PROGRAMACION AVANZADA

DOCENTE: Domínguez Chaves José Alfonso

ALUMNOS: Rodrigo Guerrero Osorio

Axel Arturo Pérez montero

Alexis David Chama Lino

Karol Jesús Ramírez cruz

Fredy Yael Huesca Rentería

SEMESTRE: 4to semestre

Xalapa, ver. 14 de marzo de 2025

Introducción

Este programa en ensamblador de 16 bits implementa una secuencia básica de animación en una pantalla gráfica y muestra un mensaje en una pantalla de texto. Se ejecuta como un programa de arranque (boot), configurando la pila, escribiendo un texto en la memoria de la pantalla y animando un sprite en dos posiciones alternas. Este código demuestra conceptos fundamentales de manipulación de memoria, control de flujo y subrutinas en ensamblador de 16 bits, ideal para sistemas embebidos o simulaciones en entornos educativos.

Código

```
JMP boot

stackTop    EQU 0xFF     ; Puntero de pila
textDisplay EQU 0x2E0    ; Dirección de la pantalla de texto
gfxDisplay EQU 0x300     ; Dirección base de la pantalla gráfica

; Definir posiciones para la animación (dos frames)
frame0      EQU 0x300    ; Posición 1
frame1      EQU 0x304    ; Posición 2 (desplazada 4 bytes a la derecha)

hello:  ¡DB “Hola profe!"
        DB 0         ; Terminador de cadena

; Sprite de 4x4 (16 bytes totales)
Sprite:
   DB "\xFF\xFF\xFF\xFF\xFF\x1F\x1F\x1F"
        DB "\x1F\x1F\xFF\xFF\xFF\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\xFF\x1F\x1F\x1F\x1F"
        DB "\x1F\x1F\x1F\x1F\x1F\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\xFF\x8C\x8C\x8C\xF4"
        DB "\xF4\x8C\xF4\xFF\xFF\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\x8C\xF4\x8C\xF4\xF4"
        DB "\xF4\x8C\xF4\xF4\xF4\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\x8C\xF4\x8C\x8C\xF4"
        DB "\xF4\xF4\x8C\xF4\xF4\xF4\xFF\xFF"
```


```
        DB "\xFF\xFF\xFF\x8C\x8C\xF4\xF4\xF4"
        DB "\xF4\x8C\x8C\x8C\x8C\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\xFF\xFF\xF4\xF4\xF4"
        DB "\xF4\xF4\xF4\xF4\xFF\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\xFF\x8C\x8C\xC4\x8C"
        DB "\x8C\x8C\xFF\xFF\xFF\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\x8C\x8C\x8C\xC4\x8C"
        DB "\x8C\xC4\x8C\x8C\x8C\xFF\xFF\xFF"
        DB "\xFF\xFF\x8C\x8C\x8C\x8C\xC4\xC4"
        DB "\xC4\xC4\x8C\x8C\x8C\x8C\xFF\xFF"
        DB "\xFF\xFF\xF4\xF4\x8C\xC4\x00\xC4"
        DB "\xC4\x00\xC4\xC4\xF4\xF4\xFF\xFF"
        DB "\xFF\xFF\xF4\xF4\xF4\xC4\xC4\xC4"
        DB "\xC4\xC4\xC4\xF4\xF4\xF4\xFF\xFF"
        DB "\xFF\xFF\xF4\xF4\xC4\xC4\xC4\xC4"
        DB "\xC4\xC4\xC4\xC4\xF4\xF4\xFF\xFF"
        DB "\xFF\xFF\xFF\xFF\xC4\xC4\xC4\xFF"
        DB "\xFF\xC4\xC4\xC4\xFF\xFF\xFF\xFF"
        DB "\xFF\xFF\xFF\x00\x00\x00\xFF\xFF"
        DB "\xFF\xFF\x00\x00\x00\xFF\xFF\xFF"
        DB "\xFF\xFF\x00\x00\x00\x00\xFF\xFF"
        DB "\xFF\xFF\x00\x00\x00\x00\xFF\xFF"

boot:
        ; Inicializa la pila
        MOV SP, stackTop
```


```
    ; Imprime mensaje en la pantalla de texto
    MOV C, hello   ; C apunta a la cadena "Hola Mundo!"
    MOV D, textDisplay; D apunta al inicio de la pantalla de texto
    CALL printText

    ; Realiza animación en la pantalla gráfica
    CALL drawSprite

    HLT            ; Detener ejecució

printText:
    PUSH A
    PUSH B
    MOV B, 0

pt_loop:
    MOVB AL, [C]    ; Leer carácter de la cadena
    CMPB AL, 0      ; ¿Terminador?
    JZ pt_done      ; Si es 0, fin del texto

    MOVB [D], AL    ; Escribir carácter en la pantalla
    INC C
    INC D
    JMP pt_loop
```


```
pt_done:
        POP B
        POP A
        RET


drawSprite:
        PUSH A
        PUSH B
        MOV B, 0        ; Contador de ciclos de animación

animLoop:
        ; --- Dibuja el sprite en frame0 ---
        MOV D, frame0   ; Selecciona posición frame0
        MOV C, sprite   ; C apunta al sprite
        MOV A, 0x10     ; 16 bytes a copiar
drawLoop0:
        MOVB AL, [C]    ; Leer byte del sprite
        MOVB [D], AL    ; Escribir en la pantalla gráfica
        INC C
        INC D
        DEC A
        JNZ drawLoop0

        CALL delay_ms   ; Retardo entre frames
```

```
    ; --- Dibuja el sprite en frame1 ---
    MOV D, frame1   ; Selecciona posición frame1
    MOV C, sprite   ; Reinicia C al inicio del sprite
    MOV A, 0x10     ; 16 bytes a copiar
drawLoop1:
    MOVB AL, [C]
    MOVB [D], AL
    INC C
    INC D
    DEC A
    JNZ drawLoop1

    CALL delay_ms   ; Retardo entre frames

    INC B           ; Incrementa contador de ciclos
    MOV A, B
    CMP A, 4        ; Realiza 4 ciclos de animación
    JNZ animLoop

    POP B
    POP A
    RET


delay_ms:
    PUSH A
```

```
        MOV A, 0xFF
delay_loop:
        DEC A
        JNZ delay_loop
        POP A
        RET
```

Desarrollo

1. Configuración del Entorno

El programa está diseñado para ejecutarse en un sistema de 16 bits con una memoria de video estructurada en dos regiones:

- **Pantalla de texto** (0x2E0): Se utiliza para mostrar mensajes de texto.
- **Pantalla gráfica** (0x300): Se usa para dibujar sprites en una animación básica.

Además, se define una ubicación en la memoria (0xFF) como el tope de la pila para la correcta gestión de subrutinas.

2. Inicialización del Programa

El código inicia configurando el puntero de pila (SP) y posteriormente ejecuta dos tareas principales:

1. **Mostrar un mensaje en la pantalla de texto.**
2. **Animar un sprite en la pantalla gráfica.**

```
MOV SP, stackTop; Inicializa la pila

MOV C, hello    ; Apunta a la cadena de texto

MOV D, textDisplay; Apunta a la pantalla de texto

CALL printText  ; Llamada a la subrutina de impresión


CALL drawSprite ; Inicia la animación del sprite

HLT             ; Finaliza la ejecución
```

3. Impresión de Texto

¡El mensaje “Hola profe!" se almacena en memoria y se imprime en la pantalla de texto. La subrutina printText realiza este proceso recorriendo cada carácter y copiándolo en la dirección de video.

Funcionamiento de printText

- C apunta al inicio del mensaje.
- D apunta a la pantalla de texto.

- Un bucle copia cada byte del mensaje hasta encontrar el terminador (0).

```
pt_loop:
    MOVB AL, [C]; Leer carácter
    CMPB AL, 0  ; ¿Es el fin del mensaje?
    JZ pt_done  ; Si es así, termina

    MOVB [D], AL; Escribir en la pantalla
    INC C       ; Avanzar en la cadena
    INC D       ; Avanzar en la pantalla
    JMP pt_loop ; Repetir hasta completar
```

4. Animación del Sprite

La animación consiste en alternar la posición de un sprite en la memoria gráfica (frame0 y frame1).

Estructura de la animación:

- El sprite de 4x4 píxeles (16 bytes) se copia en frame0.
- Se introduce un retardo (delay_ms).
- Se mueve el sprite a frame1.
- Se repite el proceso por cuatro ciclos.

```
animLoop:
    MOV D, frame0
    MOV C, Sprite
    MOV A, 0x10
drawLoop0:
    MOVB AL, [C]
    MOVB [D], AL
    INC C
    INC D
```


```
    DEC A
    JNZ drawLoop0

    CALL delay_ms; Retardo

    MOV D, frame1
    MOV C, Sprite
    MOV A, 0x10
drawLoop1:
    MOVB AL, [C]
    MOVB [D], AL
    INC C
    INC D
    DEC A
    JNZ drawLoop1

    CALL delay_ms

    INC B
    MOV A, B
    CMP A, 4  ; ¿Se han completado 4 ciclos?
    JNZ animLoop
```

5. Implementación del Retardo

El retardo se genera con un simple bucle de decremento, simulando una pausa en la ejecución:

```
delay_ms:
    PUSH A
```

```
    MOV A, 0xFF
delay_loop:
    DEC A
    JNZ delay_loop
    POP A
    RET
```

6. Finalización del Programa

Tras completar la animación, el programa se detiene (HLT), evitando que continúe la ejecución de instrucciones no deseadas.

7. Estrategias de Depuración

Describe cómo depurar programas en ensamblador, que suele ser un desafío debido a la falta de herramientas visuales avanzadas. Algunas estrategias útiles podrían incluir:

- **Uso de registros**: Inspeccionar valores clave en registros como SP, AX, o BX durante la ejecución para verificar el estado del programa.
- **Mensajes de depuración**: Agregar subrutinas que impriman valores intermedios en la pantalla de texto para verificar el flujo lógico.
- **Simuladores y emuladores**: Utilizar herramientas como DOSBox o emuladores de sistemas de 16 bits para ejecutar y observar el programa paso a paso.

8. Optimización del Código

Explora formas de optimizar el programa:

- **Reducir ciclos de ejecución**: Por ejemplo, el retardo actual (delay_ms) podría optimizarse mediante una instrucción de temporizador si el hardware lo soporta.
- **Reutilización de subrutinas**: Analiza cómo modificar el diseño para que las subrutinas como drawSprite y printText puedan manejar múltiples tamaños de datos dinámicamente.

9. Seguridad y Estabilidad

Aunque los programas en ensamblador suelen ejecutarse en entornos controlados, mencionar la importancia de manejar errores como desbordamientos de pila o lecturas/escrituras fuera de límites de memoria:

- **Protección de la pila**: Monitorear el uso de SP para evitar sobrescribir regiones críticas de memoria.
- **Validación de datos**: Implementar verificaciones para garantizar que las direcciones de memoria estén dentro del rango esperado antes de acceder a ellas.

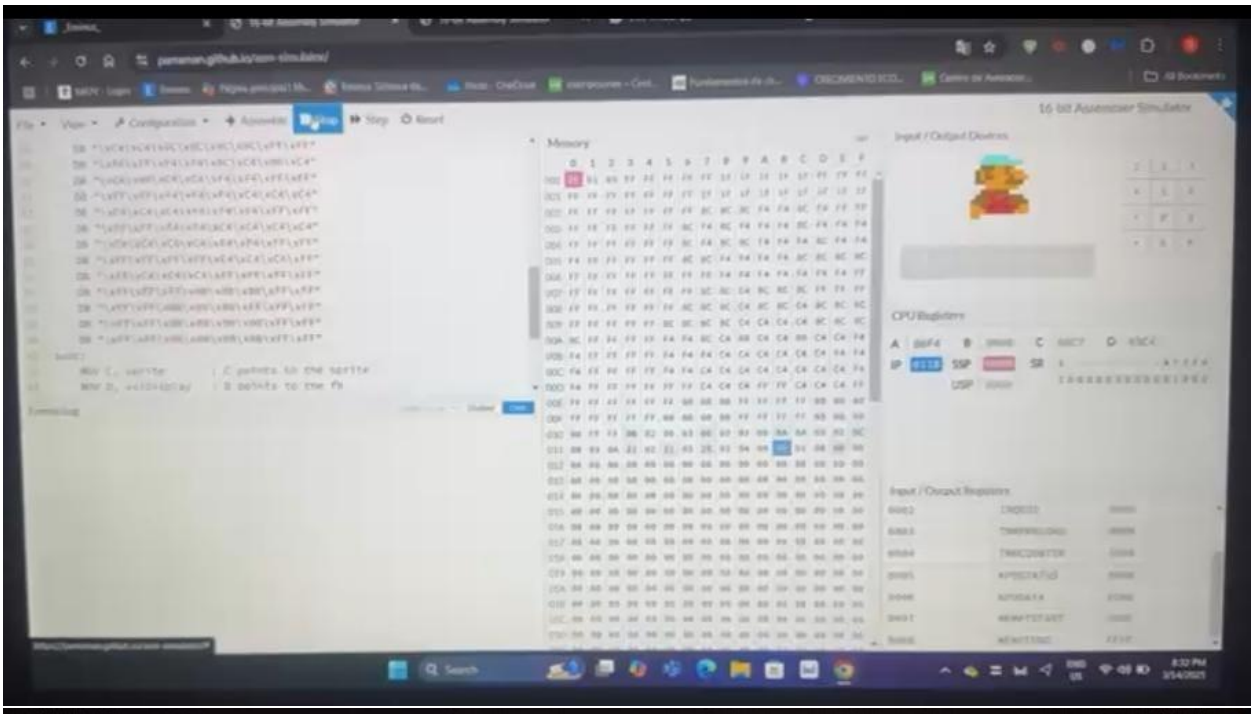
10. Posibles Extensiones del Programa

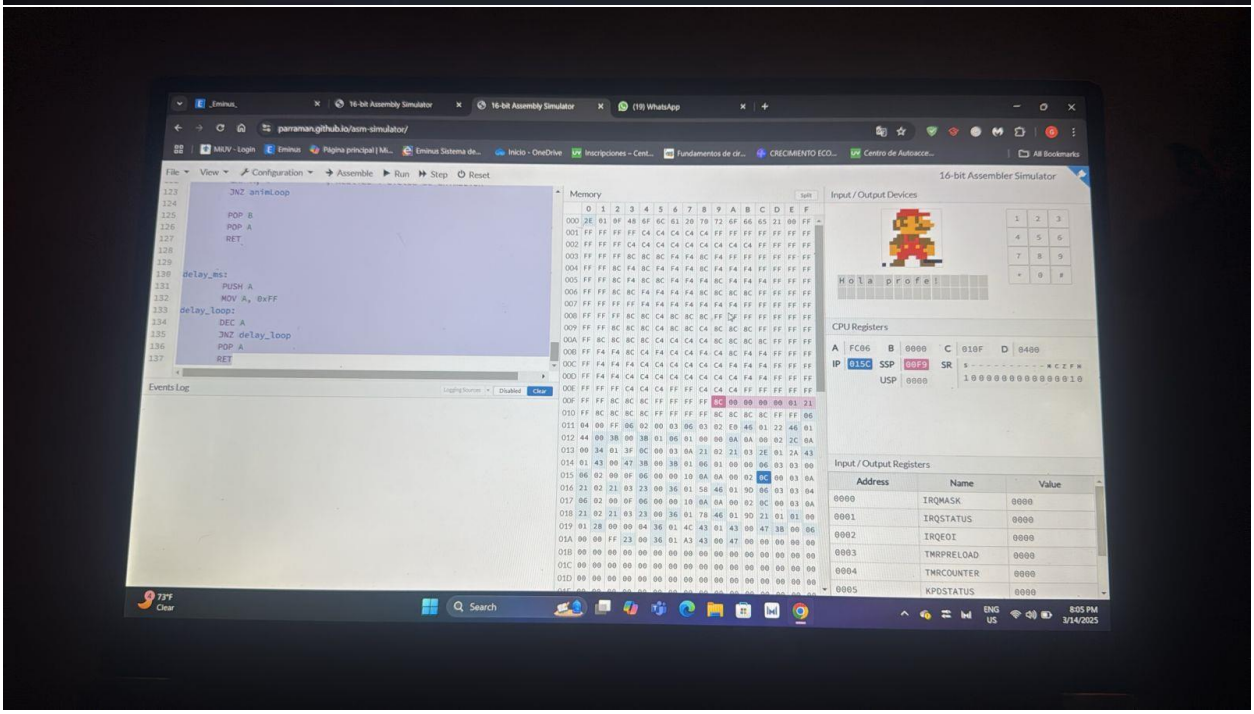
Para futuros desarrollos, podrías incluir:

- **Control de entrada del usuario**: Implementar detección de teclas para interactuar con la animación o cambiar el mensaje de texto.
- **Ampliación de gráficos**: Diseñar sprites más complejos o agregar colores (si el hardware lo permite).
- **Ciclos de animación dinámicos**: Permitir que el número de ciclos de animación se ajuste en tiempo de ejecución mediante una variable configurable.

FACULTAD DE INSTRUMENTACIÓN ELECTRÓNICA

Evidencias





Conclusión

Este programa en ensamblador de 16 bits ejemplifica los conceptos básicos de manipulación de memoria y control de flujo en un entorno educativo. Al configurar una pila, implementar una impresión de texto y desarrollar una animación gráfica simple, se logra una comprensión práctica del funcionamiento interno de sistemas de bajo nivel. La combinación de lógica secuencial, subrutinas reutilizables y animación básica proporciona una base sólida para el aprendizaje de programación en ensamblador. Este tipo de ejercicios refuerzan habilidades esenciales en diseño de sistemas embebidos, optimización de recursos y depuración de código en plataformas de hardware específicas.

Bibliografías

- Assembly Language Programming and Organization of the IBM PC. (1994). Ytha Yu & Charles Marut. McGraw-Hill.
- Irvine, K. R. (2010). *Assembly Language for x86 Processors* (6th ed.). Pearson Education.
- Stallings, W. (2014). *Computer Organization and Architecture: Designing for Performance* (10th ed.). Pearson.
- Ganssle, J. (2008). *The Art of Designing Embedded Systems* (2nd ed.). Newnes.
- Intel Corporation. (1994). *Intel 8086 Family User's Manual*. Intel.