Segunda Clase

Taller IA-32

Clase pasada…

```
section .data
msg db 'hola mundo '
num db '0',10
len EQU $-msg

section .text
global _start
_start:
XOR EDI,EDI
aca:
   MOV EDX,len
   MOV ECX,msg
   MOV EBX,1
   MOV EAX,4
   INT 0x80
   INC EDI
   CMP EDI,9
JNE aca
MOV EAX,1
INT 0x80
```

Directivas de preprocesador

- `%include "<nombredeasm>"`
- `%define <nombre> <valor>`
-
```
%if<condición>
    ...
    ...
%elif<condición2>
    ...
    ...
%else
    ...
    ...
%endif
```

Directivas de preprocesador - macros

```
%macro  intro 1
        push    ebp
        mov     ebp,esp
        sub     esp,%1
%endmacro


%macro  retz 0
        jnz     %%salto
        ret
    %%salto:
%endmacro
```

macros – basic_io.asm

```
%define BIO_EOL 10

%macro bio_iniciarmain 0
  section .text
  global _start
_start:
%endmacro

%macro bio_terminar 0
  mov eax,1
  int 80h
%endmacro

%macro bio_print 2 ; mensaje, largo
  mov edx,%2
  mov ecx,%1
  mov ebx,1 ;stdout
  mov eax,4
  int 80h
%endmacro

%macro bio_printerror 2 ; mensaje, largo
  mov edx,%2
  mov ecx,%1
  mov ebx,2 ;stderr
  mov eax,4
  int 80h
%endmacro

%macro bio_abrirl 1
  mov eax,5
  mov ebx,%1
  mov ecx,0
  mov edx,0
  int 80h
%endmacro

%macro bio_cerrar 1
  mov ebx,%1
  mov eax,6
  int 80h
%endmacro

%macro bio_leer 3
  mov edx,%3
  mov ecx,%2
  mov ebx,%1
  mov eax,3
  int 80h
%endmacro

%macro bio_abrire 1
  mov eax,8
  mov ebx,%1
  mov ecx,0
  mov edx,0
  int 80h
%endmacro

%macro bio_escribir 3
  mov edx,%3
  mov ecx,%2
  mov ebx,%1
  mov eax,4
  int 80h
%endmacro
```

Declarando Datos Inicializados

```
db    0x55               ; byte 0x55
db    0x55,0x56,0x57     ; tres bytes
db    'a',0x55           ; character y constante
db    'hello',13,10,'$'  ; string y constantes
dw    0x1234             ; 0x34 0x12
dw    'a'                ; 0x61 0x00
dw    'ab'               ; 0x61 0x62
dw    'abc'              ; 0x61 0x62 0x63 0x00
dd    0x12345678         ; 0x78 0x56 0x34 0x12
dd    1.234567e20  ; const en floating-point
dq    1.234567e20  ; const en double-precisión
dt    1.234567e20  ; const en extended-precisión
```

Registros

32-bit	16-bit	8-bit	8-bit
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL
EBP	BP		
ESI	SI		
EDI	DI		
ESP	SP		

Saltos – Jcc

JCXZ	if CX register is 0.	
JECXZ	if ECX register is 0.	
JC	if carry	(CF=1)
JNC	if not carry	(CF=0)
JZ	if zero	(ZF=1)
JNZ	if not zero	(ZF=0)
JO	if overflow	(OF=1)
JNO	if not overflow	(OF=0)
JP	if parity	(PF=1)
JNP	if not parity	(PF=0)
JS	if sign	(SF=1)
JNS	if not sign	(SF=0)
JPE	if parity even	(PF=1)
JPO	if parity odd	(PF=0)

Saltos – Jcc – Aritmeticos

JE	if equal	(ZF=1)	A==B
JNE	if not equal	(ZF=0)	A!=B
Jcc para números sin signo			
JA	if above	(CF=0 and ZF=0)	A>B
JNA	if not above	(CF=1 or ZF=1)	A<=B
JAE	if above or equal	(CF=0)	A>=B
JNAE	if not above or equal	(CF=1)	A<B
JB	if below	(CF=1)	A<B
JNB	if not below	(CF=0)	A>=B
JBE	if below or equal	(CF=1 or ZF=1)	A<=B
JNBE	if not below or equal	(CF=0 and ZF=0)	A>B

Saltos – Jcc – Aritmeticos

JE	if equal	(ZF=1)	A==B
JNE	if not equal	(ZF=0)	A!=B
Jcc para números con signo			
JG	if greater	(ZF=0 and SF=OF)	A>B
JNG	if not greater	(ZF=1 or SF<>OF)	A<=B
JGE	if greater or equal	(SF=OF)	A>=B
JNGE	if not greater or equal	(SF<>OF)	A<B
JL	if less	(SF<>OF)	A<B
JNL	if not less	(SF=OF)	A>=B
JLE	if less or equal	(ZF=1 or SF<>OF)	A<=B
JNLE	if not less or equal	(ZF=0 and SF=OF)	A>B

Direccionamiento

modo a registro
```
mov  eax, ecx
```
modo inmediato
```
mov  eax, 26
```
modo directo por registro
```
mov  eax, [esp]
```
modo directo
```
mov  eax, var_name
```
modo base y desplazamiento
```
mov  eax, [esp + 4]
```
modo base, índice y desplazamiento
```
mov  eax, [edi + ebx * 8 + 67]
```

Direccionamiento – modo base, índice y desplazamiento

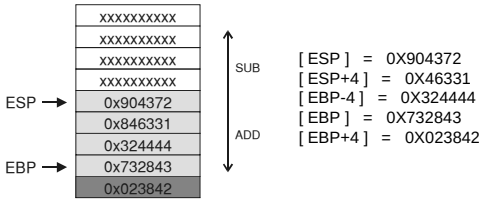
$$\underset{\text{base}}{\begin{pmatrix}\text{EAX}\\ \text{EBX}\\ \text{ECX}\\ \text{EDX}\\ \text{ESP}\\ \text{EBP}\\ \text{ESI}\\ \text{EDI}\\ \text{-}\end{pmatrix}} + \left[\underset{\text{índice}}{\begin{pmatrix}\text{EAX}\\ \text{EBX}\\ \text{ECX}\\ \text{EDX}\\ \text{EBP}\\ \text{ESI}\\ \text{EDI}\\ \text{-}\end{pmatrix}} * \underset{\text{escala}}{\begin{pmatrix}1\\ 2\\ 4\\ 8\end{pmatrix}}\right] + \underset{\text{desplazamiento}}{\begin{pmatrix}8\text{ bits}\\ 16\text{ bits}\\ 32\text{ bits}\\ \text{-}\end{pmatrix}}$$

$$\text{offset} = \text{base} + (\text{índice} \cdot \text{escala}) + \text{desplazamiento}$$

Direccionamiento – lo que no se puede hacer

- **NO! MEMORIA A MEMORIA**
  MOV [EAX], [ECX]
- **NO! DOBLE INDIRECCION**
  MOV EAX,[[ECX]]
- **NO! CUALQUIER ESCALA**
  MOV EAX, [ECX+EDX*5]
- **NO! SUMA DE TRES REGISTROS**
  MOV EAX, [ESI+EDI+ECX]
- **NO! DOBLE INDIRECCION CAMUFLADA**
  MOV EAX, [VAR] donde VAR es [dato]
- **NO! MULTIPLICAR DENTRO DE UNA DIRECCION**
  MOV EAX, [EBX * ECX]
- **NO! RESTAR DENTRO DE UNA DIRECCION**
  MOV EAX, [EBX - ECX]

Pila - General

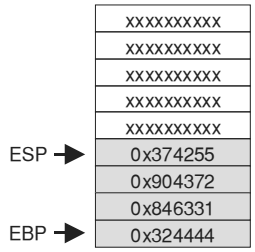


Pila – Funciones en Asm

(0)	`funcion:`	Nombre de la función, indica donde el punto de entrada
(1)	`push ebp`	Salva la base de la pila anterior
(2)	`mov ebp, esp`	Reposiciona la nueva base en el tope de la pila
(3)	`sub esp, 8`	Reserva 8 bytes en la pila, para uso local
(4)	`...`	Rutina…dirección
(5)	`...`	
(6)	`...`	
(7)	`add esp, 8`	Regresa el espacio pedido
(8)	`pop ebp`	Recupera la base anterior
(9)	`ret`	Retorno, recupera la siguiente instrucción terminando la rutina

Pila – Ejemplo

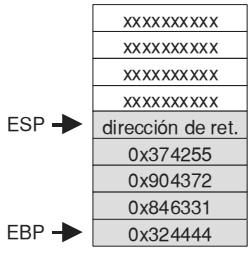
(0)	`funcion:`
(1)	`push ebp`
(2)	`mov ebp, esp`
(3)	`sub esp, 8`
(4)	`...`
(5)	`...`
(6)	`...`
(7)	`add esp, 8`
(8)	`pop ebp`
(9)	`ret`



antes del llamado

Pila – Ejemplo

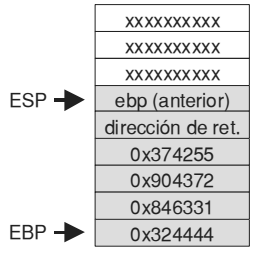
(0)	`funcion:`
(1)	`push ebp`
(2)	`mov ebp, esp`
(3)	`sub esp, 8`
(4)	`...`
(5)	`...`
(6)	`...`
(7)	`add esp, 8`
(8)	`pop ebp`
(9)	`ret`



paso (0)

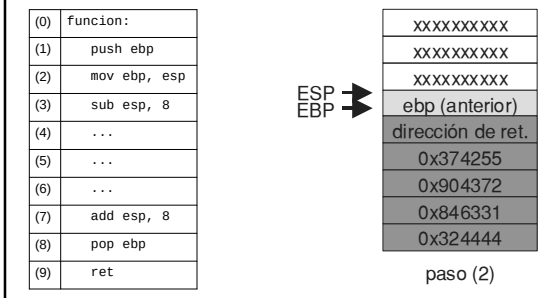
Pila – Ejemplo

(0)	`funcion:`
(1)	`push ebp`
(2)	`mov ebp, esp`
(3)	`sub esp, 8`
(4)	`...`
(5)	`...`
(6)	`...`
(7)	`add esp, 8`
(8)	`pop ebp`
(9)	`ret`



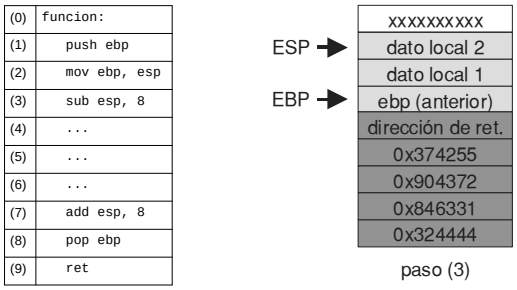
paso (1)

Pila – Ejemplo



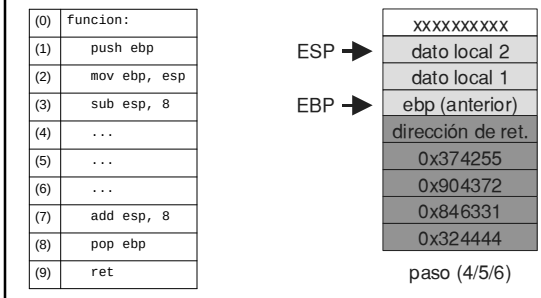
paso (2)

Pila – Ejemplo



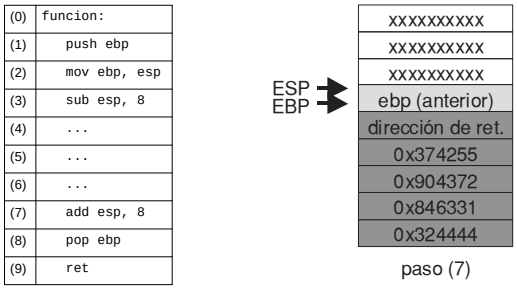
paso (3)

Pila – Ejemplo



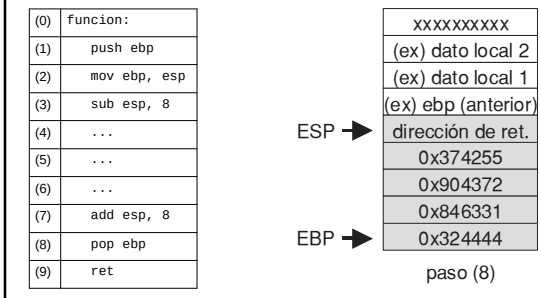
paso (4/5/6)

Pila – Ejemplo



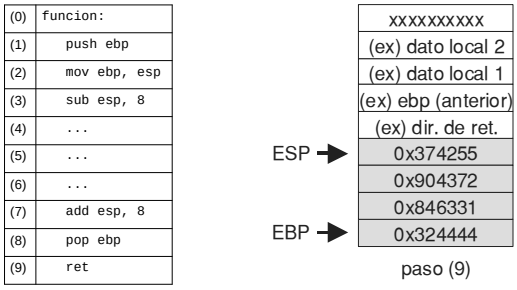
paso (7)

Pila – Ejemplo



paso (8)

Pila – Ejemplo



paso (9)

Interacción C-Assembler

programa.c

```
#include <sodio.h>
extern int funcionLoca( int a, int b, int c);
main() {
...
...
a = funcionLoca(3,4,5);
...
...
}
```

funcion.asm

```
global funcionLoca
section .text
funcionLoca:
...
...
ret
```

nasm -f elf funcion.asm -o funcion.o
gcc -o programa programa.c funcion.o

Interacción C-Assembler

programa.asm

```
extern printf
global main ; usamos "main" porque vamos a enlazar con C.
section .text
main:
...
call printf
...
...
```

nasm -f elf programa.asm
gcc programa.o -o programa

preguntas…?