Laboratorio Sesión 03: El Primer Programa en Ensamblador

Objetivo

El objetivo de esta sesión de laboratorio es presentar la estructura básica de un programa en ensamblador y enumerar algunas de las directivas de ensamblador que vamos a utilizar durante el curso.

Documentación

La documentación básica para esta sesión y todas las relacionadas con la programación en ensamblador en IA 32 es la siguiente:

- Los manuales de Intel de IA32.
- El manual del Ensamblador de GNU: "*Using as. The GNU Assembler*".

Esta documentación puede encontrarse en la página web de la asignatura.

Estructura Básica de un programa en Ensamblador

Vamos a presentar la estructura básica de un programa en ensamblador con un ejemplo. El ejemplo es la traducción a ensamblador del siguiente programa escrito en C:

```
#include <stdio.h>

char v[100] = "Esto es una frase acabada en un salto de linea.\n";
int cont;

main()
{  int i;

  cont = 0;
  i = 0;

  do {
    printf("%c", v[i]);
    if (v[i] == 'a') cont++;
    i++;
  } while (v[i] != '\n');

  printf("\nLa frase tiene %d a's.\n", cont);
}
```

Este mismo programa en ensamblador podría ser el siguiente:

```
.data
        .align 4
v:      .string "Esto es una frase acabada en un salto de linea.\n"
cont:   .int 0

.section .rodata
LC0:    .string "%c"
LC1:    .string "\nLa frase tiene %d a's.\n"

.text
        .align 4
        .globl main
        .type main,@function
main:
        pushl %ebp                  # Montar el enlace dinamico
        movl %esp,%ebp

        pushl %ebx                  # Salvar registros
        pushl %esi
```

```
        movl $0, cont              # cont <-0
        xorl %esi, %esi            # i <- 0
        movl $v, %ebx              # ebx <- @inicio v
do:
        movb (%ebx, %esi), %al     # al <- v[i]
        cmpb $'a', %al             # al == 'a'?
        jne  NoA
        incl cont                  # cont++
NoA:
        pushl %eax                 # printf ("%c", v[i])
        pushl $LC0
        call printf
        addl $8, %esp

        incl  %esi                 # i++
        cmpb $'\n', (%ebx, %esi)   # v[i] != '\n' ?
        jne do

       pushl cont                 # printf("\nLa frase tiene %d a's.\n",cont)
        pushl $LC1
        call printf
        addl $8,%esp

        popl %esi                  # Restaurar registros
        popl %ebx
        movl %ebp,%esp             # Deshacer enlace dinamico
        popl %ebp
        ret                        # Retorno
```

En este programa podemos destacar varios puntos:

- En el ejemplo se pueden identifcar claramente tres secciones:
  - La sección de datos identificada por la directiva "`.data`".
  - La sección de código identificada por la directiva "`.text`".
  - Una sección con constantes identificada por las directivas "`.section .rodata`". La directiva "`.rodata`", implica que esta sección sólo es accesible en lectura.
- En la sección de datos pueden declararse variables globales. En el ejemplo tenemos:
  - El vector de caracteres `v`. Se define con la directiva "`.string`" y su valor inicial.
  - El entero cont, definido con la directiva "`.int`" y su valor inicial.
- Tanto la sección de datos como la de código contienen al principio la directiva "`.align 4`". Esta directiva hace que la dirección inicial de cada sección sea múltiplo de 2^4.
- El punto de entrada viene identificado por la etiqueta "`main:`" y las directivas "`.gobl main`" y "`.type main,@function`".
- El programa principal (`main`) ha de verse como una subrutina, con los mismos convenios que una subrutina cualquiera: definición de variables locales, uso de registros, etc.
- Si "`cont`" es una etiqueta de un dato:
  - La instrucción "`movl cont, %eax`" movería el contenido de la posición de memoria etiquetada por "`cont`" al registro `eax`.
  - La instrucción "`movl $cont, %eax`" equivale a la instrucción "`leal cont, %eax`". Es decir mueve la direccion asociada a la etiqueta "`cont`" al registro `eax`.
- En general, las constantes en ensamblador se especifican igual que en un programa C.

Invocación del Ensamblador

La forma más fácil de compilar un programa escrito en ensamblador es utilizar directamente el compilador de C:

```
$> gcc frase.s -o frase
```

Se podría utilizar el ensamblador "**as**" y luego el montador "**ld**". Utilizando el compilador de C nos ahorramos bastante trabajo. Si queréis ver en detalle lo que hace el compilador de C usad el siguiente comando:

```
$> gcc –v frase.s –o frase
```

Para ejecutar el programa sólo hay que hacer:

```
$> ./frase
```

Algo de Linux: Gestión de procesos

Cada vez que ejecutamos un programa, el S.O. crea un proceso. Para ver los procesos de un determinado usuario se utiliza el comando **ps**. Consultad el manual para interpretar la información que muestra el comando.

Asociado a cada proceso hay un número, el **pid**, que identifica el proceso. Existe un comando fundamental que sirve para matar un proceso siempre que sea necesario: "**kill –9 pid**".

Cuando ejecutamos un comando o programa normalmente (en foreground), el intérprete de comandos no nos vuelve a mostrar el prompt ("`$>`") hasta que el programa acaba. Existen 2 caractéres de control que modifican esta situación:

- `^c` (control c). Aborta inmediatamente el programa en ejecución.
- `^z` (control z). Detiene el programa en ejecución. Se puede reanudar el programa en foreground utilizando el comando **fg**. Se puede reanudar el programa en background (el programa se ejecuta en segundo plano y aparece el promt) utilizando el comando **bg**. Si queremos ejecutar un programa directamente en background hay que hacer lo siguiente:

```
$> ./programa &
$>
```

Alguno de los comandos relacionados con la gestión de procesos son los siguientes:

comando	descripción	ejemplo
ps	Muestra los procesos del sistema	`ps` `ps –u username`
top	Muestra los procesos en ejecución de forma interactiva.	`top`
kill pid	Envia la señal de finalización de proceso	`kill –9 8123`

Trabajo previo de la sesión

Como trabajo previo de la sesión hay que realizar las siguientes tareas:

1) Averiguad el código ASCII de los dígitos '0'-'9', de las letras minúsculas 'a'-'z' y de las letras mayúsculas 'A'-'Z'.
2) Escribid una rutina en C que, dado un número natural entre 0 y 9, lo convierta al carácter ASCII correspondiente (entre '0' y '9').
3) Escribid una rutina en C que, dado un carácter entre 'a' y 'z', lo convierta en mayúsculas.
4) Escribid una rutina en C que, dado un carácter entre 'A' y 'Z', lo convierta en minúsculas.
5) ¿Qué significa la directiva ".align"? (Consultad el manual de ensamblador).
6) Explicad en detalle qué hace, en el programa ejemplo, la siguiente secuencia de instrucciones:

```
pushl cont    # printf("\nLa frase tiene %d a's.\n",cont)
pushl $LC1
call printf
addl $8,%esp
```

7) Explicad qué hace, en el programa ejemplo, la siguiente instrucción:

```
movb (%ebx, %esi), %al
```

¿la podríamos haber sustituido por ésta?

```
movsbl (%ebx, %esi), %eax
```

8) Explicad por qué utilizamos la instrucción "`cmpb $'a', %al`" y no "`cmpl $'a', %eax`"

Resultados de la sesión

Durante la sesión tenéis que intentar hacer lo siguiente (en el orden indicado):

- Compilar y ejecutar el programa en ensamblador que se ha utilizado como ejemplo.
- Modificar el programa original para que cuente el número de 'a's, 'e's y 'o's (en minúsculas) que hay en la frase.
- Modificar el programa para que sustituya mayúsculas por minúsculas y viceversa y cuente el numero de sustituciones de cada tipo.
- Modificar el programa para que sustituya los caracteres de la 'a' a la 'j' por los digitos del '0' al '9'. Es decir, sustituir 'a' por '0', 'b' por '1', etc. y cuente el numero de sustituciones.