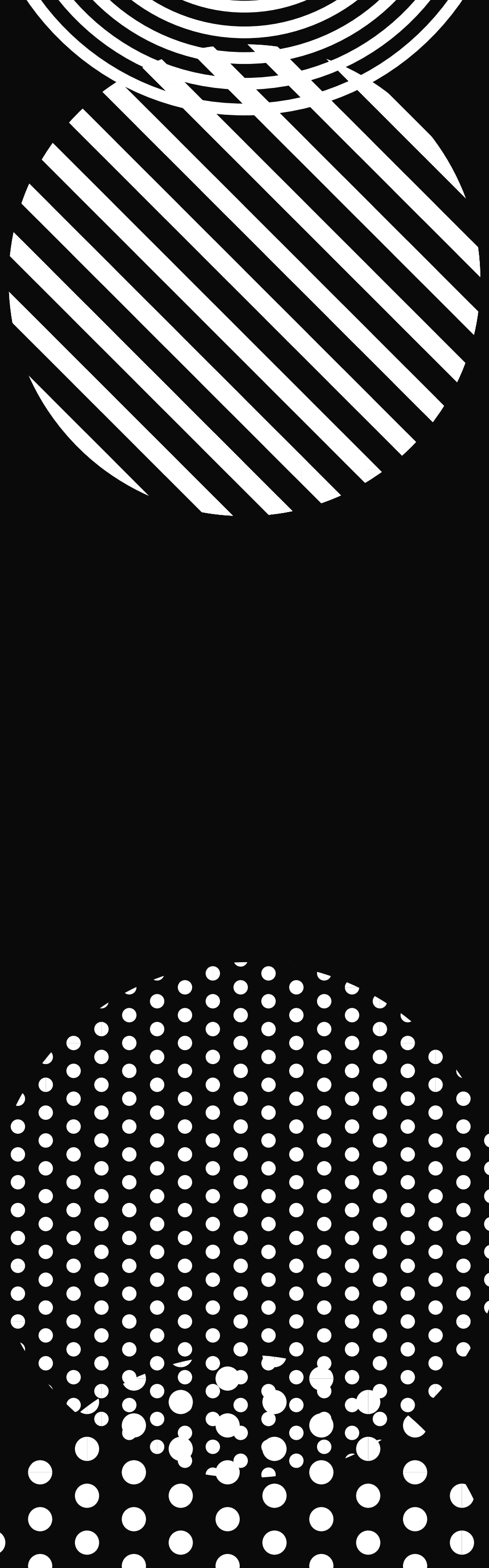


Primeros Parciales

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 ejemplificado arriba)
- multiplicación (*)
- división (/)
- 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   ----------> un operador aceptado
Resultado = 25                       y devuelve resultado
                                     correcto
./calculadora 5 A*BC 5
Error: operador <A*BC> no reconocido
```

Nota: lo valida como

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[]);

calc.asm

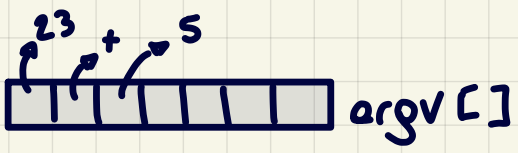
```
;calculadora básica por línea de
;comando
```

stdio.h

```
//salida con formato en stdout.
int printf(const char *fmt,
           arg1, arg2, ...);
int puts(const char *str);
```

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
```




```
global main          -> extern operar
section .text          extern printf
main:
    push ebp           ] Armado
    mov ebp, esp       ] Stack frame
    mov edx, [ebp+12]  ; Guardo argv[]
    mov ebx, [edx]     ; Guardo 1er operando
    inc edx            ; Avanzo
    mov ecx, [edx]     ; Guardo operador
    inc edx            ; Avanzo
    mov esi; [edx]     ; Guardo 2do operando
    push ecx
    call validar_operador
    cmp eax, 0
    jne .fin    * hacer
    push esi
    push ebx
    push ecx
   call operar         ; result en eax
   push eax
   push msg2
   call printf
.fin:
```

```
validar_operador:
    push ebp
    mov ebp, esp
    mov edi, [ebp+8]
    cmp edi, '+'
    jne .notOp
    cmp edi, '-'
    jne .notOp
    cmp edi, '*'
    jne .notOp
    cmp edi, '^'
    jne .notOp
    mov eax, 1
    jmp .finish
.notOp:
    mov eax, 0
    push edi
    push msg
    call printf
.finish:
    ret
```

```
  mov eax, 0
  mov esp, ebp
  pop ebp
  ret
section .data
msg db "Error: operador <%s> no reconocido", 10, 0
msg2 db "Resustado = %d", 10, 0
```

Ejercicio 3 (3 puntos)

Dada la siguiente salida en ASM:

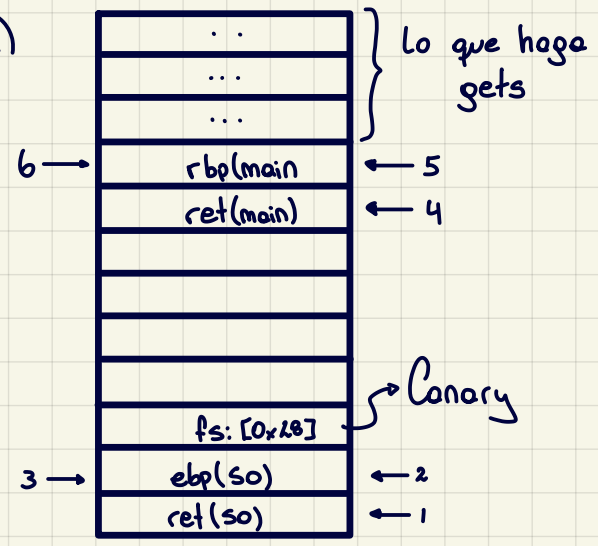
Se pide:

A. Dibuje la pila del programa para explicar en qué posición quedó guardado el Canary y el resto de los objetos que están en la pila del programa.
B. ¿Desde que posición de memoria fue obtenido el Canary ?
C. Explique el JE (jump equal) del final del programa.
D. Escriba un programa en lenguaje C cuya conversión a ASM coincida con el programa ASM del ejercicio.

I.T.B.A — Arquitectura de Computadoras — Primer Parcial

```
main:
0040059d  55              push   rbp
0040059e  4889e5          mov    rbp, rsp
004005a1  4883ec30        sub    rsp, 0x30 {var_38}
004005a5  64488b042528..  mov    rax, qword fs:[0x28]
004005ae  488945f8        mov    qword [rbp-0x8], rax
004005b2  31c0            xor    eax, eax
004005b4  488d45d0        lea    rax, [rbp-0x30 {var_38}]
004005b8  4889c7          mov    rdi, rax
004005bb  e8e0feffff      call   gets
004005c0  b800000000      mov    eax, 0x0
004005c5  488b55f8        mov    rdx, qword [rbp-0x8]
004005c9  6448331425289..  xor    rdx, qword fs:[0x28]
004005d2  7405            je     0x4005d9

004005d4  e897feffff      call   __stack_chk_fail
{ Does not return }

004005d9  c9              leave  {__saved_rbp}
004005da  c3              retn
```

a)



0x30 = 48d

b) Trae el Canary desde la posición 0x28

c) Al final, compara lo que hay en la pos. del stack donde guardó el canary con el canary (en la pos 0x28). Si son iguales, salta a la instrucción en 0x4005d9 que libera el stack y retorna al SO. Sino, no retorna y llama a la func. __stack_chk_fail.

d)

```
int main(){
    char buf[40];
    gets(*buf);
    return 0;
}
```

Chequear

Ejercicio 3 (3 puntos) ARQ 32

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

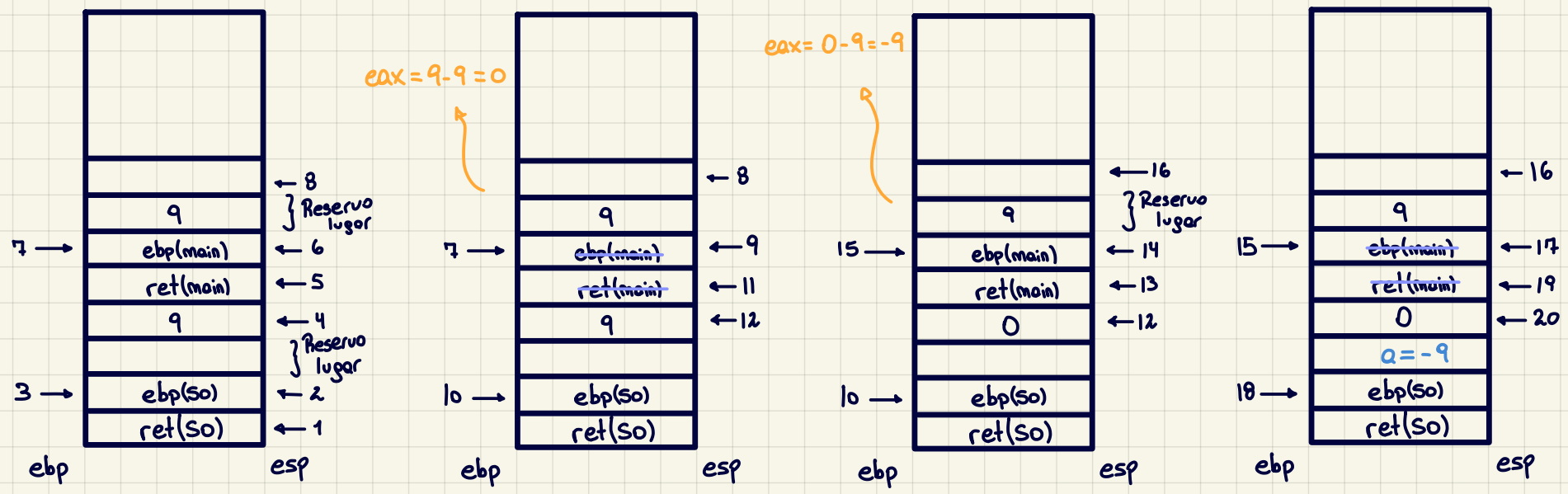
```
#include<stdio.h>

int resta (int a) {

    int operando=9;

    return operando - a;
}

int main() {

    int a = resta(resta(9));

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

    return ; }
```

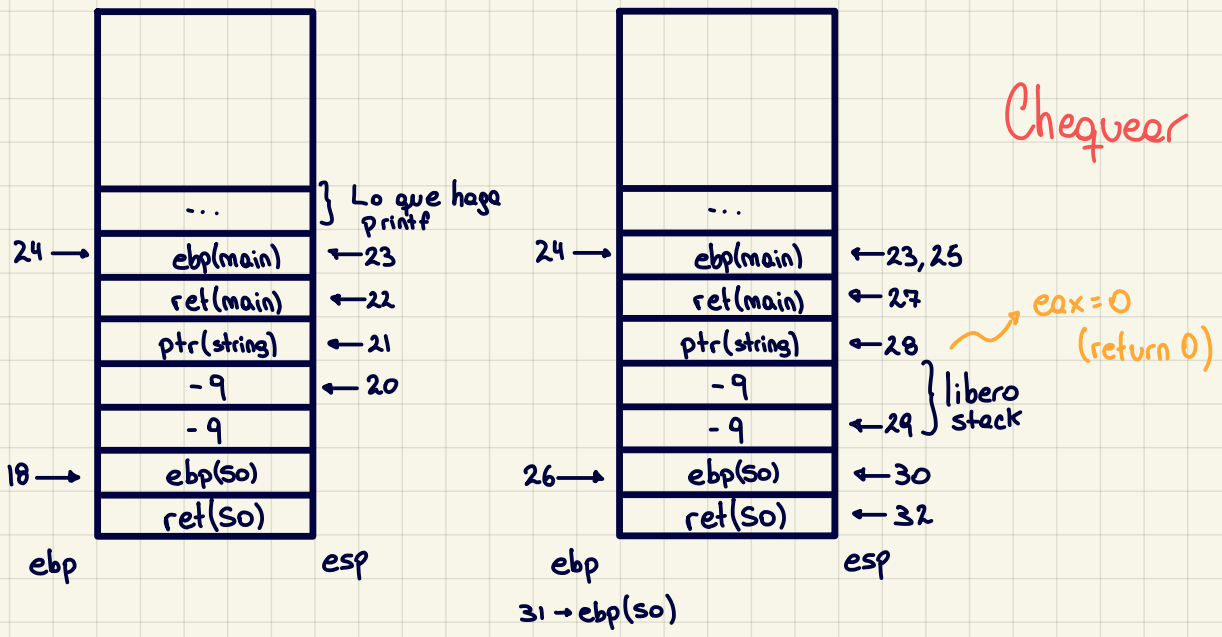
a) Explique con dibujos respectivos el contenido de la pila, durante toda la ejecución del programa. Muestre los valores de los registros del microprocesador que intervienen en los dibujos de la pila.

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

c) Muestre la salida en pantalla del programa.

Arquitectura de 32 bits.





Chequear

b) resta(9) → retorna 0 en eax

resta(resta(9)) → retorna -9 en eax

main() → retorna 0 en eax (creo)

a) "Resultado: -9".

Ejercicio 2 *(3 puntos)*

Dado el siguiente programa en ASM que quiere ser compilado cómo 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:                        → main:
          mov ecx, 9999        ; Mueve el número hasta el cual vamos a buscar
palíndromos
                               ; este programa no puede usar números mayores a 9999
empieza:
          push ecx             ; Guarda el tope en el stack
          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:                       ; dividimos por 10 para averiguar la cantidad
                               ; de dígitos del número
          mov  dx, 0           ; llevamos DX a 0 ,usaremos DX para guardar el módulo
                               ; luego de la división
          div  bx              ; hacemos AX/ BX, AX contiene el número que
                               ; estamos probando si es palindromo
          push dx              ; Push DX, los dígitos a la derecha al stack
          inc  cx              ; Incrementamos CX , CX llevará la cuenta de la
                               ; cantidad de dígitos.
          cmp  ax, 0           ; si es 0 ya llegamos
          je   cont            ; si es asi vamos al rótulo 'cont'
          jmp  pdigits         ; sino el digito 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:
```

I.T.B.A

Arquitectura de Computadoras 2023
Primer Parcial

```
          pop  ax          ; el número tiene 4 dígitos
          pop  bx          ; recuperamos los 4 dígitos en AX, BX, CX y DX.
          pop  cx          ; o sea que si el número es por ej 1221 los registros serian
          pop  dx          ;  AX->1, BX->2, CX->2 y  DX->1.
          cmp  ax, dx      ; en ese caso si (AX == DX) y  (BX == CX) entonces
                           ; el numero es palindromo
          jne  nope        ; vamos a 'yep' sino a 'nope'
          cmp  bx, cx      ;
          jne  nope        ; en forma similar en 'tres' y 'dos'
          je   yep

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          ; si el número es de un solo dígito se realiza un solo pop
          jmp  yep         ; 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         ; recuperamos ecx qu es el valor actual
          dec  ecx         ; lo decrementamos para continuar con el número
siguiente
          jnz  empieza     ; comenzamos de vuelta

yep:
          pop  ecx         ; si es palindromo recuperamos en ecx el valor original.
          push ecx         ; push para usarlo como argumento a printf

sin [ ]   push [number]    ; push del formato para printf
          call printf      ; Imprimimos
          add  esp, 4      ; Restauramos el stack pointer
esp (32 bits)

          pop  ecx         ; print cambio ecx , lo recuperamos
          push ecx         ; push del número palíndromo.
          dec  ecx         ; vamos al siguiente número
          jnz  empieza     ; seguimos
```

4, porque sino toma cualquier cosa

```
exit:
          mov  ax, 1       ; system call de exit
          mov  bx, 0
          int  80h
```

Ejercicio 1 *(4 puntos)*

Se le pide realizar un programa en assembler (Intel 32 bits) que haga lo siguiente:

- Leer la **humedad ambiente** cada 1 segundo. Use una **system call** para generar el tiempo de espera.
- **Ajuste el valor leído de humedad** según: H_correcta = H_leida/ 7
- Avisar en pantalla **si la humedad** leída en cada momento **coincide con algún valor en el arreglo** definido en el siguiente fragmento de código

```
section .data
arreglo         dd    10, 20, 30, 40, 50
cant_arreglo    dd    ($-arreglo)/4
```

(donde estoy parada - arreglo) / 4

Otra información que se le brinda

ID	syscall	ebx → tiempo solicitado	ecx → puede ser null
162	sys_nanosleep	const struct timespec *req	struct timespec *_Nullable rem
		ebp+4	ebp+8

struct timespec	Cantidad de segundos 4 bytes	Cantidad de nanosegundos 4 bytes

Cómo leer la humedad. Linux le permite a usted manejar la entrada analógica donde se lee la humedad ambiente, leyendo un archivo. A continuación se detalla cual es el archivo que debe utilizar:

```
/sys/bus/iio/in_voltage0_raw
```

Usted **no requiere programar la parte de manejo del archivo**: ya se le entrega una función escrita usando convenciones de C para pasaje de argumentos y retorno del valor:

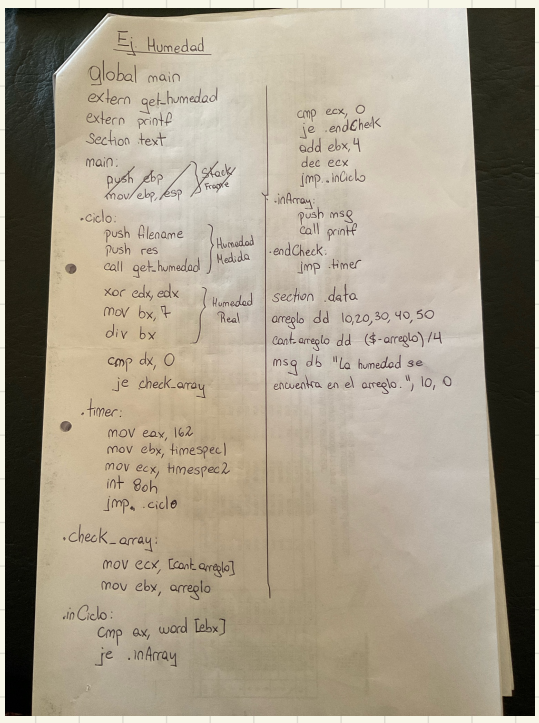
ebp+16 ebp+12

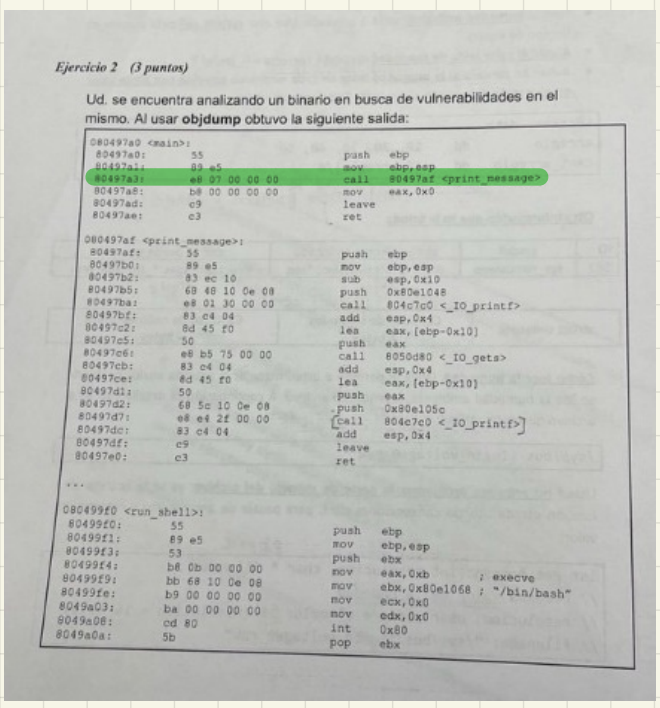
```
int get_humedad(int resolución, char * filename);
// retorna valor de la humedad
// resolución: usar siempre el valor 16 (resolución = 16 bits)
// filename: "/sys/bus/iio/in_voltage0_raw"
```

Las únicas funciones que puede usar son:

- get_humedad
- print: funciona exactamente igual que la función de librería estándar de C para intel 32.
- system calls: no hay restricciones

Si requiere alguna otra función, debe implementarla usted.

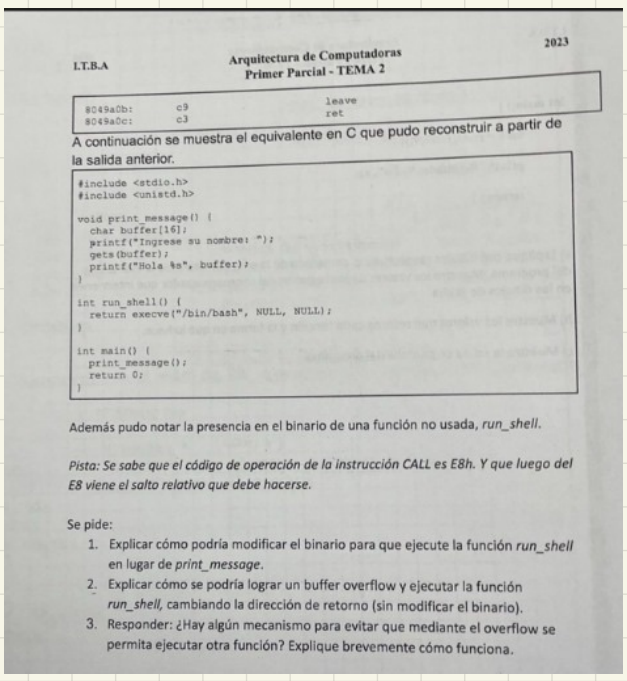




Ejercicio 2 *(3 puntos)*

Ud. se encuentra analizando un binario en busca de vulnerabilidades en el mismo. Al usar **objdump** obtuvo la siguiente salida:

```
080497a9 <main>:
 80497a9:   55                      push   ebp
 80497aa:   89 e5                   mov    ebp,esp
 80497ac:   e8 07 00 00 00          call   80497af <print_message>
 80497b1:   b8 00 00 00 00          mov    eax,0x0
 80497b6:   c9                      leave
 80497b7:   c3                      ret

080497af <print_message>:
 80497af:   55                      push   ebp
 80497b0:   89 e5                   mov    ebp,esp
 80497b2:   83 ec 10                sub    esp,0x10
 80497b5:   68 48 10 0e 08          push   0x80e1048
 80497ba:   e8 01 30 00 00          call   804c7c0 <_IO_printf>
 80497bf:   83 c4 04                add    esp,0x4
 80497c2:   8d 45 f0                lea    eax,[ebp-0x10]
 80497c5:   50                      push   eax
 80497c6:   e8 b5 75 00 00          call   8050d80 <_IO_gets>
 80497cb:   83 c4 04                add    esp,0x4
 80497ce:   8d 45 f0                lea    eax,[ebp-0x10]
 80497d1:   50                      push   eax
 80497d2:   68 5c 10 0e 08          push   0x80e105c
 80497d7:   e8 e4 2f 00 00          call   804c7c0 <_IO_printf>
 80497dc:   83 c4 04                add    esp,0x4
 80497df:   c9                      leave
 80497e0:   c3                      ret
...

080499f0 <run_shell>:
 80499f0:   55                      push   ebp
 80499f1:   89 e5                   mov    ebp,esp
 80499f3:   53                      push   ebx
 80499f4:   b8 0b 00 00 00          mov    eax,0xb        ; execve
 80499f9:   bb 68 10 0e 08          mov    ebx,0x80e1068 ; "/bin/bash"
 80499fe:   b9 00 00 00 00          mov    ecx,0x0
 8049a03:   ba 00 00 00 00          mov    edx,0x0
 8049a08:   cd 80                   int    0x80
 8049a0a:   5b                      pop    ebx
```



I.T.B.A — Arquitectura de Computadoras — Primer Parcial - TEMA 2 — 2023

```
8049a0b:   c9                      leave
8049a0c:   c3                      ret
```

A continuación se muestra el equivalente en C que pudo reconstruir a partir de la salida anterior.

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

void print_message() {
  char buffer[16];
  printf("Ingrese su nombre: ");
  gets(buffer);
  printf("Hola %s", buffer);
}

int run_shell() {
  return execve("/bin/bash", NULL, NULL);
}

int main() {
  print_message();
  return 0;
}
```

Además pudo notar la presencia en el binario de una función no usada, *run_shell*.

Pista: Se sabe que el código de operación de la instrucción CALL es E8h. Y que luego del E8 viene el salto relativo que debe hacerse.

Se pide:

1. Explicar cómo podría modificar el binario para que ejecute la función *run_shell* en lugar de *print_message*.
2. Explicar cómo se podría lograr un buffer overflow y ejecutar la función *run_shell*, cambiando la dirección de retorno (sin modificar el binario).
3. Responder: ¿Hay algún mecanismo para evitar que mediante el overflow se permita ejecutar otra función? Explique brevemente cómo funciona.

1. Vemos en la línea resaltado que el call en binario es:

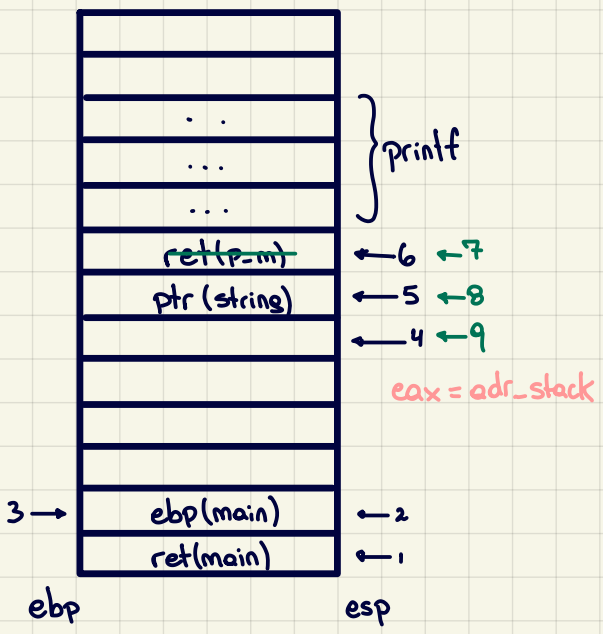
e8 07 00 00

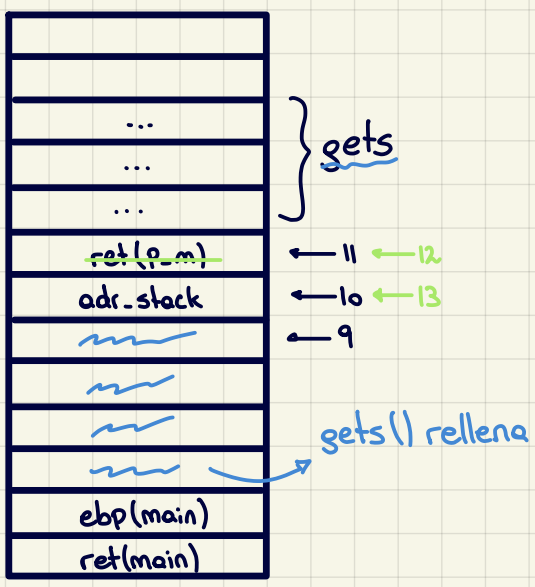
- 07 → salto relativo
- e8 → call

Por lo tanto, cambiando el salto relativo podemos correr run_shell en lugar de print_message

80499f0h - 80497a8h = 248h Entonces, modificamos e8 48 02 00 (Little Endian)

2. Armamos el stack:





Notemos que el lugar reservado en el stack es para 16 caracteres. Si ingresamos 24, vamos a pisar la dirección de retorno.

Queremos la dirección: 080499f0h

Entonces, pusheando 20 caracteres y luego ■♦ö≡ estaríamos reemplazando la direc. de retorno al main por la de run_shell, y lograríamos ejecutar dicho código.

3. Para evitar esto, los compiladores utilizan un número llamado canary. Este número se pushea a la pila y luego (a la hora de retornar) se compara lo que hay en dicho posición con el valor original. Si son iguales, no se pisó y entonces retorna. Si son distintos, no retorna y salta a otra función.