Tarea Programada N°1

Emanuel Hernández Arauz

Desarrollo de una calculadora en modo texto
basado en una arquitectura x86 sobre un sistema operativo Linux

Facultad de Ingeniería

Escuela de Ciencias de la Computación e Informática

CI-0118 Lenguaje Ensamblador

Profesor. Ing. Sleyter Angulo Chavarría, M.Sc.

II Semestre 2024

September 12, 2024

1 Diseño de diagrama de flujo

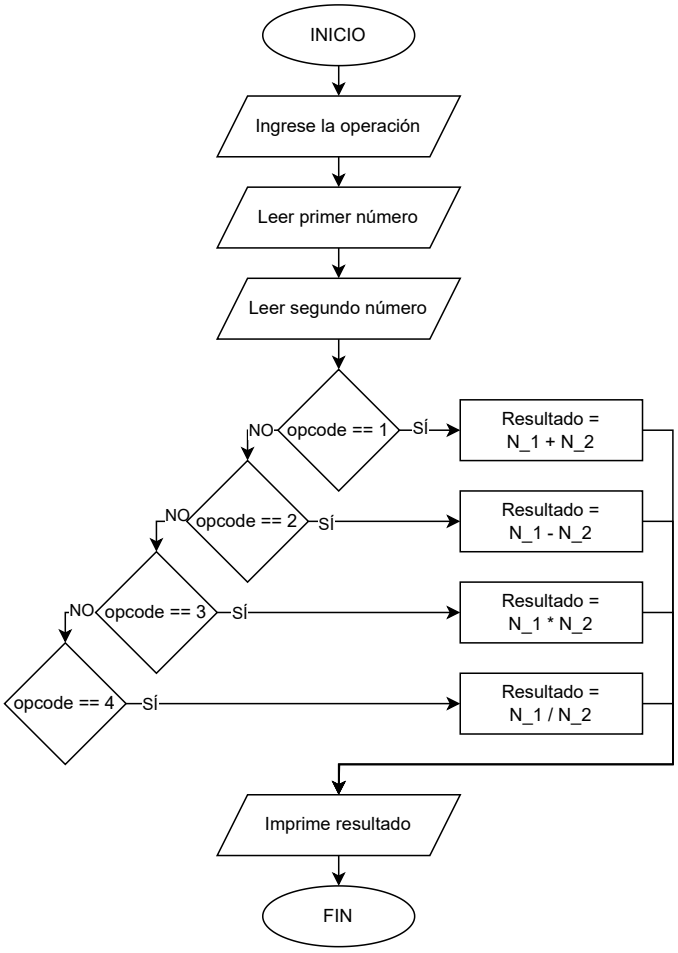

Figure 1: Diagrama de flujo de la calculadora

2 Implementación

```
section .text
    global _start

_start:
    ; Solicitud de op
    mov ecx, msg_start
    mov edx, msg_start_len
    call _print
    call _read
    xor eax, eax
    call _str_to_int
    mov [opcode], eax

    ; Solicitud de #1
    mov ecx, msg_a_in
    mov edx, msg_a_in_len
    call _print
    call _read
    xor eax, eax        ; Limpia eax antes de hacer la conversion
    call _str_to_int
    mov [val1], eax

    ; Solicitud de #2
    mov ecx, msg_b_in
    mov edx, msg_b_in_len
    call _print
    call _read
    xor eax, eax        ; Limpia eax antes de hacer la conversion
    call _str_to_int
    mov [val2], eax

    ; Resolver operacion
    call _op_calc
    mov [result], eax   ; Guarda el resultado

    ; Salida del resultado
    mov ecx, msg_res
    mov edx, msg_res_len
    call _print
    mov eax, [result]           ; N -> EAX
    call _int_to_str            ; RESULT -> ECX, STR_LEN -> ESI
    mov byte [ecx + esi], 0x0A  ; Concatena "\n" al final
    inc esi                     ; buffer legth++
```

```
    mov byte [ecx + esi], 0     ; NUL al final del buffer
    mov edx, esi                ; STR_LEN calculado en _int_to_str
    call _print

    ; Llamada al sistema para salir con normalidad
    mov eax, 0x01   ; exit
    xor ebx, ebx    ; Codigo salida (0x0)
    int 0x80        ; Llamada al sistema


; LLamada al sistema para imprimir (REQ: ecx = msg, edx = length_msg)
_print:
    mov eax, 0x04   ; write
    mov ebx, 0x01   ; stdout
    int 0x80
    ret

; Llamada al sistema para entrada de datos (RET: buffer = STDIN_INPUT)
_read:
    mov eax, 0x03   ; read
    mov ebx, 0x00   ; stdin
    mov ecx, buffer ; buffer_dir
    mov edx, 0x10   ; ancho del buffer
    int 0x80
    ret

; Convierte los caracteres del buffer a numero entero
; (REQ: eax = 0, ecx = buffer | RET: eax = result)
_str_to_int:
    movzx ebx, byte [ecx]   ; Carga el caracter actual (ECX) en EBX
    cmp bl, 0x0A            ; Compara el caracter con '\n' (0x0A)
    je  eol                 ; Salta al retorno en caso de '\n'
    sub ebx, '0'            ; Resta '0' (0x30) al caracter para obtener el val. num.
    ; Concatenar resultado (eax = eax * 10 + ebx)
    imul eax, eax, 0x0A     ; Multiplicar EAX por 10
    add eax, ebx            ; Suma el ultimo digito extraido de la cadena
    inc ecx                 ; Avanza al siguiente caracter
    jmp _str_to_int

eol:
    ret
```

```
; Convierte un entero a una cadena de caracteres
; (REQ: eax = n, ecx = buffer | RET: esi = STR_LEN) --> *Clap* NICE!
_int_to_str:
    mov edi, 0x0A           ; Divisor x10 para extraer digitos del numero
    lea ebx, [ecx + 15]     ; Apunta ESI al final del buffer (16 bytes)
    mov byte [ebx], 0       ; Termina la cadena con NUL
    dec ebx                 ; Ajusta EBX para empezar a transcribir los digitos
    xor esi, esi            ; indice = 0

loop_i_t_s:
    xor edx, edx            ; Limpiar EDX antes de la division
    div edi                 ; Divide EAX entre 10, EAX = cociente, EDX = residuo
    add dl, '0'             ; Agrega '0' (0x30) al digito para convertirlo en caracter
    mov [ebx], dl           ; Guardar el caracter en el buffer
    dec ebx                 ; base--
    inc esi                 ; indice++ (STR_LEN)
    test eax, eax           ; Verifica si el cociente es 0
    jnz loop_i_t_s          ; De lo contrario continua el loop
    lea ecx, [ebx + 1]      ; Ajusta ECX para que apunte al inicio de la cadena
    ret

; Verifica el codigo de operacion y hace el salto acorde
_op_calc:
    mov eax, [opcode]
    cmp eax, 0x01
    je _add
    cmp eax, 0x02
    je _sub
    cmp eax, 0x03
    je _mul
    cmp eax, 0x04
    je _div
    ret

; Suma los numeros (RET: eax = result)
_add:
    mov eax, [val1]
    mov ebx, [val2]
    add eax, ebx
    ret
```

```
; Resta los numeros (RET: eax = result)
_sub:
    mov eax, [val1]
    mov ebx, [val2]
    sub eax, ebx
    ret

; Multiplica los numeros (RET: eax = result)
_mul:
    mov eax, [val1]
    mov ebx, [val2]
    mul ebx
    ret

; Divide los numeros (RET: eax = result)
_div:
    mov eax, [val1]
    mov ebx, [val2]
    div ebx
    ret

section .data
    msg_start db 'Ingrese el numero con la operacion que desea realizar', 0x0A, \
                '1. SUMAR',         0x0A,  \
                '2. RESTAR',        0x0A,  \
                '3. MULTIPLICAR',   0x0A,  \
                '4. DIVIDIR',       0x0A
    msg_start_len equ $ - msg_start
    msg_a_in db 'Ingrese el primer numero', 0x0A
    msg_a_in_len equ $ - msg_a_in
    msg_b_in db 'Ingrese el segundo numero', 0x0A
    msg_b_in_len equ $ - msg_b_in
    msg_res db 'El resultado de la operacion es: '
    msg_res_len equ $ - msg_res

section .bss
    buffer  resb 0x10   ; Reserva 16 bytes para la entrada de datos
    opcode  resw 0x02   ; Reserva 4 bytes para guardar el # de operacion
    val1    resw 0x02   ; Reserva 4 bytes para guardar el primer operando
    val2    resw 0x02   ; Reserva 4 bytes para guardar el segundo operando
    result  resw 0x02   ; Reserva 4 bytes para guardar el resultado
```

3 Resultados

La manera en la que los datos le son desplegados al usuario, se realiza mediante consola, haciendo uso de las subrutinas *_print* e *_int_to_str*, antes mostradas en 2; además de las cadenas de texto pertenecientes a la sección ".data" y las variables "buffer" y "result", pertenecientes a la sección ".bss".

Lamentablemente por cuestiones de tiempo y de "salud mental", no me fue posible desarrollar las respectivas extensiones para desplegar los datos en los formatos hexadecimal, octal y binario.

4 Instrucciones para compilar/ejecutar el programa

Situado en la carpeta donde se encuentran los archivos del proyecto, se encuentra el Makefile encargado de simplificar el proceso de compilación del programa. Una vez estando en la terminal de linux, se proceden a ejecutar las siguientes instrucciones según se necesite

```
1. Compilación y enlace:
make

2. Ejecución:
./calc

3. Limpieza de archivos *.o
make clean
```