Universidad Nacional de General Sarmiento

Organización del computador

TP Final – ARM

Comisión 4 Grupo 2 – Segundo cuatrimestre 2023

Docentes:

- Lic. Mariano Vargas
- Prof. Ignacio Tula
- Adscripta Zoe Sacks

Alumnos:

- Federico Buen
- Abel Aquino
- Maximiliano Sanchez

Objetivo del software

Se nos pidió desarrollar un software escrito en lenguaje ensamblador en arquitectura ARM para resolver la siguiente consigna:

Realizar un programa que reciba como input un mensaje normal y devuelva el mismo mensaje cifrado aplicando el método César mejorado.

Al mensaje cifrado se agregará un bit de paridad al final de la cadena tomando el valor 0 o 1 dependiendo de la paridad de la cadena.

El programa también debe ser capaz de descifrar un mensaje codificado devolviendo el mensaje original.

Para realizar el cifrado del mensaje el programa procesa la cadena ingresada aplicando un desplazamiento carácter a carácter con un vector con 5 posiciones los cuales representan el tamaño del desplazamiento.

El programa debe interactuar con el usuario en modo consola permitiendo ingresar el mensaje por teclado y mostrar el resultado por pantalla.

La consigna tiene dos etapas bien definidas como requerimiento:

- Codificar / decodificar un mensaje con un vector de desplazamientos
- Decodificar el mensaje con una palabra clave

Diseño del software

La arquitectura general del software es un solo código fuente donde se encuentra bien definida la estructura del main en el que se realizan las llamadas a las subrutinas que se utilizan para simplificar las distintas partes del software, pudiendo resolver por parte la consigna, aplicando el concepto de modularización y reutilización.

Se desarrollaron los conceptos de direccionamiento, manejo de registros, manejo del stack, llamada a subrutinas y llamadas a interrupciones de I/O al SO.

También se debieron realizar subrutinas para el manejo de codificación ASCII, transformación de los caracteres en dígitos y viceversa para manejar las entradas y salidas de la información.

Se realiza la reserva de memoria mediante etiquetas de los espacios requeridos para procesar los distintos datos que requiere el programa como cadena de entrada, cadena de salida, vector de desplazamientos, y los datos utilizados de forma interna necesarios para resolver las subrutinas, como por ejemplo flags de control.

Criterios de diseño aportados por el grupo

Ingresos válidos por el usuario

Usando vector de desplazamientos para codificar y decodificar

La cadena del mensaje es una cadena de caracteres (letras de la ‘a’ a la ‘z’ sin caracteres especiales, se puede incluir espacios los cuales serán ignorados en la codificación, aunque serán contabilizados como carácter procesado para mostrar el largo de la cadena), 5 dígitos de rango [0-9] que representan el vector de desplazamientos de 5 posiciones y por último un carácter que representa la acción solicitada (‘c’ – codificar el mensaje y ‘d’ – decodificar el mensaje). El carácter de separación para estos datos de entrada es punto y coma ‘;’.

El largo máximo de la cadena del mensaje es de 48 caracteres incluyendo espacios.

El mínimo es de 5 caracteres de la ‘a’ a la ‘z’, sin contar espacio

Ejemplo: cadena del mensaje;[0-9];[0-9];[0-9];[0-9];[0-9];c/d

Usando la palabra clave para decodificar

Para decodificar un mensaje sin tener el vector de desplazamiento se puede utilizar la palabra clave otorgada por el programa en el momento de la codificación. El ingreso válido en este caso es el mensaje codificado, punto y coma (‘;’), palabra clave de 5 caracteres, punto y coma (‘;’) y el carácter ‘d’ para decodificar.

Ejemplo: cadena del mensaje a decodificar;clave;d

Tener en cuenta para la decodificación del mensaje tanto con la entrada del vector o la palabra clave debe incluir el bit de paridad correcto.

Cualquier entrada que no respete estos formatos de ingreso producirán un incorrecto resultado de la ejecución del software.

Codificación de la palabra clave

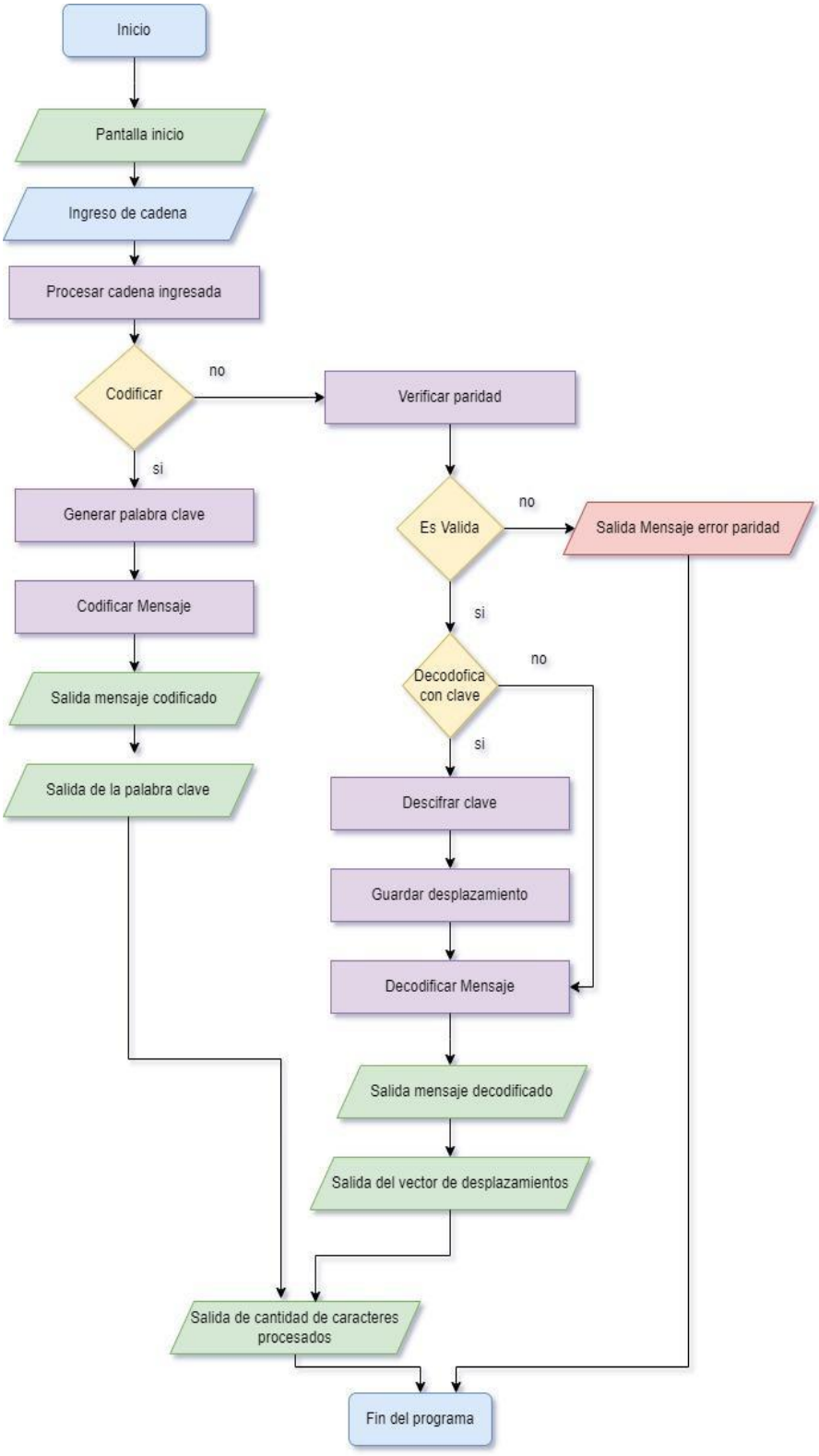
El criterio usado para resolver la creación de la palabra clave que se usa para decodificar el mensaje fue:

El programa tiene un vector de desplazamiento interno de 5 posiciones.

Se toman los primeros 5 caracteres distintos de espacio del mensaje a codificar (si o si la cadena debe tener 5 letras como mínimo). La clave se genera codificando cada una de las letras con cada posición del vector de desplazamiento interno. El resultado de esa codificación es la palabra clave que se muestra al usuario.

Esto da como resultado una palabra clave codificada distinta a la codificación de mensaje en sí, dando un doble factor de seguridad.

Diagrama de flujo general del software



Ejecución del programa

Todo el programa interactúa con el usuario por modo consola.

Al iniciar, el programa muestra la pantalla de presentación, los mensajes de opciones para ingresar y queda en espera del ingreso del usuario por teclado

```
occ4g2@Alysa:~ $ ./tpfinal
#=========================================================#
#                                                         #
#           MÁQUINA DE CODIFICACIÓN DE MENSAJES           #
#            Cifrado Cesar mejorado, método OC            #
#                                                         #
#=========================================================#

Ingrese el mensaje con alguno de los siguientes formatos de entrada:
(mensaje a codificar o decodificar;[0-9];[0-9];[0-9];[0-9];[0-9];c/d)
(mensaje a decodificar;clave;d)
---------------------------------------------------------
```

El usuario puede ingresar un mensaje a codificar, con su respectivo vector de desplazamiento y la opción elegida. Al presionar enter, si la cadena de ingreso es correcta, el programa devolverá:

- El mensaje procesado, incluye el bit de paridad (0/1).
- La palabra clave codificada para utilizar en la decodificación.
- La cantidad de caracteres procesados del mensaje (incluyendo los espacios).

```
occ4g2@Alysa:~ $ ./tpfinal
#=========================================================#
#                                                         #
#           MÁQUINA DE CODIFICACIÓN DE MENSAJES           #
#            Cifrado Cesar mejorado, método OC            #
#                                                         #
#=========================================================#

Ingrese el mensaje con alguno de los siguientes formatos de entrada:
(mensaje a codificar o decodificar;[0-9];[0-9];[0-9];[0-9];[0-9];c/d)
(mensaje a decodificar;clave;d)
---------------------------------------------------------
el hombre de negro huia a traves del desierto;3;4;2;7;6;c
---------------------------------------------------------
Mensaje procesado:
hp jvsevg kk qiiyu kykh g wvcckv hgs jhwklxws1
---------------------------------------------------------
Palabra clave para decodificar:
---------------------------------------------------------
hslpu
---------------------------------------------------------
Caracteres procesados:
45
occ4g2@Alysa:~ $
```

Para la decodificación del mensaje el usuario puede optar por ingresar el vector de desplazamientos usado en la codificación o la palabra clave suministrada por el proceso de codificación.

En ambos casos, si los datos ingresados son correctos, el programa mostrará por pantalla:

- El mensaje decodificado.
- El vector de desplazamientos usado en la codificación.
- La cantidad de caracteres procesados del mensaje (incluyendo los espacios).

```
occ4g2@Alysa:~ $ ./tpfinal
#============================================================#
#                                                            #
#           MÁQUINA DE CODIFICACIÓN DE MENSAJES              #
#            Cifrado Cesar mejorado, método OC               #
#                                                            #
#============================================================#

Ingrese el mensaje con alguno de los siguientes formatos de entrada:
(mensaje a codificar o decodificar;[0-9];[0-9];[0-9];[0-9];[0-9];c/d)
(mensaje a decodificar;clave;d)
------------------------------------------------------------
hp jvsevg kk qiiyu kykh g wvcckv hgs jhwklxws1;3;4;2;7;6;d
------------------------------------------------------------
Mensaje procesado:
el hombre de negro huia a traves del desierto
------------------------------------------------------------
El vector de desplazamiento usado:
------------------------------------------------------------
34276
------------------------------------------------------------
Caracteres procesados:
45
occ4g2@Alysa:~ $
```

```
occ4g2@Alysa:~ $ ./tpfinal
#=========================================================#
#                                                         #
#           MÁQUINA DE CODIFICACIÓN DE MENSAJES           #
#            Cifrado Cesar mejorado, método OC            #
#                                                         #
#=========================================================#

Ingrese el mensaje con alguno de los siguientes formatos de entrada:
(mensaje a codificar o decodificar;[0-9];[0-9];[0-9];[0-9];[0-9];c/d)
(mensaje a decodificar;clave;d)
-----------------------------------------------------------
hp jvsevg kk qiiyu kykh g wvcckv hgs jhwklxws1;hslpu;d
-----------------------------------------------------------
Mensaje procesado:
el hombre de negro huia a traves del desierto
-----------------------------------------------------------
El vector de desplazamiento usado:
-----------------------------------------------------------
34276
-----------------------------------------------------------
Caracteres procesados:
45
occ4g2@Alysa:~ $
```

En el caso del ingreso erróneo del bit de paridad o de la omisión de una letra en el mensaje a decodificar el programa muestra el mensaje de error y finaliza.

```
occ4g2@Alysa:~ $ ./tpfinal
#=========================================================#
#                                                         #
#           MÁQUINA DE CODIFICACIÓN DE MENSAJES           #
#            Cifrado Cesar mejorado, método OC            #
#                                                         #
#=========================================================#

Ingrese el mensaje con alguno de los siguientes formatos de entrada:
(mensaje a codificar o decodificar;[0-9];[0-9];[0-9];[0-9];[0-9];c/d)
(mensaje a decodificar;clave;d)
-----------------------------------------------------------
hp jvsevg kk qiiyu kykh g wvcckv hgs jhwklxws0;3;4;2;7;6;d
-----------------------------------------------------------
Error en el bit de paridad, ingreso incorrecto
-----------------------------------------------------------
occ4g2@Alysa:~ $
```

```
occ4g2@Alysa:~ $ ./tpfinal
#=========================================================#
#                                                         #
#          MÁQUINA DE CODIFICACIÓN DE MENSAJES            #
#           Cifrado Cesar mejorado, método OC             #
#                                                         #
#=========================================================#

Ingrese el mensaje con alguno de los siguientes formatos de entrada:
(mensaje a codificar o decodificar;[0-9];[0-9];[0-9];[0-9];[0-9];c/d)
(mensaje a decodificar;clave;d)
---------------------------------------------------------
hp jvsevg kk qiiyu kkh g wvcckv hgs jhwklxws1;hslpu;d
---------------------------------------------------------
Error en el bit de paridad, ingreso incorrecto
---------------------------------------------------------
occ4g2@Alysa:~ $
```

Subrutinas principales

Nombre	fn_procesar_cadena_ingresada
Descripción	Procesa la cadena ingresada por el usuario, separando el mensaje a procesar, el vector de desplazamientos o la palabra clave y la opción ingresada (c/d).
Parámetros	r0 -> puntero a la cadena ingresada r1 -> puntero a total_car (donde se guarda la cantidad de caracteres del mensaje) r2 -> puntero al vector de desplazamientos r3 -> puntero a cadena opción.
Return	Esta función no tiene valor de retorno

Nombre	fn_codificar_letra
Descripción	Codifica una letra de acuerdo a un desplazamiento dado. Esta función es el centro del programa. Para codificar un texto se llama letra por letra a esta subrutina.
Parámetros	r1 -> carácter a codificar r2 -> valor del desplazamiento
Return	r0 -> el carácter codificado

Nombre	fn_decodificar_letra
Descripción	Decodifica una letra de acuerdo a un desplazamiento dado. Esta función es la inversa de la anterior. Para decodificar un texto se llama letra por letra a esta subrutina.
Parámetros	r1 -> carácter a decodificar r2 -> valor del desplazamiento
Return	r0 -> el carácter decodificado

Nombre	fn_codificar_texto
Descripción	Transforma una cadena de texto codificando o decodificando según la opción elegida. Llama a las subrutinas fn_codificar_letra, fn_decodificar_letra, fn_agregar_bit_paridad y fn_borrar_bit_paridad.
Parámetros	r0 -> puntero a la cadena a codificar / decodificar. r1 -> puntero donde se guarda la cadena procesada. r2 -> valor del desplazamiento r3 -> opción (codificar / decodificar).
Return	Esta función no tiene valor de retorno

Subrutinas secundarias

Nombre	fn_print_linea
Descripción	Imprime una línea decorativa en la pantalla somo delimitador de las salidas.
Parámetros	Esta función no recibe parámetros
Return	Esta función no tiene valor de retorno

Nombre	fn_print
Descripción	Imprime por pantalla una cadena de caracteres ascii pasadas por parámetro.
Parámetros	r1 -> puntero a la cadena de caracteres ascii r2 -> cantidad de caracteres a imprimir
Return	Esta función no tiene valor de retorno

Nombre	fn_input
Descripción	Guarda en una cadena los caracteres ascii ingresados por teclado por el usuario.
Parámetros	r1 -> puntero a la cadena de caracteres ascii donde guarda r2 -> cantidad de caracteres a ingresar
Return	Esta función no tiene valor de retorno

Nombre	fn_generar_clave
Descripción	Genera una clave alfabética o palabra clave para poder decodificar un mensaje cifrado, sin conocer el vector de desplazamientos, con los primeros 5 caracteres de la cadena ingresada. Utiliza un vector de desplazamientos interno.
Parámetros	r0 -> puntero a la cadena ingresada r1 -> puntero a la cadena clave r2 -> puntero al vector de desplazamiento interno
Return	Esta función no tiene valor de retorno

Nombre	fn_descifrar_clave
Descripción	Descifra la clave alfabética y lo guarda en clave descifrada. Utiliza el mismo vector de desplazamiento interno.
Parámetros	r0 -> puntero a la clave r1 -> puntero a la clave descifrada r2 -> puntero al vector de desplazamiento interno
Return	Esta función no tiene valor de retorno

Nombre	fn_guardar_desplazamiento
Descripción	Calcula el desplazamiento de acuerdo con la palabra clave y las primeras 5 letras de la cadena ingresada para decodificar. Guarda el resultado en el vector de desplazamientos. Guarda el resultado como caracteres ascii en una cadena para poder mostrar por pantalla el desplazamiento usado.
Parámetros	r0 -> puntero a la cadena ingresada r1 -> puntero a clave_des r2 -> puntero a cadena_des r3 -> puntero al vector de desplazamiento
Return	Esta función no tiene valor de retorno

Nombre	fn_agregar_bit_paridad
Descripción	Agrega el bit de paridad al final de la cadena codificada (0 si la cadena tiene cantidad par de caracteres, 1 si la cadena tiene cantidad impar de caracteres).
Parámetros	Esta función no recibe parámetros
Return	Esta función no tiene valor de retorno

Nombre	fn_verificar_bit_paridad
Descripción	Verificar que el bit de paridad del mensaje a decodificar sea correcto.
Parámetros	r0 -> puntero a la cadena para decodificar r1 -> total de caracteres de la cadena
Return	r0 -> retorna 0 si es correcto el bit de paridad, 1 si es incorrecto

Nombre	fn_borrar_bit_paridad
Descripción	Borra el ultimo carácter de la cadena procesada si esta decodificando para no mostrar por pantalla el bit de paridad
Parámetros	Esta función no recibe parámetros
Return	Esta función no tiene valor de retorno

Nombre	fn_cantidad_caracteres
Descripción	Tranforma en cadena de caracteres ascii la cantidad de caracteres procesados
Parámetros	r0 -> número con la cantidad de caracteres r1 -> puntero a la cadena donde se guarda
Return	Esta función no tiene valor de retorno

Nombre	fn_guardar_des_en_cadena
Descripción	Transforma el vector de desplazamientos en cadena de caracteres ascii para motrar por pantalla
Parámetros	r0 -> puntero al vector de deplazamiento r1 -> puntero a la cadena cadena_des donde se guarda
Return	Esta función no tiene valor de retorno

Dificultades y aciertos

Al realizar el programa nos hemos encontrado con que varios del grupo tienen conocimiento en programar, por lo cual el pensar el programa en alto nivel y luego llevarlo a bajo nivel fue mucho más fácil.

También se fueron compartiendo ideas y opciones para llevar a cabo el programa.

Las dificultades fueron que, aunque hemos hecho bastante seguimiento de la codificación fue difícil poder encontrar momentos de encuentros en los cuales poder ver entre todos el Código y poder así no realizar todo mediante mensajes.

A pesar de esto el grupo logró realizar el trabajo de una manera correcta, debatiendo sobre algunos conceptos que estaban en el trabajo práctico y que luego pudimos llevar adelante unificando criterios.

Para finalizar este trabajo podemos decir que en lo que corresponde al trabajo como equipo hemos tenido aciertos y errores, en momentos se nos dificulto entender e incluso ponernos de acuerdo para poder realizar algunas tareas en el mismo.

Dado esto se notó que hubo alguien que logró llevar adelante, motivando al equipo.

Por último, hemos aprendido mucho sobre este trabajo, utilizando rutinas que quizás no hemos usado antes o desarrollando algoritmos que permiten realizar distintos tipos de operaciones.

Conclusiones generales

Este trabajo fue el cierre de la materia Organización del Computador, en él aplicamos los conocimientos adquiridos a lo largo del cuatrimestre, aplicando técnicas de programación que no estábamos acostumbrados. Al programar en lenguaje de bajo nivel aprendimos a gestionar los recursos limitados del hardware como los registros, a crear subrutinas que en lenguajes de alto nivel ya vienen incorporadas al lenguaje, como por ejemplo ciclos de repeticiones o condicionales, divisiones a través de restas sucesivas, uso del stack para el resguardo de los datos entre llamadas a las funciones. También aplicamos métodos de interrupciones llamando al SO para gestionar la interacción con el usuario a través de operaciones de I/O.

Aprender a programar en esamblador nos ayudó a comprender mejor como trabaja internamente la computadora y como los métodos y funciones de los lenguajes de alto nivel se pueden codificar en lenguaje de bajo nivel con la repetición de instrucciones básicas.