

Universidad Nacional de Cuyo
Facultad de Ingeniería
Licenciatura en Ciencias de la Computación

TRABAJO PRÁCTICO N^o 2

LENGUAJE ENSAMBLADOR

2024

Gonzalo Padilla

Marzo 2024

1. a) Programa que sume dos datos

```
MOV A, 128 ; Colocamos 128 en el registro A
ADD A, 42  ; Le sumamos 42
HLT
```

El resultado queda en el registro A.

1. b) Programa que realice la suma y la resta con dos datos almacenados en memoria

```
        JMP start          ; Saltar al programa principal

numA:   DB 128              ; Sumando A
numB:   DB 42               ; Sumando B
suma:   DB 0                ; Suma
resta:  DB 0                ; Resta

start:   ; Programa principal
        MOV A, [numA]       ; Cargamos numA en A
        ADD A, [numB]       ; Le sumamos numB
        MOV [suma], A       ; Movemos el resultado a suma
        MOV A, [numA]       ; Cargamos numA en A
        SUB A, [numB]       ; Le restamos numB
        MOV [resta], A      ; Movemos el resultado a resta
        HLT
```

El resultado de la suma queda en la dirección 0x04, y el de la resta en 0x05.

1. c) Escribir un programa que compare dos números. Si son iguales el programa debe finalizar y si son distintos los debe sumar

```
        JMP start          ; Saltar al programa principal

numA:   DB 128              ; Sumando A
numB:   DB 42               ; Sumando B
suma:   DB 0                ; Resta

start:   ; Programa principal
        MOV A, [numA]       ; Cargamos numA en A
        CMP A, [numB]       ; Lo comparamos con numB
        JE end              ; Si son iguales, finalizamos
        ADD A, [numB]       ; Si no, los sumamos
        MOV [suma], A       ; Movemos el resultado a suma
end:     HLT
```

El resultado de la suma queda en la dirección 0x04.

1. d) Un programa que lea un dato e indique si es par o impar

```

    JMP start          ; Salta al programa principal

dato:                    ; Número a analizar
    DB 42
msgEven:                ; Mensaje si es par
    DB "Es par"
    DB 0
msgOdd:                 ; Mensaje si es impar
    DB "Es impar"
    DB 0

start:                  ; Programa principal
    MOV A, [dato]       ; Cargamos el número en el registro A
    MOV C, msgEven      ; Cargamos el mensaje "Es par"
    AND A, 0x01         ; Obtenemos el primer bit
    JZ print           ; Si es par, omitimos la siguiente instrucción
    MOV C, msgOdd       ; Si no lo es, cargamos el otro mensaje

print:                  ; Imprimimos el mensaje
    MOV D, 0xE8         ; Cargamos la dirección del output
.loop:
    MOV A, [C]          ; Movemos el caracter al output
    MOV [D], A          ; Incrementamos punteros
    INC C
    INC D
    CMP A, 0            ; Verificamos si es el final
    JNZ .loop           ; Si no lo es, repetimos
    HLT

```

El resultado se imprime en el output.

1. e) Programa que indique el funcionamiento del stack (pila).

Implemento un programa con funciones que pueden retornar valores y aceptar argumentos, mediante el uso del stack. El programa calcula el factorial de un número n , e imprime la fórmula $n! = n \times (n-1) \times \dots \times 2 \times 1 = \text{factorial}(n)$ en el output. Se tienen la función `main()`, la función `factorial(n)`, que es recursiva, y las funciones `printNo(n)` y `printChar(c)`, que imprimen un número y un caracter, respectivamente. La función `main()` hace llamados a todas las demás; y `factorial(n)` llama a `printNo(n)`, `printChar(c)`, y a sí misma por ser recursiva.

Antes de llamar a una función, se reserva en el stack el espacio suficiente para el valor de retorno, excepto si no retorna nada, y luego se agregan también sus argumentos, si los requiere. Cada función puede también crear variables locales en su porción del stack, para que no se pierdan sus datos al llamar a otra función.

```

    CALL main           ; Salta al programa principal
    HLT

output: DB 0xE8         ; Posición del cursor
dato:   DB 5            ; Dato

main:                  ; Programa principal
    DEC SP              ; Alocamos 1 byte para factorial(n)

```

```

    PUSH [dato]           ; Argumento 'n' al stack
    CALL printNo          ; Llamamos a printNo(n)
    MOV [SP+1], '!'       ; Argumento '!' al stack
    CALL printChar        ; Llamamos a printChar('!')
    MOV [SP+1], '='       ; Argumento '=' al stack
    CALL printChar        ; Llamamos a printChar('=')
    INC SP                ; Sacamos argumento del stack
    PUSH [dato]           ; Argumento 'n' al stack
    CALL factorial        ; Calculamos el factorial
    MOV [SP+1], '='       ; Argumento '=' al stack
    CALL printChar        ; Llamamos a printChar('=')
    INC SP                ; Sacamos argumento del stack
    CALL printNo          ; Llamamos a printNo(n!)
    INC SP                ; Liberamos memoria
    RET

factorial:                ; factorial(n) como función recursiva
    PUSH [SP+2]           ; Argumento 'n' al stack
    MOV A, [SP+1]         ; Argumento 'n' a registro A
    CMP A, 0              ; Comparamos 'n' con 0
    JZ base               ; Si 'n' == 0, saltamos a base
    CMP A, 1              ; Comparamos 'n' con 1
    JZ base               ; Si 'n' == 1, saltamos a base
    CALL printNo          ; Llamamos a printNo(n)
    MOV [SP+1], 'x'       ; Argumento 'x' al stack
    CALL printChar        ; Llamamos a printChar('x')
    MOV A, [SP+3]         ; Argumento 'n' a registro A
    DEC A                 ; Decrementamos 'n'
    PUSH A                ; Argumento 'n-1' al stack
    CALL factorial        ; Llamamos a factorial(n-1)
    INC SP                ; Sacamos argumento del stack
    MOV A, [SP+1]         ; Resultado de factorial(n-1) a A
    MOV B, [SP+3]         ; Argumento 'n' a B
    MUL B                 ; Multiplicar n*(n-1)!
    MOV [SP+4], A         ; Resultado a valor de retorno
    INC SP                ; Liberamos memoria
    RET

base:
    MOV [SP+1], 1         ; Argumento '1' al stack
    CALL printNo          ; Llamamos a print(1)
    MOV [SP+4], 1         ; Si es cero, 1 a valor de retorno
    INC SP                ; Liberamos memoria
    RET

printNo:                  ; Imprime un número
    MOV D, [output]
    MOV B, [SP+2]
    MOV A, B
    DIV 100               ; Calculo dígito 1
    MOV C, A
    JZ skip1              ; Si dígito 1 es 0, saltamos
    ADD C, 48              ; Paso dígito 1 a ASCII
    MOV [D], C            ; Paso dígito 1 al output
    INC D                 ; Incremento cursor

skip1:
    MUL 100
    SUB B, A

```

```
MOV A, B
DIV 10          ; Calculo dígito 2
OR C, A
JZ skip2       ; Si dígitos 1 y 2 son 0, saltamos
MOV C, A
ADD C, 48      ; Paso dígito 2 a ASCII
MOV [D], C     ; Paso dígito 2 al output
INC D         ; Incremento cursor
skip2:
MUL 10
SUB B, A       ; Calculo dígito 3
ADD B, 48      ; Paso dígito 3 a ASCII
MOV [D], B     ; Paso dígito 3 al output
INC D         ; Incremento cursor
MOV [output], D ; Actualizo cursor
RET

printChar:     ; Imprime un caracter
MOV D, [output] ; Posición cursor a D
MOV C, [SP+2]  ; Caracter a C
MOV [D], C     ; Caracter al output
INC D         ; Incremento cursor
MOV [output], D ; Actualizo cursor
RET
```

2. a) Cargar números en las direcciones 60, 61, 62 y 63.
Restarle una constante (por ejemplo el, hexadecimal 7).
Transferir el resultado a las direcciones 70, 71, 72 y 73.

2. b) Cargar N números (por ejemplo 16) a partir de la dirección 60.
Terminar el ingreso de números, si ingresa un dato igual a 0.

2. c) Cargar N números a partir de la dirección 60.
Restarle una constante (por ejemplo el, hexadecimal 5).
Terminar el ingreso de números, si el resultado de la resta es cero.

2. d) Cargar la línea de memoria RAM desde la memoria 40 a la 4F con 16 datos y transferirlos a partir de la dirección de memoria 60.



2. e) Ejemplo de Hello World en español. Cambiar la salida por: Hola Mundo. ¿Qué tal? Explicar y/o comentar el programa en español.

```
; Escribe "Hola Mundo. ¿Qué tal?" en la salida

    JMP start
hello:
    DB "Hola Mundo. ¿Qué tal?" ; Variable
    DB 0                      ; Terminación de string

start:
    MOV C, hello      ; Puntero al string
    MOV D, 232        ; Puntero a la salida
    CALL print
    HLT               ; Detener ejecución

print:                ; print(C: *de, D: *hacia)
    PUSH A
    PUSH B
    MOV B, 0
.loop:
    MOV A, [C]        ; Obtener el caracter actual
    MOV [D], A        ; Escribirlo en la salida
    INC C              ; Incrementar punteros
    INC D
    CMP B, [C]        ; Verificar si es el final de la string
    JNZ .loop         ; Si no, volver al bucle

    POP B
    POP A
    RET
```