Manual de Laboratorio: Implementación en JDoodle (NASM 32-bit Linux)

Este manual didáctico ha sido diseñado con el propósito de guiar a los estudiantes en la implementación práctica de programas en lenguaje ensamblador utilizando la sintaxis de NASM para sistemas Linux de 32 bits, optimizados para el entorno interactivo de JDoodle. A través de este documento, se presentan estructuras de código fundamentales que sirven como pilares para el desarrollo de software de bajo nivel.

Temario y Códigos de Ejemplo

1. Ciclos Condicionales

- Objetivo: Ejecutar un bucle iterativo controlado mediante una condición lógica.
- Punto clave: Utiliza el registro ECX como contador regresivo y la instrucción JE para salir del ciclo al llegar a cero.

```
None
section .text
   global _start
_start:
   mov ecx, 5        ; Contador
ciclo:
   cmp ecx, 0
   je fin            ; Salta si es cero
   dec ecx
   jmp ciclo
fin:
   mov eax, 1
   int 0x80
```

Explicación rápida: El programa inicializa el registro ECX en 5. En cada iteración, se compara con 0; si es igual, salta a la etiqueta 'fin'. De lo contrario, decrementa ECX y repite el ciclo.

2. Incremento y Decremento

- Objetivo: Demostrar el uso correcto de contadores mediante la modificación directa de registros.
- Punto clave: Las instrucciones INC y DEC modifican el valor del registro especificado sumando o restando una unidad.

```
None
section .text
   global _start
_start:
   mov eax, 0
   inc eax            ; EAX ahora es 1
   dec eax            ; EAX ahora es 0
   mov eax, 1
   int 0x80
```

Explicación rápida: Este código muestra operaciones aritméticas básicas de un solo operando. Se incrementa EAX de 0 a 1 y luego se decrementa de nuevo a 0 antes de finalizar mediante la llamada al sistema.

3. Captura de Cadenas

- Objetivo: Leer una entrada de texto introducida por el usuario desde la entrada estándar (stdin).
- Punto clave: Utiliza la llamada sys_read (EAX=3) con el descriptor de archivo para stdin (EBX=0) y almacena el resultado en un búfer reservado.

```
None
section .data
   msg db 'Escribe algo: ', 0
section .bss
   buffer resb 20
section .text
   global _start
_start:
   mov eax, 3         ; sys_read
   mov ebx, 0         ; stdin
   mov ecx, buffer
   mov edx, 20
   int 0x80

   mov eax, 1
   int 0x80
```

Explicación rápida: El programa solicita una cadena de texto reservando espacio en memoria dinámica (sección .bss) e invocando la interrupción 0x80 con la configuración adecuada para capturar la entrada de la consola.

4. Instrucciones Aritméticas (Suma)

- Objetivo: Realizar operaciones aritméticas binarias fundamentales directamente sobre registros de propósito general.
- Punto clave: La instrucción ADD suma el operando de origen al operando de destino, almacenando el resultado definitivo en este último.

```
None
section .text
   global _start
_start:
   mov eax, 5
   add eax, 3        ; EAX = 8
   mov eax, 1
   int 0x80
```

Explicación rápida: Se carga el número 5 en el registro EAX y luego se utiliza ADD para adicionarle 3, resultando en un valor de 8 contenido en EAX antes del cierre del proceso.

5. Manipulación de la Pila (Stack)

- Objetivo: Almacenar de forma temporal y recuperar valores numéricos haciendo uso de la pila del sistema.
- Punto clave: Las instrucciones PUSH y POP administran la pila bajo el principio de LIFO (Last In, First Out).

```
None
section .text
   global _start
_start:
   mov eax, 10
   push eax          ; Guardar en pila
   pop ebx           ; Recuperar en EBX
   mov eax, 1
   int 0x80
```

Explicación rápida: El valor 10 de EAX se ingresa en la parte superior de la pila mediante PUSH, y posteriormente es extraído mediante POP para ser depositado de forma segura en el registro EBX.

6. Conversión a Decimal (Representación)

- Objetivo: Transformar valores numéricos internos a caracteres legibles para su correcta visualización en la consola estándar.
- Punto clave: Cada dígito numérico obtenido debe ser sumado con la constante decimal 48 (0x30) para realizar la conversión directa a su equivalente ASCII.

```
None
; Nota: Este proceso requiere divisiones sucesivas entre 10
; y el uso de la pila para invertir el orden de los dígitos.
```

Explicación rápida: Debido a que la salida estándar interpreta bytes individuales como caracteres ASCII, cualquier número compuesto debe descomponerse en sus dígitos elementales sumando 0x30 antes de imprimir.