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

Trabajo Práctico Nº 2

Lenguaje Ensamblador

2024

Gonzalo Padilla

Marzo 2024

1. a) Programa que sume dos datos

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

printNo:            ; Lo pasamos a ASCII y lo imprimimos
    MOV D, 0xE8
    MOV B, A
    DIV 100
    MOV C, A
    ADD C, 48
    MOV [D], C
    MUL 100
    SUB B, A
    MOV A, B
    DIV 10
    MOV C, A
    ADD C, 48
    MOV [D+1], C
    MUL 10
    SUB B, A
    ADD B, 48
    MOV [D+2], B
    HLT
```

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

```
; 1.b) Sumar dos números almacenados en memoria

    JMP start       ; Salta al programa principal

datos:              ; Números a comparar
    DB 128
    DB 42

start:              ; Programa principal
    MOV D, datos    ; Cargamos la dirección del 1er número
    MOV A, [D]      ; Cargamos el primer número en el registro A
    ADD A, [D+1]    ; Le sumamos el segundo número
    PUSH A          ; Guardamos el resultado en el stack
    MOV A, [D]
    SUB A, [D+1]

bin2ascii:          ; Bonus: Lo pasamos a ASCII y lo imprimimos
    MOV D, 0xE8
    MOV B, A
    DIV 100
    MOV C, A
    ADD C, 48
    MOV [D], C
    MUL 100
    SUB B, A
```

```
    MOV A, B
    DIV 10
    MOV C, A
    ADD C, 48
    MOV [D+1], C
    MUL 10
    SUB B, A
    ADD B, 48
    MOV [D+2], B
    HLT
```

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

```
; 1.c) Comparar dos números

        JMP start         ; Salta al programa principal

datos:                        ; Números a comparar
        DB 42
        DB 128

start:                        ; Programa principal
        MOV D, datos        ; Cargamos la dirección del 1er número
        MOV A, [D]        ; Cargamos el primer número en el registro A
        CMP A, [D+1]        ; Comparamos con el segundo número
        JE end                ; Si son iguales, finaliza el programa
        ADD A, [D+1]        ; Si no, los sumamos

bin2ascii:                ; Bonus: Lo pasamos a ASCII y lo imprimimos
        MOV D, 0xE8
        MOV B, A
        DIV 100
        MOV C, A
        ADD C, 48
        MOV [D], C
        MUL 100
        SUB B, A
        MOV A, B
        DIV 10
        MOV C, A
        ADD C, 48
        MOV [D+1], C
        MUL 10
        SUB B, A
        ADD B, 48
        MOV [D+2], B
        HLT

end:
        HLT                ; Fin del programa
```

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

```
; 1.d) Es par o impar

        JMP start         ; Salta al programa principal

dato:                       ; Número a analizar
        DB 41
msgEven:
        DB " es par"        ; Mensaje si es par
        DB 0
msgOdd:
        DB " es impar"       ; Mensaje si es impar
        DB 0

start:                       ; Programa principal
        MOV A, [dato]       ; Cargamos el número en el registro A
        PUSH A
        CALL printNo        ; Lo imprimimos
        POP A
        MOV D, msgEven
        AND A, 0x01         ; Obtenemos el primer bit
        JZ ifEven          ; Si es par, salteamos la siguiente instrucción
        MOV D, msgOdd
ifEven:
        CALL printMsg
        HLT


printNo:                ; Bonus: Lo pasamos a ASCII y lo imprimimos
        MOV D, 0xE8
        MOV B, A
        DIV 100
        MOV C, A
        ADD C, 48
        MOV [D], C
        MUL 100
        SUB B, A
        MOV A, B
        DIV 10
        MOV C, A
        ADD C, 48
        MOV [D+1], C
        MUL 10
        SUB B, A
        ADD B, 48
        MOV [D+2], B
        RET

printMsg:
        MOV C, 0xE8
        ADD C, 3
.loop:
        MOV A, [D]          ;
        MOV [C], A          ;
        INC C
        INC D
```

```
        CMP A, 0        ;
        JNZ .loop        ;
        RET
```

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

```
; 1.e) Programa que indica el funcionamiento del stack

        CALL main       ; Salta al programa principal
        HLT

output:       DB 0xE8
dato:       DB 5
equal:       DB " = "
        DB 0
exclm:       DB "!"
        DB 0
times:       DB " × "
        DB 0


main:                       ; Programa principal
        SUB SP, 4       ; Alocamos 4 bytes para string x
        PUSH [dato]
        CALL bin2ascii
        INC SP
        MOV A, SP
        ADD A, 1
        PUSH A
        CALL print
        INC SP
        PUSH exclm
        CALL print
        INC SP
        PUSH equal
        CALL print
        INC SP
        DEC SP              ; Alocamos 1 byte
        PUSH [dato]       ; Pasamos el argumento al stack
        CALL factorial      ; Calculamos el factorial
        INC SP              ; Sacamos argumento del stack
        SUB SP, 4       ; Alocamos 4 bytes
        PUSH [SP+5]       ; Pasamos el factorial como argumento
        CALL bin2ascii      ; Convertimos el número a ASCII
        INC SP              ; Sacamos argumento del stack
        MOV A, SP
        ADD A, 1
        PUSH A
        CALL print       ; Imprimimos el resultado
        INC SP
        ADD SP, 9       ; Liberamos memoria
        RET


factorial:              ; Factorial como función recursiva
```

```
        DEC SP                  ; Alocamos 1 byte
        MOV A, [SP+3]           ; Argumento 'n' a registro A
        CMP A, 0            ; Comparamos argumento con 0
        JZ ifZero            ; Salto condicional
        DEC A                   ; Decrementamos A
        PUSH A                   ; Argumento 'n-1' al stack
        CALL factorial           ; Llamamos 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 A * !(A-1)
        MOV [SP+4], A           ; Resultado a valor de retorno
        INC SP                   ; Liberamos memoria
        RET
ifZero:
        MOV [SP+4], 1           ; Si es cero, retornamos 1
        INC SP                   ; Liberamos memoria
        RET


bin2ascii:                  ; Convierte de binario a ASCII
        MOV A, [SP+2]
        MOV B, A
        DIV 100
        MOV C, A
        ADD C, 48
        MOV [SP+3], C
        MUL 100
        SUB B, A
        MOV A, B
        DIV 10
        MOV C, A
        ADD C, 48
        MOV [SP+4], C
        MUL 10
        SUB B, A
        ADD B, 48
        MOV [SP+5], B
        MOV [SP+6], 0
        RET

print:
        MOV C, [SP+2]
        MOV D, [output]
loop:
        MOV A, [C]
        CMP A, 0
        JZ endloop
        CMP D, 0
        JNZ ahead
        MOV D, 0xE8
ahead:        MOV [D], A
        INC C
        INC D
        JMP loop
endloop:
        MOV [output], D
```

```
RET
```