Trabajo Practico Integrador
Grupo 10

Organización del Computador Comision 3

Integrantes del grupo: Bruno Berini, Lucas Ledesma, Santiago Martinez

Comisión: COM-0011

El trabajo solicitado para la materia consiste en realizar un programa que permita codificar o decodificar un mensaje usando el cifrado cesar mejorado, ya que, en vez de pasarle una clave, se pasa un vector de claves, y además se tiene en cuenta el bit de paridad

El trabajo lo dividimos en tres partes, primero comenzamos realizando las subrutinas para extraer el mensaje, la clave y la opción. Luego continuamos por la parte de codificar, y por último decodificar.

1ra Parte: ExtraerMensaje; ExtraerClave; ExtraerOpcion

Para la subrutina de extraer Mensaje, lo que se hace es utilizar un ciclo que compare el siguiente byte cargado en registro r2 y se va guardando en etiqueta “mensaje” a medida que avanza el desplazamiento. El ciclo termina cuando el byte es “;” o es un bit de paridad, como 0 o 1. Respecto al bit de paridad, se asume que cuando se introduce es porque se va a decodificar. 0 es si la cantidad de letras del mensaje es par, y 1 si es impar. No se cuentan los espacios cuando se verifica el mismo.

En el caso de extraer Clave, se utiliza la última posición recorrida que se guarda en la etiqueta “conEspacios” (guarda el largo del mensaje contando los espacios). Antes de iniciar el ciclo para guardar la clave, se determina si es una clave numérica, o si es una clave con palabras. Esto lo decidimos viendo si se introdujo un bit de paridad (por defecto en bit de paridad hay un 8). Si es igual a 8, no se introdujo bit de paridad y va a extraerNros, sino, agrega a la ultima posición 2 nros para saltar el “;” y entra a extraerPalabra.
extraerNros termina cuando se detecta una “c” o una “d” y extraerPalabra termina cuando se detecta un “;”. A tener en cuenta, que también se puede dar un mensaje de error si la cantidad de dígitos es menor a 5, esto solo ocurre si se quiere codificar una palabra.

Para extraer opción, simplemente se utiliza la última posición cargada en el registro que cargaba la clave, y se lo carga en la dirección de memoria de “opción” designada en la parte de datos del programa. Sin embargo, como extraerPalabra termina en “;” es posible que no haya tomado una c o una d, es por eso que se hacen esas comparaciones al principio, y si no es igual, se avanza un byte de desplazamiento más para que tome una c o una d.

En estas 3 subrutinas se utilizó siempre el direccionamiento relativo a registro con desplazamiento para cargar carácter por carácter en las posiciones correctas.

2da Parte: Codificar

Para este apartado del programa, creamos 8 subrutinas. La principal, que sería “Codificar” comprueba que la opción cargada sea ‘c’. Si no es ‘c’ sale de “codificar” y se va al main para continuar con la siguiente subrutina.
Sino primero usa la subrutina de convertir_ascii_a_entero (se asume que por posición solo hay un digito, del 0 al 9), luego carga las direcciones de memoria necesaria en los registros en la subrutina cargarRegCodificar. Una vez cargada las direcciones de memoria de la clave y el mensaje, entra a una subrutina, cmpAbcMsje, que comprueba si el carácter de mensaje está en mayúscula, o está en minúscula. Dependiendo de eso, entra en uno u otro ciclo para aplicar el desplazamiento correspondiente, y si hay un espacio simplemente lo saltea con saltar_espacio. Este ciclo termina cuando el desplazamiento es igual al largo del mensaje.

Tanto en el ciclo de mayúsculas como en el ciclo de minúsculas el funcionamiento es similar, se suma el desplazamiento obtenido en la clave, y simplemente cambian algunas comparaciones que se hacen para determinar si el desplazamiento luego de sumar, se pasa de la z o Z. En caso que se pase, se resta 26 que es el total del largo del abecedario, para mayúsculas o para minúsculas.
Luego se utiliza la subrutina de mostrarMensaje y mostrarCantCaracteres para terminar con el programa.
En mostrarCantCaracteres asumimos que el mensaje no supere los 100 caracteres, de lo contrario no podría ser mostrado en pantalla la cantidad. No se cuentan los espacios.

3ra Parte: Decodificar

Para la parte de decodificar (en caso de ser la opción d la seleccionada por el usuario) se usan 3 subrutinas principales (c/u contiene varias dentro de ella para ejecutar los procesos necesarios). Son: calcularParidad; calcularDesplazamiento; decodificarMensaje.
En calcularParidad se verifica que la cantidad de caracteres efectivamente sea par o impar según el bit de paridad introducido. Para esto se va restando de a 2 la cantidad de letras del mensaje "sinEspacios", si llega a 0 es par, si llega a 1 es impar. Si coincide con el bitUsuarioInput, sigue con la siguiente subrutina, y si NO coincide, lanza el mensaje de error para corregir el bit de paridad.

Para calcularDesplazamiento, primero se hace un llamado a esta subrutina donde se cargan los registros necesarios para ejecutar los procesos. La lógica del programa se basa en tomar un carácter del mensaje, ver si es mayuscula o minúscula, y entrar en el ciclo correspondiente para poder calcular y guardar el desplazamiento necesario. El ciclo termina cuando el desplazamiento al cargar un byte de la clave es igual al largo de la palabraClave introducida. Se usan subrutinas auxiliares para pasar la clave a mayus o minúsculas y verificar si el desplazamiento da en negativo, para realizar la resta al reves. El calculo es: ascii de mensaje – ascii de palabraClave = desplazamiento. Se asume que el desplazamiento no supere los 2 dígitos por posición, es decir, de 0 a 9.

Luego de calcular el desplazamiento correspondiente, se guarda en claveInt para luego entrar a decodificarMensaje.
En esta subrutina, también se verifica en que ciclo se debe entrar (mayus o minus) y el desplazamiento se aplica restando al ascii del mensaje el nro correspondiente al mismo. En caso que se pase (sea menos que A o a) se le suma la cantidad de caracteres del abecedario, es decir 26, que es al revés de decodificar (donde se le restaba 26)

Finalmente termina con mostrarMensaje y con mostrarDesplazamiento ya que se usó decodificar.
MostrarDesplazamiento toma la clave guardada en asciz y la muestra en pantalla directamente debajo del mensaje decodificado.

A lo largo del programa también se usan unas subrutinas secundarias como:
Salida: simplemente hace pop {lr} y bx lr.
IniciarRegistros: desde el 0 al 12, mueve 0 a todos los registros
VerificarClaveLength: comprueba que si se extrajeronNros, la cantidad sea mayor o igual a 5, caso contrario lanza mensaje de error.

Convertir_ascii_a_entero: le resta a cada byte de la clave numérica el hexadecimal 30, siempre asumiendo que se trata de solo un digito por desplazamiento.

El desarrollo de este programa fue muy útil para aprender cómo se manejan los datos en memoria, y uno de los errores usuales que teníamos era de violación de segmento, generalmente por colocar o simplemente cambiar de orden una etiqueta, dependiendo donde se coloque, puede alterar la ejecución del programa, por lo que hay que tener cuidado a la hora de definir tipo de datos y la posición donde se alojan los mismos. Sin embargo, varios de estos problemas pudieron ser resueltos debugueando y encontrando el momento exacto que un dato era mal cargado en el registro, lo que desencadenaba luego una serie de errores.

El trabajo se encuentra cargado en el directorio **tpTerminado, archivo tp.asm**