Documentación Proyecto 2 : Lenguaje ensamblador

Massetti Matías

Privitello Gaetano

Universidad Nacional del Sur

2016

Definiciones y especificación de requerimientos:

a) Definición general del proyecto de software:

El proyecto es parte de la asignatura “Organización de Computadoras”, dictada en la Universidad Nacional del Sur, en la ciudad de Bahía Blanca.

El mismo consiste en implementar en lenguaje ensamblador, un programa para volcar el contenido de un archivo en formato hexadecimal y ASCII.

La elaboración del proyecto significa una introducción en el lenguaje ensamblador, sus principales características y usos.
El programa está destinado a cualquier usuario del sistema operativo Linux y no son requeridos conocimientos extras a los mencionados en este documento para ejecutar el mismo.

b) Especificación de requerimientos del proyecto:

La salida del programa será similar a la producida por el comando *hexdump -c.*
El programa toma el contenido del archivo de entrada y muestra por pantalla lo siguiente:

[Dirección Base] **[Contenido hexadecimal]** **[Contenido ASCII]**

La salida es organizada en filas de 16 bytes. La primera columna muestra la dirección base de los siguientes 16 bytes, expresada en hexadecimal.

Luego siguen 16 columnas que muestran el valor de los siguientes 16 bytes del archivo a partir de la dirección base, expresados en hexadecimal.

La última columna (delimitada por caracteres '|') de cada fila muestra el valor de los mismos 16 bytes, pero expresados en formato ASCII, mostrando solo los caracteres imprimibles, e indicando la presencia de caracteres no imprimibles con '.'

c) Uso del sistema

La implementación del programa fue realizada sobre el ensamblador Yasm, sobre arquitectura x86, haciendo uso de las llamadas al sistema provistas por el sistema operativo Linux.
El programa (denominado "**volcar.asm**") funciona en la máquina virtual OCUNS dada por la cátedra.
Para ejecutarlo se debe insertar lo siguiente en la terminal:
$ ./volcar [-h] <archivo>
Dónde:
. <archivo>: La ruta a un archivo de cualquier formato (binario,imagen,texto u otro), de tamaño máximo 1MB.
. -h : Imprime un mensaje de ayuda y tiene una terminación normal (0). Es opcional y siempre aparece en primera posición en la lista de argumentos.

El programa termina su ejecución luego de imprimir el mensaje de ayuda, sin considerar otros argumentos que pudieran aparecer a continuación.
Cuando el programa finaliza la ejecución, el mismo se encarga de informar la situación de terminación (exit status) a quien lo haya invocado. Para ello, se hace uso de la llamada al sistema sys_exit, respetando lo siguiente:

EBX	**DETALLE**
0	Terminación normal
1	Terminación anormal
2	Terminación anormal por error en el archivo de salida

Descripción de procesos y servicios ofrecidos por el sistema

El programa comienza analizando la cantidad de parámetros que recibe del usuario:

-Si el usuario sólo envía un parámetro, el programa analizará si es la cadena "–h" , en caso de serlo mostrará un mensaje de ayuda en pantalla , caso contrario el parámetro será tratado como un archivo de entrada .

-Si el usuario ingresa dos parámetros en caso de encontrarse con –h se mostrara el mensaje de ayuda y finalizara, si no se encuentra el parámetro –h, se produce un error.

-Si el usuario no ingresa parámetros, el programa finaliza con error.

-Si el usuario ingresa la ruta del archivo de entrada, el programa comenzará a procesarlo, en primer lugar se abre y luego se comienza a leer.

Las tareas explicadas anteriormente son realizadas por la rutina _start.

Los restantes procesos son los siguientes:

ControlarParametros:
Controla la los parámetros ingresados por el usuario y determina si se produjo un error o si el programa puede continuar con su ejecución normal.

ArchivoEntrada:
Abre un archivo y lo asocia a un descriptor de archivo

Proceso:
Carga al buffer de entrada, 16 bytes desde el descriptor de archivo

Completar:
Realiza la lectura del archivo y procesa los bytes obtenidos en el buffer, imprime una línea de 16 bytes y controla que no se haya encontrado con un fin de archivo (EOF).

Ultima:
Rutina encargada de procesar los últimos 16 bytes del archivo. En caso de ser menos de 16 bytes rellena con blancos los faltantes.

EscribirBlancos:
Escribe caracteres blancos ‘’ “.

Fin:
Finalización exitosa del programa (Coloca en el registro ebx -- exit_success).

Exit:
Cierra los archivos abiertos y finaliza el programa.

Error:
Finalización anormal del programa.

ErrorArhivo:
Finalización anormal del programa por conflictos con el archivo.

Ayuda:
Imprime mensaje de ayuda en pantalla.

LineadeSalida:
Arma las plantillas de hexa, ASCII y dirección base. Para luego imprimirlas
CharSiguiente:
Procesa un carácter del buffer de entrada. Y lo coloca correctamente en las plantillas necesarias.

ResetContador1 y ResetContador2
Son métodos auxiliares utilizados para la formación de la dirección base.

Para la implementación del proyecto optamos por una forma de trabajar que implica tener **Tablas de información** a partir de las cuales se realizaran los mapeos correspondientes.
A continuación daremos ejemplos claros de los procedimientos utilizados.

Como funciona:

1) TablaEjemplo '0123456789ABCDEF'
   (*Tabla de la cual se mapearan los dígitos necesarios.)*

2) mov eax,1 (*Es la posición desde la cual se tomaran 2 bytes . El registro eax cumplirá la función de índice)*

Ejemplo:
Teniendo como referencia la TablaEjemplo, tenemos:

mov bx,[TablaEjemplo+1*eax]

(Dependiendo del valor de eax , será la cantidad de lugares a la derecha que se moverá el "índice" y donde caiga se tomaran los siguientes 2 dígitos . Al ser bx un registro de 8 BYTES solo entraran 2 dígitos Hexadecimales.)

Supongamos las siguientes instrucciones :

mov eax,3
mov bx,[TablaEjemplo+1*eax]

(Aquí lo que ocurrirá es :
De TablaEjemplo : "012.3456789ABCDEF" (Movemos 3 lugares)
Tomamos los siguientes 2 dígitos luego del desplazamiento
Es decir bx <-- 34)

Ahora una vez cargados los dígitos a utilizar realizamos el siguiente procedimiento que de cierta manera es el inverso al anteriormente explicado:

Disponemos de lo que llamamos Plantillas (Es la "estructura" por defecto de algo que queremos representar)
También podríamos darle el nombre de "Molde" , a partir de esta plantilla nosotros armaremos todas las representaciones

Supongamos que contamos con la siguiente plantilla :
plantillaEj : '00000000'

Tomemos como referencia el procedimiento anterior, hemos cargado en el reg bx <- '34' Ahora quisiéramos poder representarlo en nuestra plantilla , realizaremos una sustitución de caracteres (los pisaremos).

Entonces:

```
mov eax,3

mov bx,[TablaEjemplo+1*eax]

mov [plantillaEj+4],bx     ;
```

(El efecto de esta instrucción es análogo al anterior
Es decir nos movemos en nuestra plantilla '0000.XX00,donde
XX serán los valores que serán sustituidos por el contenido
de bx , entonces '00003400')