

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 sintaxis 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`” “`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 `Buscar` y `BuscarElemento`.
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 vuestrlos códigos funcionan bien.
2. Traducid a ensamblador del x86 la rutina `BuscarElemento`. Utilizad como base el fichero `BuscaElemento.s` y comprobad el resultado con ayuda de los ficheros `Main.c` y `Buscar.c` (podéis usar `make test1` para compilar). Entregad en el Racó de la asignatura el fichero `BuscaElemento.s`.
3. Traducid a ensamblador del x86 la rutina `Buscar`. Utilizad como base el fichero `Busca.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 `Busca.s`.
4. Recordad entregar en el Racó de la asignatura los ficheros `BuscaElemento.s` y `Busca.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: _____

Grupo: _____

Nombre: _____

Hoja de respuesta al Estudio Previo

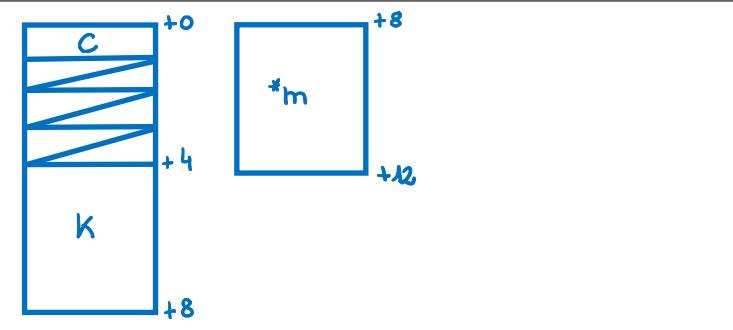
1. Explicad qué significan y para qué sirven los parámetros “argc” y “argv” del main.

argc es una variable entera que indica el número de argumentos introducidos.
argv es un vector que contiene los argumentos introducidos. Ambas parámetros se usan en la función main().

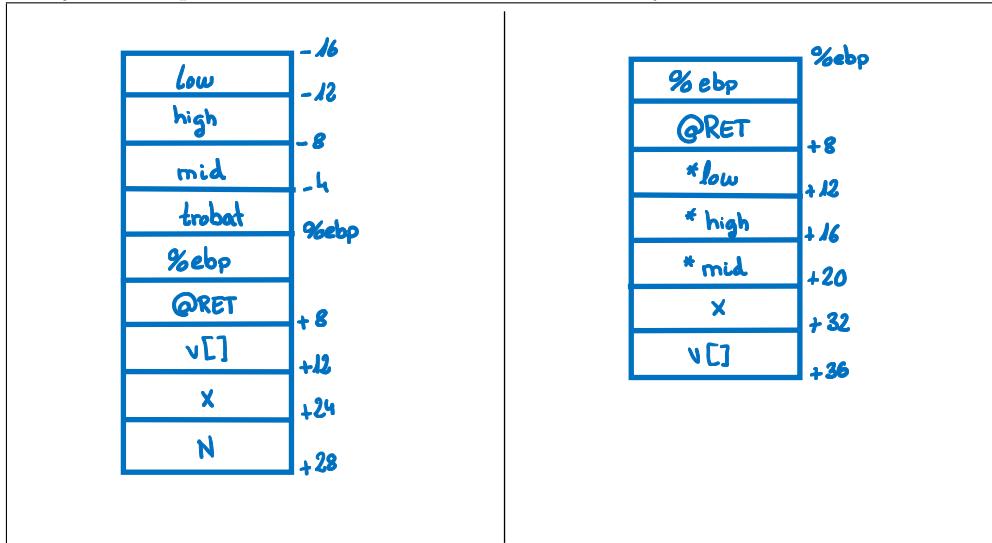
2. Explicad para qué sirve la función atoi

Es una función que convierte un string que pasamos como parámetro en un integer que devuelve.

3. Dibujad el struct S1:



4. Dibujad el bloque de activación de las rutinas Buscar y BuscarElemento.



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

<table border="1"> <tr><td>a</td><td>-412</td></tr> <tr><td>b</td><td>-408</td></tr> <tr><td>n</td><td>-404</td></tr> <tr><td>trabat</td><td>-400</td></tr> <tr><td>X</td><td>-396</td></tr> <tr><td>Y</td><td>-384</td></tr> <tr><td>Z</td><td>-372</td></tr> <tr><td>v[0]</td><td>-360</td></tr> <tr><td>...</td><td>-348</td></tr> <tr><td>v[39]</td><td>-12</td></tr> <tr><td>%ebp</td><td>%ebp</td></tr> <tr><td>@RET</td><td>+8</td></tr> <tr><td>argc</td><td>+12</td></tr> <tr><td>argv</td><td>+12</td></tr> </table>	a	-412	b	-408	n	-404	trabat	-400	X	-396	Y	-384	Z	-372	v[0]	-360	...	-348	v[39]	-12	%ebp	%ebp	@RET	+8	argc	+12	argv	+12	
a	-412																												
b	-408																												
n	-404																												
trabat	-400																												
X	-396																												
Y	-384																												
Z	-372																												
v[0]	-360																												
...	-348																												
v[39]	-12																												
%ebp	%ebp																												
@RET	+8																												
argc	+12																												
argv	+12																												

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

<table border="1"> <tr><td>i</td><td>-4</td></tr> <tr><td>%ebp</td><td>%ebp</td></tr> <tr><td>@RET</td><td>+8</td></tr> <tr><td>a</td><td>+20</td></tr> <tr><td>b</td><td>+21</td></tr> </table>	i	-4	%ebp	%ebp	@RET	+8	a	+20	b	+21	<pre> pushl %ebp movl %esp, %ebp subl \$4, %esp movb 8(%ebp), %al # a.c → %al i: cmpb %al, 20(%ebp) jne else # Si a.c ≠ b salta movl \$0, %eax # i=0 jmp end else: movl 16(%ebp), %eax # a.m → %eax movl (%eax), %eax # *a.m → %eax end: addl 12(%ebp), %eax # i + a.k movl %ebp, %esp popl %ebp ret </pre>
i	-4										
%ebp	%ebp										
@RET	+8										
a	+20										
b	+21										

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

<p>%ebp</p> <table border="1"><tr><td>%ebp</td></tr><tr><td>@RET</td></tr><tr><td>x</td></tr><tr><td>y</td></tr></table> <p>+8 +20 +21</p>	%ebp	@RET	x	y	<pre>pushl %ebp movl %esp, %ebp movzbl 8(%ebp), %eax # x.c pushl %eax # x.v pushl 12(%ebp) # x.m pushl 16(%ebp) # x.m movzbl 8(%ebp), %eax # x.c pushl %eax # x.v call lamaSencilla addl \$13, %esp movl 16(%ebp), %eax # x.m->%eax movl %eax, (%ecx) # *(x.m)->%ecx movl 12(%ebp), %eax # x.v->%eax movl %ebp, %esp popl %ebp ret</pre>
%ebp					
@RET					
x					
y					