



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

Ingeniería En Instrumentación Electrónica



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\xF\xF\xF"

DB "\xF\xF\xFF\xFF\xFF\xFF\xFF"

DB "\xFF\xFF\xFF\xFF\xF\xF\xF"

DB "\xF\xF\xF\xF\xF\xFF\xFF"

DB "\xFF\xFF\xFF\xFF\x8C\x8C\x8C\xF4"

DB "\xF4\x8C\xF4\xFF\xFF\xFF\xFF"

DB "\xFF\xFF\xFF\x8C\xF4\x8C\xF4\xF4"

DB "\xF4\x8C\xF4\xF4\xF4\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 ejecución

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
```

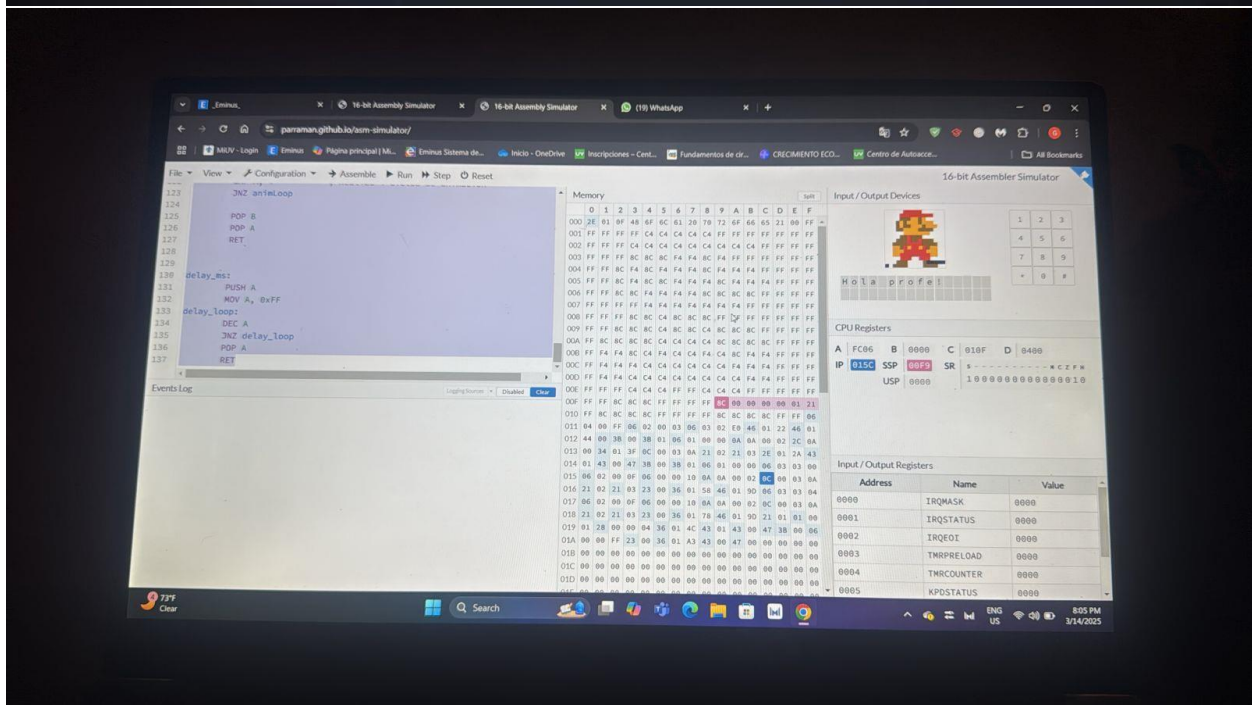
```
JNZ delay_loop
```

```
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.





Conclusión

Este programa en ensamblador de 16 bits demuestra el manejo de memoria, el control de flujo y la manipulación de dispositivos de salida en un entorno de bajo nivel. A través de la impresión de texto y la animación de un sprite, se logra una interacción básica con la pantalla de texto y la pantalla gráfica, simulando un sistema embebido o un entorno de desarrollo retro.

Uno de los principales logros del código es la implementación de subrutinas modulares como `printText`, `drawSprite` y `delay_ms`, las cuales permiten reutilizar código y mejorar la organización del programa. Gracias a esta estructura, se facilita la comprensión y el mantenimiento del código, haciendo posible su modificación o ampliación para incluir más funcionalidades, como el manejo de entrada del usuario o animaciones más complejas.

Otro aspecto relevante es el uso de la pila para almacenar valores temporales y preservar el estado de los registros al llamar subrutinas. Esta técnica es fundamental en programación en ensamblador, ya que garantiza que las llamadas a funciones no alteren valores importantes en el flujo principal del programa.

Sin embargo, el programa presenta algunas limitaciones. Por ejemplo, la función `delay_ms` utiliza un simple bucle de decremento que depende de la velocidad de ejecución del procesador. En una arquitectura real, este método podría no ser preciso y requeriría el uso de temporizadores de hardware para lograr pausas exactas. Además, la animación es relativamente básica, pues solo alterna entre dos posiciones sin interacción con otros elementos en pantalla.

En términos de aplicaciones prácticas, este código es un buen ejemplo de cómo se pueden construir sistemas de visualización en ensamblador para dispositivos con capacidades gráficas limitadas. Su estructura modular permite ampliarlo para agregar más cuadros de animación, mejorar la resolución de los sprites o incluso implementar un sistema de detección de colisiones.

En conclusión, este programa es una demostración clara de los fundamentos de programación en ensamblador, abordando conceptos clave como el acceso a memoria, la manipulación de registros y el uso de la pila. Su implementación proporciona una base sólida para desarrollar proyectos más avanzados en sistemas de bajo nivel, haciendo de este ejercicio una excelente práctica para comprender el funcionamiento interno de los procesadores y la memoria en sistemas embebidos o de propósito específico.