lenguaje ensamblador

- **Aritmética**

Instrucción NEG: Invierte todos los bits, posteriormente agrega (suma) un 1 y el resultado se guarda en el mismo operando.

- MOV EAX, 0xF049 ; EAX = 0xF049
- NEG EAX ; EAX = 0x0FB7 (0x0FB6 + 0x0001)

@truerand0m

lenguaje ensamblador

- **Comparaciones**

Instrucción CMP: Realiza la resta implícita (SUB) del origen al destino (CMP destino, origen), el resultado no se guarda en el destino pero cambia el estado de las banderas.

- MOV EAX, 0xF049 ; EAX = 0xF049
- MOV EBX, 0x05CA ; EBX = 0x05CA
- CMP EAX, EBX ; EAX = 0xF049 y EBX = 0x05CA

@truerand0m

lenguaje ensamblador

- **Comparación**

Instrucción TEST: Realiza la conjunción implícita (AND) bit a bit entre el origen y el destino (TEST destino, origen), el resultado no se guarda en el destino pero cambia el estado de las banderas.

- MOV EAX, 0xF049 ; EAX = 0xF049
- MOV EBX, 0x05CA ; EBX = 0x05CA
- TEST EAX, EBX ; EAX = 0xF049 y EBX = 0x05CA

@truerand0m

lenguaje ensamblador

- **Manipulación de bits**

Las instrucciones **SHR** y **SHL** son usadas para corrimiento de bits en los registros.

El formato para corrimiento a la derecha es **“SHR destino, contador”** y el formato para el corrimiento a la izquierda es **“SHL destino, contador”**.

@truerand0m

lenguaje ensamblador

- **Corrimientos**

Durante dichos corrimientos, el bit procesado se mueve a la bandera de acarreo (CF).

; Mueve el valor 0x0A (0000 1010 en binario) al registro BL

- MOV BL, 0x0A ; 0000 1010

; Corrimiento del registro BL de 2 bits a la izquierda, el resultado es 0010 10**00** (40d o 28h)

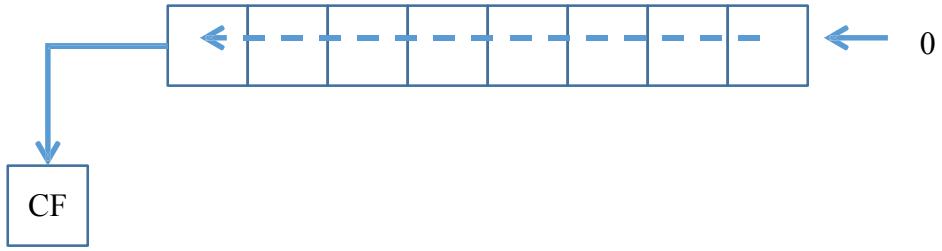
- SHL BL, 2

@truerand0m

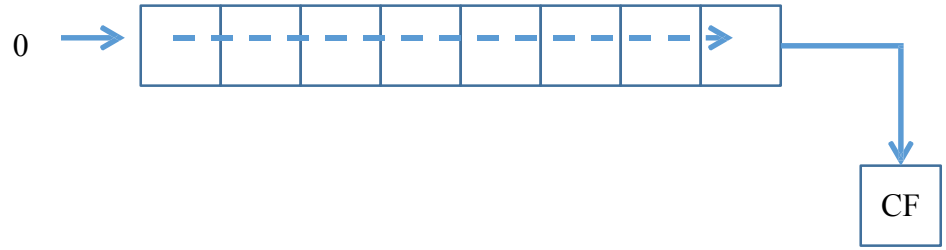
lenguaje ensamblador

- **Aritmética**

SHL



SHR



@truerand0m

lenguaje ensamblador

- **Manipulación de bits**

Las instrucciones de rotación **ROR** y **ROL** hacen que los bits **desplazados** regresen al otro extremo. En otras palabras, durante la rotación a la derecha (ROR) los bits menos significativos se rotan a la posición más significativa.

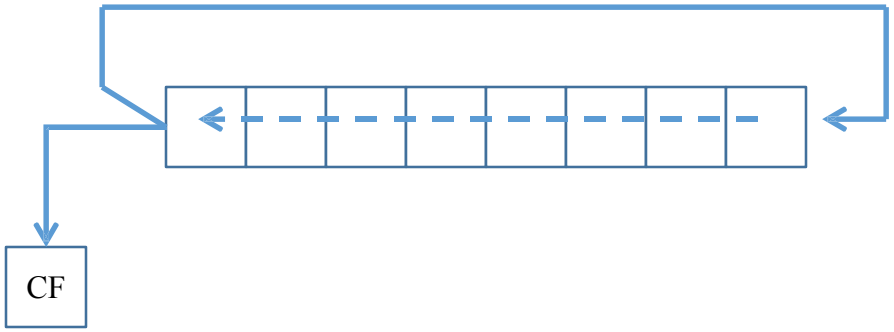
- MOV BL, 0x0A ; Mueve el valor 0x0A al registro BL :. BL = 0000 10**10**
- ROR BL, 2 ; Rotación del registro BL de 2 bits a la derecha .: BL = **10**00 0010

@truerand0m

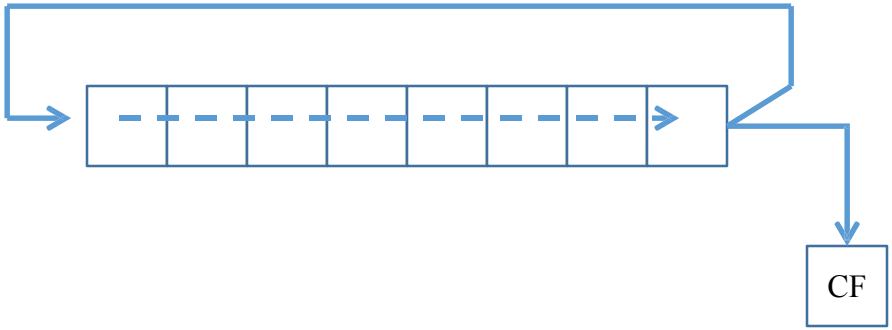
lenguaje ensamblador

- **Rotación**

ROL



ROR



@truerand0m

```
; shift arithmetic left:
; agrega un 0 en el bit menos significativo y el bit más significativo
; se desplaza a CF (si es 1) + ->01111111<- -
; CF = 0 111111110<-  nuevo cero ->
; la bandera al hacer shl no se activa y el número se duplica
section .text
global _start
_start:
    mov al,0b01111111
    shl al,1

    ; aqui si se activa CF
    mov al,0b11111111
    shl al,1

    ; shift right
    mov al,0b01111111
    shr al,1
```

@truerand0m

lenguaje ensamblador

- **Transferencia de datos**

Instrucción MOV: Asigna el origen al destino

(MOV destino, origen).

Direccionamiento inmediato.

- MOV EAX, 0xF049 ; EAX = 0xF049
- MOV EBX, 0x05CA ; EBX = 0x05CA
- MOV EAX, EBX ; EAX = 0x05CA y EBX = 0x05CA

@truerand0m

lenguaje ensamblador

- **Transferencia de datos**

Instrucción LEA *(Load Effective Address)*: Carga la dirección efectiva del operando origen dentro del operando destino (LEA destino, origen).

- LEA EAX, [EBP-0x04] ; dirección de la primera variable local

@truerand0m

lenguaje ensamblador

```
global _start
section .text
_start:
    ; | 1234 | <- celda 0x804a000 <-mem1    # i r $eax
    lea eax, [mem1]
    ; | 1234 | <- celda 0x804a000 <-mem1<-eax

    lea ebx, [eax]
    ; | 1234 | <- celda 0x804a000 <-mem1<-eax<-ebx
    mov eax, 1
    ; | 1234 | <- celda 0x804a000 <-mem1<-ebx
    ; eax = 1
    int 0x80
section .data
    mem1: dw 0x1234
```

@truerand0m

lenguaje ensamblador

- **Transferencia de datos**

La instrucción **LEA** no siempre es utilizada para referenciar direcciones de memoria, en ocasiones se usa para calcular valores debido a que requiere pocas instrucciones.

Por ejemplo, al factorizar el número 7 en la instrucción

“LEA EBX, [EAX*7+7]”

se obtiene la expresión

“EBX = (EAX+1) * 7”.

@truerand0m

lenguaje ensamblador

- **Transferencia de datos**

Instrucción XCHG: Intercambia el contenido del operando destino y con el de origen (XCHG destino, origen). Direccionamiento inmediato.

- MOV EAX, 0xF049 ; EAX = 0xF049
- MOV EBX, 0x05CA ; EBX = 0x05CA
- XCHG EAX, EBX ; EAX = 0x05CA y EBX = 0xF049

@truerand0m

lenguaje ensamblador

- **Transferencia de datos**

Instrucción PUSH: Almacena el operando en la pila
(PUSH operando). Direccionamiento inmediato.

- MOV EAX, 0xF049 ; EAX = 0xF049
- PUSH EAX ; Ultimo valor pila = 0xF049

@truerand0m

lenguaje ensamblador

- **Transferencia de datos**

Instrucción POP: Extrae y almacena el último valor almacenado en la pila en el operando (POP operando). Direccionamiento inmediato.

```
PUSH FOOD
PUSH BAAD
MOV EAX, 0xF00D     ; EAX = 0xFFFF
POP EAX             ; EAX = BAAD
                    ; Ultimo valor en la pila =  0xFOOD
```

@truerand0m

Programación en ensamblador

```
; prog1.asm
global _start
section .text
                    ; Linea requerida por ld
                    ; etiqueta que indica el
                    ; inicio de la sección
                    ; ejecutable
                        ; inicio del programa
_start:
      mov eax,0x04      ; eax = 0x04
      mov ebx,0x06      ; ebx = 0x06
      add eax,ebx       ; eax = eax + ebx
```

@truerand0m

```
section .text
global _start                    ; must be declared for linker (ld)
_start:                          ; entry point
    mov eax,0x1234               ; movs valores inmediatos
    mov ebx,0x5678
    mov edi,eax                  ; movs entre registros
    mov esi,ebx
    mov edi,mem1                 ; movs direcciones
    mov esi,mem2                 ; check x/x &mem1, content x/s &mem1

    mov [name],dword 'Done'      ; x/s 0x804...
    mov eax,1                    ; system call number (sys_exit)
    int 0x80                     ;call kernel

section .data
    name: db 'Test', 0xa
    mem1: db "hello",0xa
    mem2: db "world",0xa
```

@truerand0m

Programación en ensamblador

Ensamblar el código fuente

```
nasm -f elf -o prog1.o prog1.asm
```

¿Ejecutar?

```
./prog1.o
```

@truerand0m

Programación en ensamblador

¿Permission denied?

Enlazar el código objeto con las bibliotecas necesarias para producir el ejecutable.

```
ld -o prog1 prog1.o
```

¿Ejecutar?

```
./prog1
```

@truerand0m

Programación en ensamblador

¿Segmentation fault?

Solución: gdb

@truerand0m

Programación en ensamblador

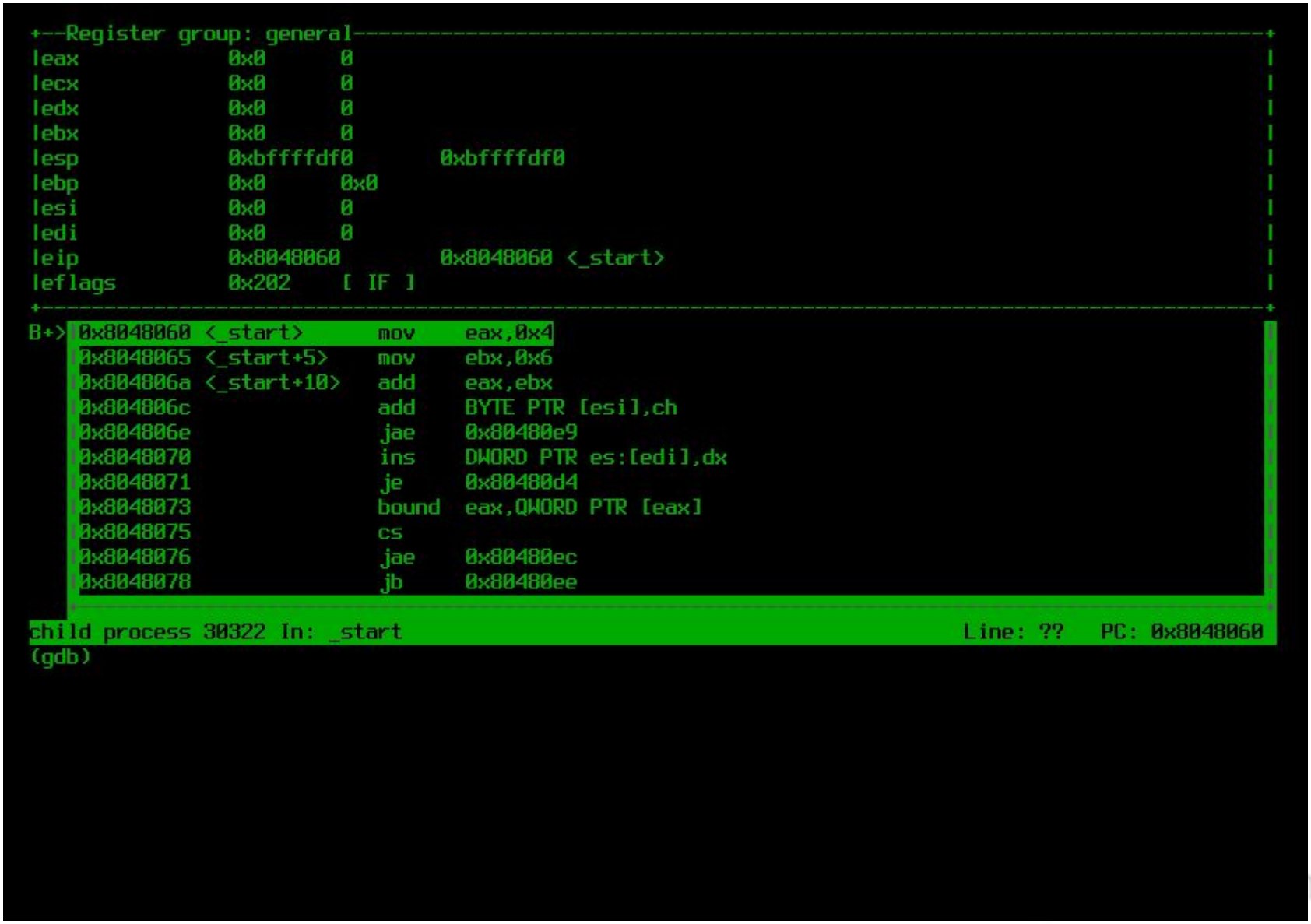
Utilizar GDB para ejecutar el programa en un entorno controlado.

```
gdb prog1
```

Dentro de GDB escribir los siguientes comandos:

```
break _start
run
set disassembly-flavor intel
layout asm
layout regs
```

@truerand0m


+--Register group: general
leax 0x0 0
lecx 0x0 0
ledx 0x0 0
lebx 0x0 0
lesp 0xbffffdf0 0xbffffdf0
lebp 0x0 0x0
lesi 0x0 0
ledi 0x0 0
leip 0x8048060 0x8048060 <_start>
leflags 0x202 [IF]
B+> 0x8048060 <_start> mov eax,0x4
0x8048065 <_start+5> mov ebx,0x6
0x804806a <_start+10> add eax,ebx
0x804806c add BYTE PTR [esi],ch
0x804806e jae 0x80480e9
0x8048070 ins DWORD PTR es:[edi],dx
0x8048071 je 0x80480d4
0x8048073 bound eax,QWORD PTR [eax]
0x8048075 cs
0x8048076 jae 0x80480ec
0x8048078 jb 0x80480ee
child process 30322 In: _start Line: ?? PC: 0x8048060
(gdb)

Programación en ensamblador

break _start: establece un breakpoint al comienzo del programa

run: ejecuta programa hasta el breakpoint

set disassembly-flavor intel: muestra código ASM en sintaxis intel

layout asm: despliega código asm

layout regs: despliega registros

@truerand0m

```
B+ 0x8048060 <_start>      mov    eax,0x4
   0x8048065 <_start+5>    mov    ebx,0x6
   0x804806a <_start+10>   add    eax,ebx
```

- Ejecutar stepi

- Ejecutar stepi

- Ejecutar stepi

@truerand0m

Saltos

- Son instrucciones utilizadas para dirigir el flujo de ejecución de un programa hacia una localidad de memoria (relativa o absoluta), comúnmente se especifica una etiqueta.

- Existen dos tipos de saltos: condicionales y no condicionales.

@truerand0m

Saltos no condicionales

- Siempre salta hacia otra porción de código en alguna dirección de memoria (JMP destino), por lo que no se necesita revisar el estado de las banderas.

JMP <destino>

Por ejemplo, las instrucciones JMP, CALL y RET.

@truerand0m

Saltos condicionales

- Utilizan el valor de las banderas (resultado de las operaciones con registros) para saltar a una nueva rama y transferirle el control si una condición se cumple.

@truerand0m

Saltos condicionales

- La estructura de un salto condicional es típicamente de la forma **JXX**, donde las letras "X" (que pueden ir de 1 a 4 letras) describen las siguientes condiciones:
  - Comparación general (Ej. Bandera Zero = 0, Bandera Signo = 1, etc.)
  - Comparación sin signo (ambos operadores)
  - Comparación con signo

  JXX <destino> ; donde XX es una condición a evaluar

@truerand0m

lenguaje ensamblador

- **Saltos**

```
MOV EAX, variable_a
CMP EAX, 0x08040200
JNE bloque_2

bloque_1:
     primer bloque de código
     JMP fin
bloque_2:
     segundo bloque de código
fin:
```

@truerand0m

lenguaje ensamblador

- **Saltos**

La estructura de un salto condicional es típicamente de la forma **JXX**, donde las letras “X” (que pueden ir de 1 a 4 letras) describen las siguientes condiciones:

- ❑ Comparación general
- ❑ Comparación sin signo
- ❑ Comparación con signo

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación general

- JZ: Salta si la bandera cero está activa (ZF = 1)
- JE: Salta si es igual (ZF = 1)
- JNZ: Salta si la bandera cero está deshabilitada (ZF = 0)
- JNE: Salta si no es igual (ZF = 0)
- JC: Salta si hubo acarreo (CF = 1)
- JNC: Salta si no hubo acarreo (CF = 0)

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación general

- JS: Salta si el número tiene signo (SF = 1)
- JNS: Salta si el número no tiene signo (SF = 0)
- JO: Salta si fue un desbordamiento (OF = 1)
- JNO: Salta si no fue un desbordamiento (OF = 0)

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación general

- JP: Salta si hubo paridad, número de bits habilitados par (PF=1)
  - JNP:Salta si no hubo paridad, número de bits habilitados impar (PF=0)
  - J**CX**Z: Salta si el registro CX tiene el valor 0.
  - J**ECX**Z: Salta si el registro ECX tiene el valor 0.

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación **sin signo** (ambos operadores)

- JA: (above) Salta si es mayor, **op1 > op2** (CF = 0 y ZF = 0)
- JNBE: Salta si no es menor o igual, **op1 > op2** (CF=0 y ZF=0)
- JAE: Salta si es mayor o igual, **op1 >= op2** (CF = 0)
- JNB: Salta si no es menor, **op1 >= op2** (CF = 0)

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación **sin signo** (ambos operadores)

- JB: (below) Salta si es menor, **op1 < op2** (CF = 1)
- JNAE: Salta si no es mayor o igual, **op1 < op2** (CF = 1)
- JBE: Salta si es menor o igual, **op1 <= op2**

  (CF = 1 y ZF = 1)
- JNA: Salta si no es mayor, **op1 <= op2** (CF = 1 y ZF = 1)

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación **con signo**

- JG: (greater) Salta si es mayor, **op1 > op2**
  (SF = OF y ZF = 0)
- JNLE: Salta si no es menor o igual, **op1 > op2**
  (SF=OF y ZF=0)
- JGE: Salta si es mayor o igual, **op1 >= op2** (SF = OF)
- JNL: Salta si no es menor, **op1 >= op2** (SF = OF)

@truerand0m

lenguaje ensamblador

- **Saltos**

Comparación **con signo**

- JL: (less) Salta si es menor, **op1 < op2** (SF != OF)
- JNGE: Salta si no es mayor o igual, **op1 < op2** (SF != OF)
- JLE: Salta si es menor o igual, **op1 <= op2**

  (SF != OF o ZF= 1)
- JNG: Salta si no es mayor, **op1 <= op2** (SF != OF o ZF = 1)

@truerand0m

```
section .text
global _start
_start:
        mov ecx,1 ; valor 1
        mov edx,2       ; valor 2
        cmp ecx,edx   ; if ecx == edx:
        je iguales       ; iguales ZF=1
        jmp distintos ; else  distintos
salir:
        mov ebx,1
        mov eax,1
        int 0x80
distintos:
        mov edx,len_dis   ; len a escribir
        mov ecx,msg_dis ; buffer
        mov ebx,1       ; salida estandar
        mov eax,4       ; llamada a imprimir
        int 0x80
        jmp salir
iguales:
        ; cantidad a escribir
        mov edx,len_equ
        mov ecx,msg_equ     ; buffer
        mov ebx,1     ; salida estandar
        mov eax,4     ; num de llamada a
imprimir
        int 0x80
        jmp salir

section .data:
        msg_dis: db 'son distintos',0xa
        len_dis: equ $ - msg_dis
        msg_equ: db 'son iguales',0xa
        len_equ: equ $ - msg_equ
```

@truerand0m

Procedimientos

- Un procedimiento es un conjunto de instrucciones delimitadas por una etiqueta indicando el comienzo de la misma y terminando con la instrucción RET.

```
Inicio_procedimiento:
        <Código>
RET
```

@truerand0m

Procedimientos

- El procedimiento es llamado desde otra función usando la instrucción CALL, teniendo como argumento la etiqueta al inicio del procedimiento.

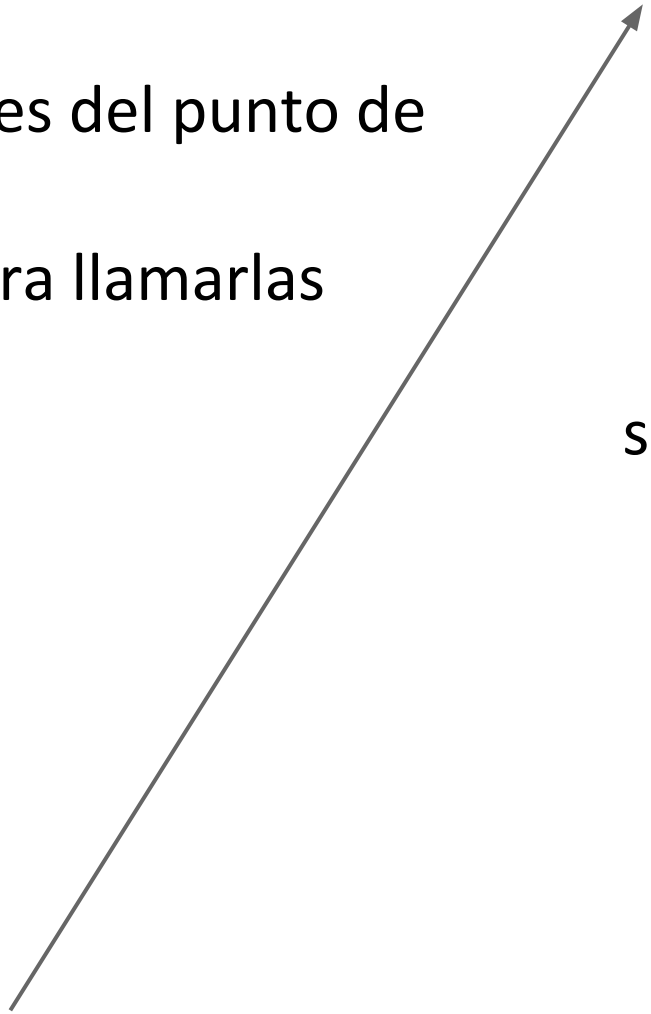
CALL <Procedimiento>

@truerand0m

```
section .text
global _start
; Se declaran antes del punto de
entrada
; se utiliza call para llamarlas
suma:
    mov eax,ecx
    add eax,edx
    ret

_start:
    mov ecx,2
    mov edx,4
    call suma
    mov eax,0
    mov ebx,0

    mov ecx,3
    mov edx,3
    call suma
salida:
    mov eax,1
    int 0x80
```



@truerand0m

Procedimientos

- Nota: un procedimiento con frecuencia puede utilizar un prólogo y un epílogo para gestionar el uso de la pila. No obstante, no es forzosa la presencia de ambos.
- Lenguajes de programación como C, los incluyen como parte de cada función.

@truerand0m

Interrupciones y excepciones

- El procesador proporciona dos mecanismos para modificar la ejecución de un programa: interrupciones y excepciones.
  - Una interrupción es evento asíncrono que generalmente se presenta por la acción de un dispositivo de entrada/salida.
  - Una excepción es un evento síncrono que se genera cuando el procesador detecta una o más condiciones predefinidas durante la ejecución de una instrucción.

@truerand0m

Interrupciones y excepciones

- Por ejemplo:
  - El mnemónico de interrupción **INT 80h** se encarga de atender las llamadas al sistema (syscall) en Linux.
  - INT 21h atiende las syscall en sistemas DOS.

@truerand0m

Interrupciones y excepciones

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md

Linux Syscall Reference

Show 25 entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	-	-	-	fs/namei.c:2520
10	sys_unlink	0x0a	const char __user *pathname	-	-	-	-	fs/namei.c:2352
11	sys_execve	0x0b	char __user *	char __user *__user *	char __user *__user *	struct pt_regs *	-	arch/alpha/kernel/entry.S:925
12	sys_chdir	0x0c	const char __user *filename	-	-	-	-	fs/open.c:361
13	sys_time	0x0d	time_t __user *tloc	-	-	-	-	kernel/posix-timers.c:855
14	sys_mknod	0x0e	const char __user *filename	int mode	unsigned dev	-	-	fs/namei.c:2067
15	sys_chmod	0x0f	const char __user *filename	mode_t mode	-	-	-	fs/open.c:507
16	sys_lchown16	0x10	const char __user *filename	old_uid_t user	old_gid_t group	-	-	kernel/uid16.c:27
17	not implemented	0x11	-	-	-	-	-	
18	sys_stat	0x12	char __user *filename	struct __old_kernel_stat __user *statbuf	-	-	-	fs/stat.c:150
19	sys_lseek	0x13	unsigned int fd	off_t offset	unsigned int origin	-	-	fs/read_write.c:167
20	sys_getpid	0x14	-	-	-	-	-	kernel/timer.c:1337
21	sys_mount	0x15	char __user *dev_name	char __user *dir_name	char __user *type	unsigned long flags	void __user *data	fs/namespace.c:2118
22	sys_oldumount	0x16	char __user *name	-	-	-	-	fs/namespace.c:1171
23	sys_setuid16	0x17	old_uid_t uid	-	-	-	-	kernel/uid16.c:67
24	sys_getuid16	0x18	-	-	-	-	-	kernel/uid16.c:212

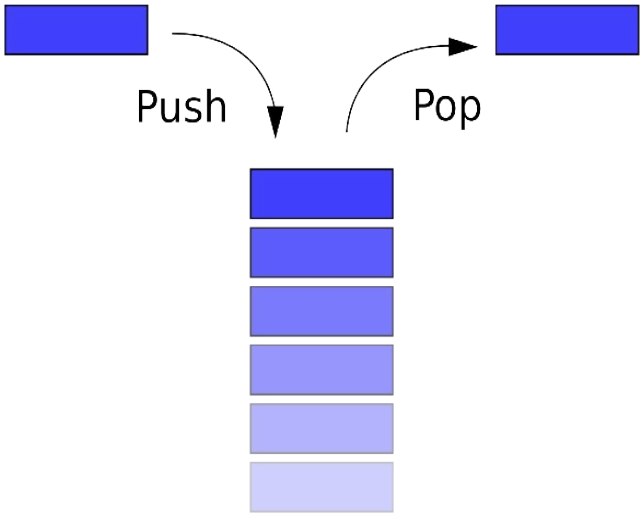
Showing 1 to 25 of 338 entries First Previous 1 2 3 4 5 Next Last

Generated from Linux kernel 2.6.35.4 using Exuberant Ctags, Python, and DataTables.
Project on GitHub. Hosted on GitHub Pages.

rand0m

Pila

- La pila (Stack) es una estructura de datos tipo LIFO (último en entrar, primero en salir) usada para colocar y remover elementos.



@truerand0m

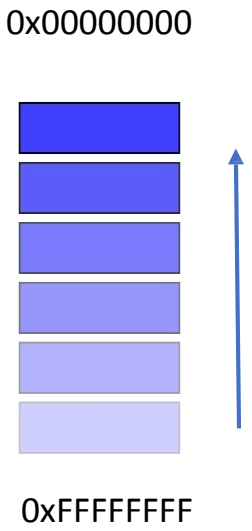
Pila

- Frecuentemente se utiliza para almacenamiento temporal de variables locales, argumentos y direcciones de retorno.
- Su principal uso es la gestión de datos intercambiados entre llamadas a funciones.
- Cada vez que se realiza una llamada se genera un nuevo *stack frame* en la pila.

@truerand0m

Pila

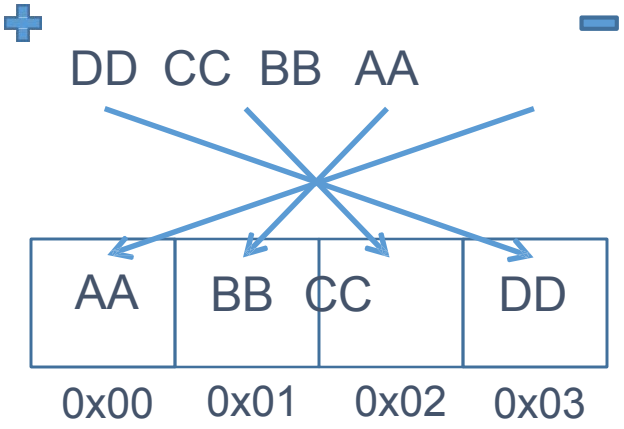
- La pila crece hacia las direcciones de memoria más bajas, es decir, cada vez que se insertan valores en la pila se utilizan direcciones de memoria más pequeñas.



@truerand0m

Lenguaje Ensamblador

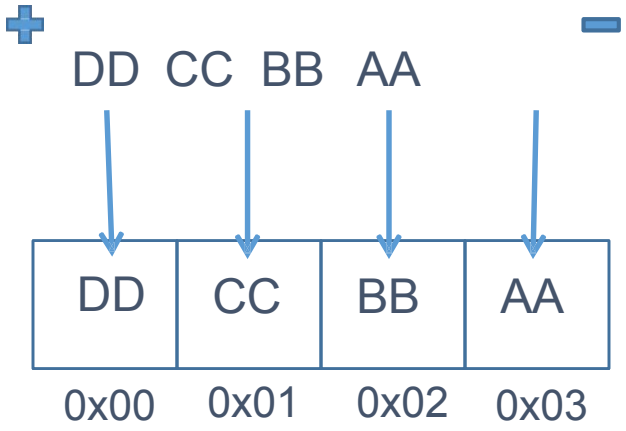
- ***Orden de bytes***
- Little-endian
- Significa que el byte menos significativo se almacena en la dirección más baja de memoria. Usado por Intel.



@truerand0m

Lenguaje Ensamblador

- Big-endian
- Representa los bytes en “orden natural”. El byte más significativo se almacena en la dirección de memoria más baja.



@truerand0m

Sintaxis AT&T

- La notación es: mnemónico origen, destino
- Los registros se denotan por el signo %
- Las constantes numéricas son precedidas del signo $
- Los números hexadecimales inician con 0x
- Los números binarios inician con 0b

@truerand0m

Sintaxis AT&T

- Cuando se efectúa operaciones con direcciones de memoria, el tamaño para 8, 16, 32 y 64 bits se especifica con los sufijos: b, w, l y q, a diferencia de Intel donde se usa: BYTE PTR, WORD PTR, DWORD PTR y QWORD PTR.

Ejemplo:

```
movl   $0x0,0x8(%rsp)
mov    DWORD PTR [rsp+0x8],0x0
```

@truerand0m

Directivas

- Son instrucciones propias del ensamblador y no están relacionadas con los mnemónicos definidos para un procesador.

- Generalmente son usadas para almacenar valores en el ejecutable o para señalar secciones .

- Ejemplos de directivas son DB, DW, EQU, SECTION, GLOBAL, etc.

@truerand0m

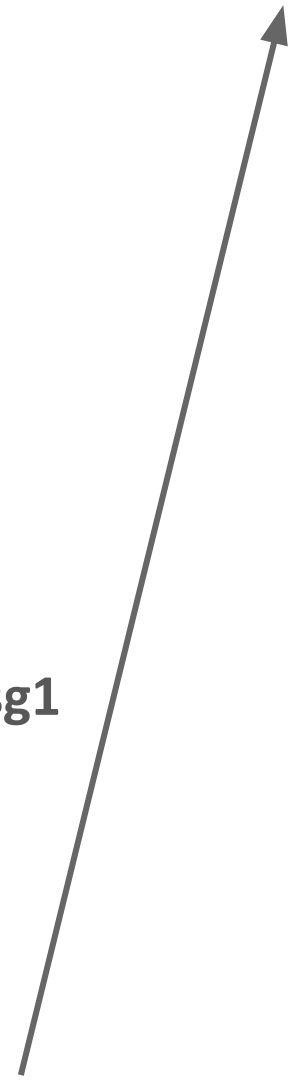
Generar un programa que compare si el contenido del registro EAX es igual a 0xFACEB00C,
si lo es que muestre el mensaje “EAX tiene 0xFACEB00C :)”,
en caso contrario “EAX no tiene 0xFACEB00C :V”.

@truerand0m

Codigo

```
global _start
section .text
_start:
      cmp
eax,0xFACEB00C
      jne bloque2

bloque1:
      mov edx,len_msg1
      mov ecx,msg1
      mov ebx,1
      mov eax,4
      int 0x80
      jmp fin
```



```
bloque2:
      mov edx,len_msg2
      mov ecx,msg2
      mov ebx,1
      mov eax,4
      int 0x80
fin:
      mov eax,1
      int 0x80

section .data
      msg1 db 'EAX tiene 0xFACEB00C :)',0xa
      len_msg1 equ $ - msg1
      msg2 db 'EAX no tiene 0xFACEB00C :V',0xa
      len_msg2 equ $ - msg2
```

@truerand0m

Tarea

- Significado de la “R” en los registros de propósito general de 64 bits(*)
- Generar el código ensamblador de las sentencias switch, y for(comentadas linea por linea
- Investigar tipos de datos en ensamblador y su tamaño

@truerand0m