffer], al    ; Guarda en buffer el carácter a imprimir

    ; Imprimir el texto "Resultado: "
    mov eax, 4          ; syscall write
    mov ebx, 1          ; descriptor STDOUT
    mov ecx, msg        ; dirección del texto
    mov edx, 11         ; longitud
    int 0x80

    ; Imprimir el carácter convertido
    mov eax, 4
    mov ebx, 1
    mov ecx, buffer
    mov edx, 1
    int 0x80

    ; Salir del programa
    mov eax, 1          ; syscall exit
    xor ebx, ebx
    int 0x80
```

b. ¿Para qué sirve la instrucción ‘movzx’?

movzx significa Move with Zero-Extend. Copia un valor pequeño (byte o Word) a un registro de 16/32 bits y llena los bits superiores con ceros, no conserva signos.

c. ¿Está usando algún modo de direccionamiento? ¿Cuál?

Si, esta usando modo de direccionamiento directo en por ejemplo: mov al [num1], mov al [num2], movzx eax byte [result]. El operando entre corchetes indica que se esta accediendo a una dirección de memoria fija.

d. Explica que imprime el programa y por qué.

El programa imprime el carácter “@” ya que lo que hace es sumar 5 + 11 y el resultado es 16, que se pasa a ascii.

```
Resultado: @
```

e. Modifica el programa para que imprima lo siguiente: A, \, $, & y 1. Documenta tu procedimiento.

```
section .data
chars db 'A', '\\', '$', '&', '1'    ; caracteres deseados (nota: '\' → '\\')
msg db 'Caracteres: ', 0

section .bss
buffer resb 1

section .text
global _start

_start:
    ; Mostrar encabezado
    mov eax, 4
    mov ebx, 1
    mov ecx, msg
    mov edx, 12       ; longitud real de "Caracteres: "
    int 0x80

    ; Imprimir los caracteres uno por uno
    mov esi, chars
    mov ecx, 5        ; número de caracteres

print_loop:
    mov al, [esi]     ; cargar carácter
    mov [buffer], al  ; guardarlo en buffer

    mov eax, 4
    mov ebx, 1
    mov ecx, buffer
    mov edx, 1
    int 0x80

    inc esi
    loop print_loop

    ; Salir
    mov eax, 1
    xor ebx, ebx
    int 0x80
```

f. ¿Fue la única forma de modificar el código? ¿Qué otra línea se pudo modificar?

No, no fue la única forma.
Para obtener los caracteres deseados también se pudo haber modificado:

- el valor almacenado en num1 o num2,
- la instrucción add eax, 48, cambiando la conversión ASCII,
- o se pudo haber colocado directamente el valor ASCII deseado en el buffer antes de imprimirlo, sin necesidad de hacer una suma.

En resumen, había varias líneas que se podían editar para obtener los mismos resultados.

g. Modificar el código para usar direccionamiento inmediato e indirecto para imprimir ‘@’

Modo inmediato:
Consiste en cargar directamente el valor ASCII del carácter.
El carácter @ tiene ASCII 64, por lo que se puede poner directamente mov al, 64.

Modo indirecto:
Consiste en almacenar el carácter @ en memoria y acceder a él a través de un registro que guarda su dirección.
Por ejemplo, guardar ‘@’ en una variable y luego usar un registro como ESI o EBX para apuntar hacia ella y leerla indirectamente.

2. Preguntas teóricas

a. ¿Cómo afecta el modo de direccionamiento a la eficiencia de un programa en ensamblador?

El modo de direccionamiento afecta directamente el número de ciclos de reloj que requiere una instrucción. Los modos más rápidos son los que usan registros, mientras que los más lentos son los que acceden a memoria. Un modo de direccionamiento más simple suele producir un programa más rápido y compacto. Los modos más complejos pueden aumentar el tiempo de ejecución.

b. ¿Qué papel juegan los modos de direccionamiento en la programación de sistemas y controladores?

Los modos de direccionamiento permiten acceder de forma flexible a memoria, puertos de hardware, estructuras internas del sistema, buffers de entrada/salida y tablas de dispositivos. En programación de bajo nivel son esenciales porque permiten manipular datos y hardware de forma eficiente. Sin estos modos, los controladores y sistemas

operativos no tendrían la capacidad de modificar o leer áreas específicas de memoria o registros de dispositivos.