

Condición mínima de aprobación: 2,5 puntos del Ejercicio 1, 1,5 puntos del Ejercicio 2 y 1,5 puntos del Ejercicio 3.

Ejercicio 1 (4 puntos)

Se pide implementar en arquitectura Intel 32 bits una calculadora por línea de comando. El modo de uso y resultado que muestra, responden al siguiente ejemplo:

```
/calculadora 5 + 15
Resultado = 20
```

Las operaciones que la calculadora aceptará son:

- suma (+, caso exemplificado arriba) • división (/)
- multiplicación (*) • potencia (^)

No es necesario que valide los números ingresados, tampoco el resultado.

Lo único que se le pide es una validación simple del operador matemático, que de no ser válido avisa, con un mensaje, el error:

```
/calculadora 5 A 5
Error: operador <A> no reconocido
Calculadora 5 * ABC 5
Error: operador <A*BC> no reconocido
```

Nota: lo valida como

un operador aceptado

y devuelve resultado

correcto

/calculadora 5 A*BC 5
Error: operador <A*BC> no reconocido

Se le pide:

- Desarrolle la función "main", en el archivo "calc.asm" que debe recibir los argumentos de la línea de comando, validar los operadores y llamar a la función operar()
- Se solicita finalizar la ejecución de su programa sin utilizar ninguna system call.
- La función "operar" ya se encuentra implementada y debe hacer uso de ella. No debe implementar más funciones en 'C'. Puede implementar cualquier otra función en assembler que requiera, utilizando su criterio.
- Ud. tiene el archivo de encabezado stdio.h cómo referencia del prototipo de 2 funciones que pueden ser útiles. Puede hacer uso de las 2 funciones de la librería estándar referidas (printf, puts).
- Recordatorio: el prototipo de la función "main" en 'C' es: int main(int argc, char *argv[]);

<p>I.T.B.A</p>	<p>Arquitectura de Computadoras Primer Parcial</p>	<p>2023</p>
<pre>calc.asm ;calculadora básica por Línea de comando</pre>	<pre>operar.c #include <stdlib.h> //para atoi() #include <math.h> //para pow() //implementación de función operar(): int operar(char *n1, char *n2, char *operador) { int i1=atoi(n1), i2=atoi(n2); switch(operador[0]) { case '+': return (i1+i2); case '-': return (i1-i2); case '/': return (i1/i2); case '^': return pow(i1,i2); default: return -1; //no debería } //dar caso default, puesto //que se valida operador</pre>	<pre>stdio.h //salida con formato en stdout. int printf(const char *fmt, arg1, arg2, ...); int puts(const char *str);</pre>

calc.asm

```
EXTERN printf
EXTERN operar
GLOBAL main
```

section .text

main:

```
push ebp          ; armo stack
mov  ebp, esp
```

```
push ebx
push ecx
push edx
```

```
mov  ebx, [ebp+8] ; argc
mov  ecx, [ebp+12] ; *argv[]
add  ecx, 4      ; salteo executable
```

```
mov  edx, [ecx+4] ; *operador
```

check_operador:

```
mov  al, byte [edx] ; primer char debe cumplir
cmp  al, '*'
je   .calc
cmp  al, '/'
je   .calc
cmp  al, '+'
je   .calc
```

```
cmp al, (-)
```

```
je .calc
```

```
jmp .error
```

.operar:

```
push edx ; *operador
mov edx, ecx ; *n1
add ecx, 4 ;
push ecx ; *n2
push edx ; *n1
call operar
add esp, 4*3 ; "libero" parametros
jmp .result
```

.error:

```
push edx ; *operador no valido
push error
call printf
add esp, 4*2
jmp .end
```

.result:

```
push eax ; cuenta hecha en operar
push resultado
call printf
add esp, 4*2
```

.end:

```
pop edx
pop ecx
pop ebx
```

```
mov eax, 0
```

```
mov esp, ebp
pop ebp
ret
```

section .data

```
error db "Error: operador <%s> no reconocido \n"
resultado db "Resultado=%d \n"
```

Ejercicio 2 (3 puntos)

Dado el siguiente programa en ASM que quiere ser compilado como programa principal y linkeditado con las dependencias necesarias, encuentre al menos 3 errores:

```

section .data
number db "%d", 10, 0 ; declaro el formato para imprimir

section .text
    global main
    extern printf ; Usamos la función standard printf

_start: declarado mal → debería decir main
    mov ecx, 9999 ; Mueve el número hasta el cual vamos a buscar
    palindromos
    ; este programa no puede usar números mayores a 9999
    empieza: falta;
    push ecx ; Guarda el tope en el stack ①
    mov ax, cx ; Lleva el número a AX
    mov bx, 10 ; Guarda 10 en BX para dividir por 10
    mov cx, 0 ; Reseteo CX para la próxima operación

```

```

pdigits:
    mov dx, 0 ; dividimos por 10 para averiguar la cantidad
    ; de dígitos del número
    div bx ; llevamos DX a 0 ,usaremos DX para guardar el módulo
    ; luego de la división
    push dx ; hacemos AX/ BX, AX contiene el número que
    ; estamos probando si es palíndromo
    inc cx ; Push DX, los dígitos a la derecha al stack
    ; Incrementamos CX , CX llevará la cuenta de la
    ; cantidad de dígitos
    cmp ax, 0 ; si es 0 ya llegamos
    je cont ; si es así vamos al rótulo 'cont'
    jmp pdigits ; sino el dígito de la derecha e incrementamos el contador

```

```

cont:
    cmp cx, 4 ; si el número contiene 4 dígitos
    je cuatro ; vamos a 'cuatro'
    cmp cx, 3 ; si el número contiene 3 dígitos
    je tres ; vamos a 'tres'
    cmp cx, 2 ; si el número contiene 2 dígitos
    je two ; vamos a 'dos'
    cmp cx, 1 ; si el número contiene 1 dígito
    je one ; vamos a 'uno'

```

cuatro:
llama un rótulo que no existe

ret start
%
9999
retso

} se va ~~se~~ luego de ejecutar

numero ya estaba en
stack en orden para
printf

mal des referencia puntero
cuando necesitas un puntero
no el primer char

mal porque al hacer
add sp, 8 ante su pila
"libero valores" de parametro
printf y ya no hay mas nada que el retso

0010 0111 0000 1111 ← ocupa 16 bits valido para cx

9
9
9
9
9999
ret so

```

pop ax = 9
pop bx = 9
pop cx = 9
pop dx = 9
cmp ax, dx
flag=1
jne nope
cmp bx, cx
jne nope
je yep

```

; el número tiene 4 dígitos
; recuperamos los 4 dígitos en AX, BX, CX y DX.
; o sea que si el número es por ej 1221 los registros serían
; AX->1, BX->2, CX->2 y DX->1.
; en ese caso si (AX == DX) y (BX == CX) entonces
; el número es palíndromo
; vamos a 'yep' sino a 'nope'
;

; en forma similar en 'tres' y 'dos'

```

tres:
pop ax
pop bx
pop cx
cmp ax, cx
jne nope
je yep

```

```

dos:
pop ax
pop bx
cmp ax, bx
jne nope
je yep

```

```

uno:
pop ax
jmp yep

```

; si el número es de un solo dígito se realiza un solo pop
; no hay necesidad de analizar ese valor ya que todos
; los números de un solo dígito son palíndromos
; se salta a 'yep'

```

nope:
pop ecx
dec ecx
siguiente ← ; falta
jnz empieza

```

; recuperamos ecx que es el valor actual
; lo decrementamos para continuar con el número
;

```

yep:
pop ecx
push ecx

```

; si es palíndromo recuperamos en ecx el valor original.
; push para usarlo como argumento a printf

```

push [number]
call printf
add sp, 8

```

; push del formato para printf
; Imprimimos
; Restauramos el stack pointer
;

* pop ecx
push ecx
dec ecx
jnz empieza

no es sintaxis de asm intel SP debería ser esp

; print cambio ecx , lo recuperamos
; push del número palíndromo.
; vamos al siguiente número
; seguimos

← mal pop libera
char de [number]
osea (%)

Ejercicio 3 (3 puntos) AEQ 32

Dado el siguiente programa compilado con GCC sin ningún parámetro adicional más que el archivo fuente:

```
#include<stdio.h>
```

```
int resta (int a) {
```

```
    int operando=9;
```

```
    return operando - a;
```

L.T.B.A

Arquitectura de Computadoras
Primer Parcial - TEMA 2

2023

```
}
```

```
int main() {
```

```
    int a = resta(resta(9));
```

```
    printf("Resultado: %d\n", a);
```

```
    return ;
```

a) Explique con dibujos respectivos el contenido de la pila, durante toda la ejecución del programa. Muestre los valores de los registros del microprocesador que intervienen en los dibujos de la pila.

b) Muestre los valores que retorna cada función y la forma en que lo hace.

c) Muestre la salida en pantalla del programa.

```
# include <stdio.h>

int resta ( int a) {
    int operando = 9
    return operando - a;
}

int main() {
    int a = resta (resta(9));
    printf (" Resultado: %d \n", a);
    return ;
}
```

①

operando
ebp
ret resta
9
canary
ebp
retso

gcc guarda lugar de más veces asume que lo hace ahora

resta(9)
interna se ejecuta antes de que asignarle lugar a "a" y la función externa resta

eax = operando - a
osea en este llamado eax = 0 xq 9-9=0

②

operando
ebp
ret main
0
canary
ebp
retso

resta(0) se ejecuta antes de la asignación de "a"

eax = operando - a
eax = 9

add esp,16
"libera" memoria de la función resta (resta(9))

③

a = 9
canary
ebp
retso

se genera la variable local a y se le asigna valor de eax = 9.

④

canary
ebp
retmain
LCB
a

printf → devuelve # de caracteres impresos con éxito.
eax = 12 ya que printf devuelve cantidad de caracteres impresos con éxito
"(Resultado: 9 \n)"
12345678910" 12

⑤

add esp, 20
para liberar printf.

canary
ebp
retso

esp=retso
y se libera espacio de ocupado por main

Se termina el programa pero como el return final esta vacio el compilador tira un warning despues de linkeditar cuando se correr

\$./ejecutable
Resultado: 9
\$