Laboratorio Sesión 04: Ensamblador: subrutinas y estructuras

Objetivo

El objetivo de esta sesión es introducir la programación en ensamblador de subrutinas y estructuras en la arquitectura x86 bajo Linux. En concreto, se trabajarán aspectos como la programación de llamadas a subrutinas, la codificación de subrutinas completas y el accesos a estructuras de datos. Además se repasan conceptos como la programación de estructuras de control y el acceso a vectores y/o matrices.

Conocimientos Previos

Para realizar esta práctica deberíais repasar las traducciones directas de C a ensamblador del x86 de la codificación de subrutinas. Además deberías respasar la forma en que se almacenan y acceden las estructuras.

Los "Makefile"

Entre los ficheros de esta práctica encontraréis un elemento nuevo: un fichero llamado `Makefile`. Este fichero auxiliar sirve para compilar códigos cuando hay muchos ficheros de entrada distintos sin tener que teclear todos los nombres cada vez. Para utilizarlo basta con que tecleéis `make` y él mismo generará el resultado. Si abrís el fichero con un editor de textos veréis que su sintáxis es muy sencilla. En esta práctica podéis ejecutar el `make` con tres parámetros diferentes (uno por apartado): `test0`, `test1` y `test2`.

Estudio Previo

1. Explicad qué significan y para qué sirven los parámetros "argc"z "argv"del `main`.
2. Explicad para qué sirve la función `atoi`
3. Dibujad el struct S1:

```
typedef struct {
  char c;
  int k;
  int *m;
}S1;
```

4. Dibujad el bloque de activación de las rutinas `Insertar` y `Asignar`.
5. Dibujad el bloque de activación de la rutina `main`.
6. Dibujad el bloque de activación y traducid el siguiente código a ensamblador del x86:

```
int SencillaSub(S1 a, char b) {
  int i;

  if (a.c==b)
    i=0;
  else
    i=*(a.m);
  return i+a.k;
}
```

7. Dibujad el bloque de activación y traducid a ensamblador del x86 la siguiente subrutina:

```
int LlamaSencilla(S1 x, char y) {
  *(x.m)=SencillaSub(x,x.c);
  return x.k;
}
```

Trabajo a realizar durante la Práctica

1. Compilad (`make test0`) y probad la aplicación original escrita en C. Anotad algunos resultados originales para poder comprobar que vuestros códigos funcionan bien.

2. Traducid a ensamblador del x86 la rutina `Asignar`. Utilizad como base el fichero `Asigna.s` y comprobad el resultado con ayuda de los ficheros `Main.c` y `Insertar.c` (podéis usar `make test1` para compilar). Entregad en el Racó de la asignatura el fichero `Asigna.s`.

3. Traducid a ensamblador del x86 la rutina `Insertar`. Utilizad como base el fichero `Inserta.s`. Probadla con el fichero `Main.c` y con la rutina en ensamblador del apartado anterior (`make test2`). Entregad en el Racó de la asignatura el fichero `Inserta.s`

4. Recordad entregar en el Racó de la asignatura los ficheros `Asigna.s` y `Inserta.s`. Debéis entregar sólo los dos ficheros fuentes, sin comprimir ni cambiarles el nombre, y sólo una versión por pareja de laboratorio (es indistinto que miembro de la pareja entregue).

Nombre: Tanvir Hossain Grupo: 23

Nombre: Edgar Pérez

Hoja de respuesta al Estudio Previo

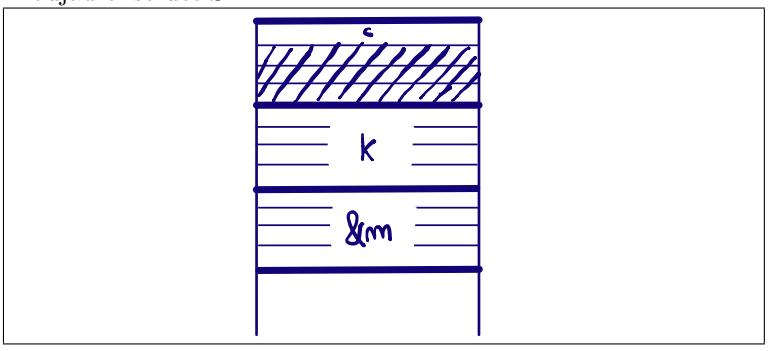
1. Explicad qué significan y para qué sirven los parámetros "argc" y "argv" del `main`.

"argc" indica el numero de parametros entrado +1 (nombre del programa)
"argv" es el array de estos elementos a partir del 1.
argv[0] contiene el nombre del ejecutable.

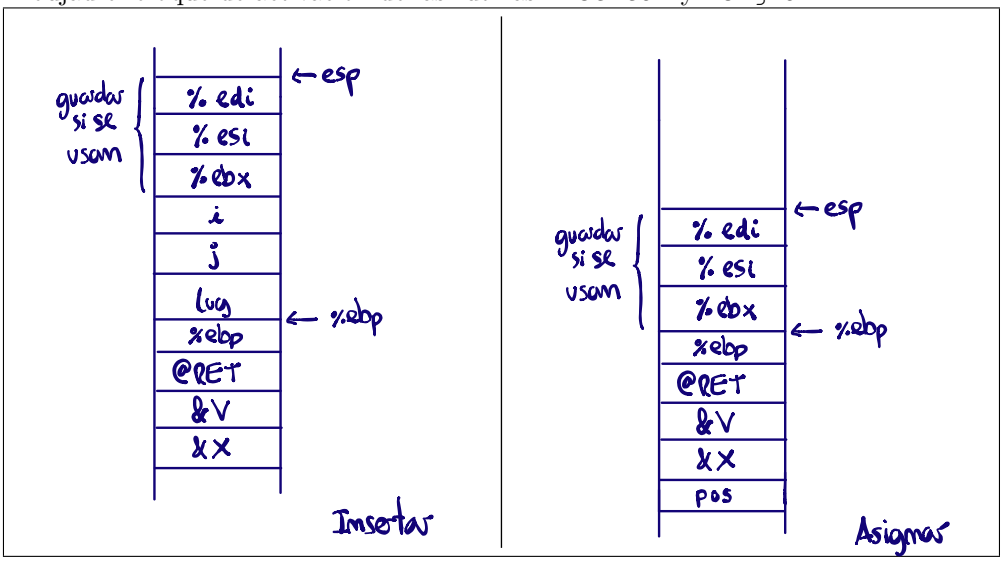
2. Explicad para qué sirve la función `atoi`

atoi: ascii to int
convierte el caracter "3" al entero 3. (le resta '0')

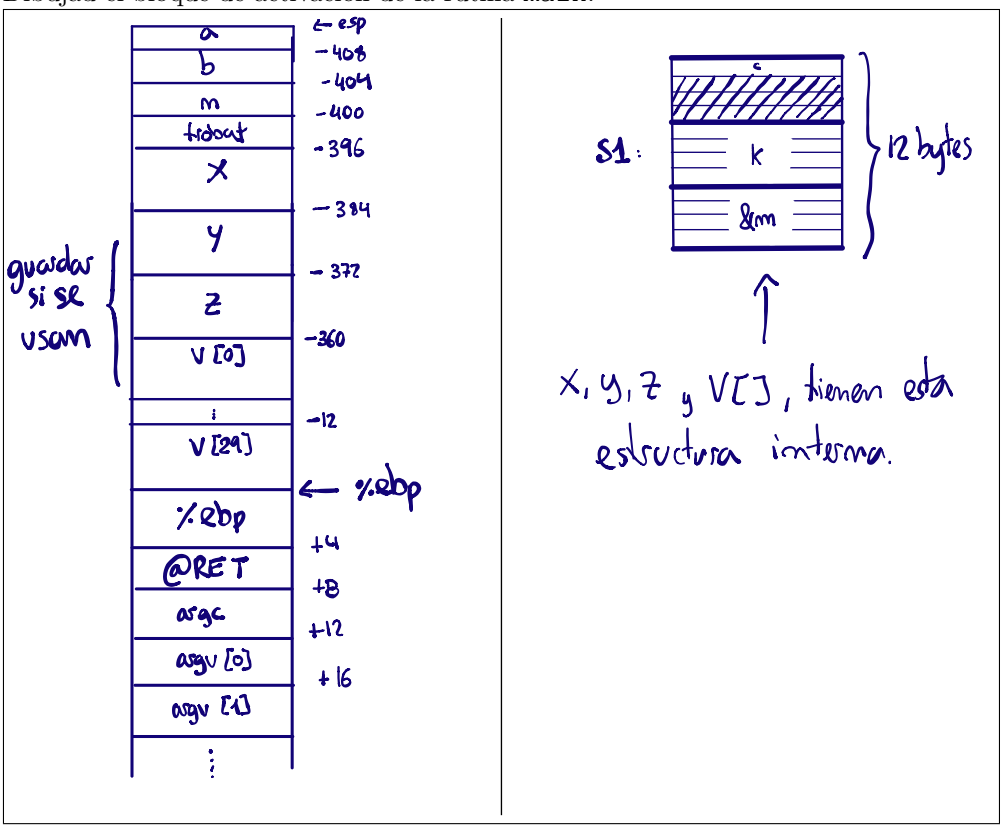
3. Dibujad el struct S1:



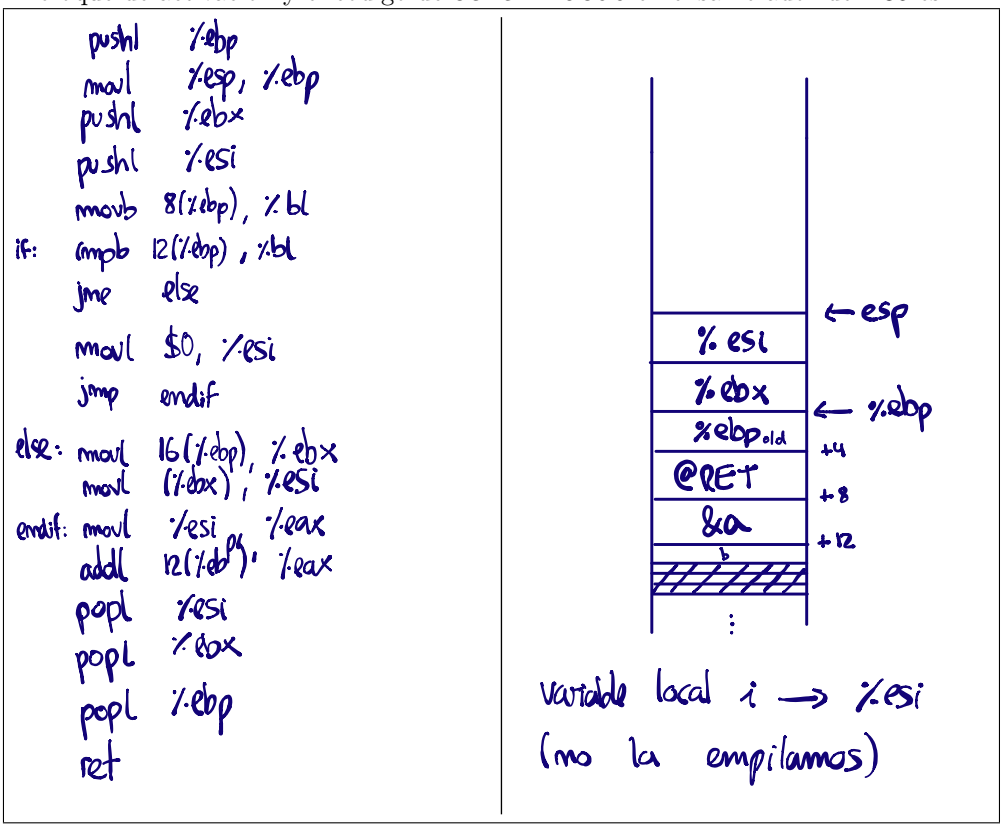
4. Dibujad el bloque de activación de las rutinas `Insertar` y `Asignar`.



5. Dibujad el bloque de activación de la rutina `main`.

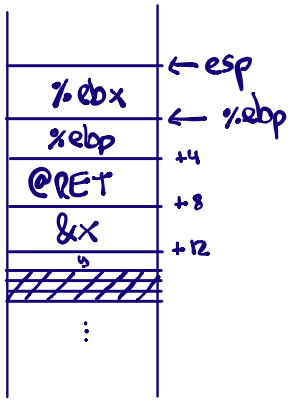


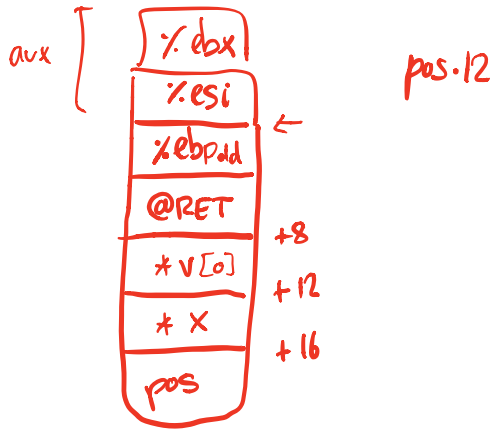
6. El bloque de activación y el código de `SencillaSub` en ensamblador del x86 es:



7. El bloque de activación y el código de `LlamaSencilla` en ensamblador del x86 es:

```
pushl  %ebp
movl   %esp, %ebp
pushl  %ebx
movl   8(%ebp), %ebx  #@x
movzbl (%ebx), %eax
pushl  %eax
pushl  %ebx
call   SencillaSub
movl   8(%ebx), %ecx  #&m
movl   %eax, (%ecx)
movl   4(%ebx), %eax
popl   %ebx
popl   %ebp
ret
```




aux
%.ebx
%.esi
%ebp.dd
@RET
*v[o]
* x
pos
+8
+12
+16
pos.12