Organización del computador

Informe

TP FINAL

Comisión 03 UNGS 2021

GRUPO 5

García, Tobías Joaquín

Profesoras: Martha Semken & Viviana Nakatsuka

Datos utilizados:

entrada: .asciz donde se guarda la entrada del usuario

salida: .asciz donde se guarda la salida del código codificado o decodificado

mensaje: .asciz que sirve para guardar solo el mensaje extraído de la entrada del usuario

clave_texto: .asciz que sirve para guardar solo la clave extraída de la entrada del usuario en texto

clave_numero: .int que sirve para guardar la conversión de ascii a entero de clave_texto

f0, f1, f2, f3, f4: . word que sirven para mover espacio en memoria y que no se sobrescriban otros datos

opción: .asciz que sirve para guardar solo la opción (c/d) extraída de la entrada del usuario

c1 y **c2:** .asciz que funcionan como cartel para mostrar los caracteres procesados

encriptado: .asciz que sirve como cartel para decir que este mensaje está encriptado en caso de decodificación

desplazamiento_usado: .asciz que sirve como cartel para decir el desplazamiento usado en caso de decodificación

cartel_mensaje: .asciz que sirve como cartel para mostrar el mensaje decodificado

salto_de_linea: .asciz que guarda un salto de línea para usarse cuando se necesite

Funciones utilizadas:

extraer_mensaje: función que se encarga de extraer el mensaje contenido en la entrada del usuario

extraer_mensaje:
cargo **mensaje** en registro
hago un bucle que recorre la entrada y carga los caracteres en **mensaje** hasta encontrar un **";"**
si encuentro punto y coma salgo del bucle y agrego un salto de línea a **mensaje**

extraer_clave: función que se encarga de extraer la clave de codificación contenida en la entrada del usuario

extraer_clave:
cargo **clave_texto** en registro
defino una **bandera** en **r12**, 0 si es **positivo** y 1 si es **negativo**
pregunto si el primer carácter es un "**-**" y pongo la **bandera** en su valor correspondiente
recorro la cadena con un bucle y cargo sus caracteres en **clave_texto** hasta encontrar un punto y coma

extraer_opcion: función que se encarga de extraer la opción (c/d) contenida en la entrada del usuario

extraer_opcion:
cargo opción en registro
hago un bucle que recorre y carga los caracteres en opción hasta encontrar un punto y coma

(esta función está en orden lineal y se podría haber hecho en orden constante ya que es simplemente cargar el primer carácter del resto de la entrada, pero decidí mantener la misma estructura que las otras)

ascii_a_entero: función que se encarga de convertir un . asciz con valores numéricos a un .int con su valor numérico correspondiente

ascii_a_entero:

cargo **clave_texto** en r0
pongo **r1** en 0 y **r10** en 10
entro en un **bucle**
cargo en r2 el primer carácter
pregunto si el primer carácter de **r0** es un espacio, si lo es, me voy a terminar la función
multiplico a **r1** en 10 (la primera vez no se multiplica ya que es 0)
le resto 0x30 a **r2** para conseguir su valor correspondiente
le sumo r2 a r1
aumento r0 en uno para pasar al siguiente carácter
vuelvo al **bucle**
para terminar la función, cargo clave_numero en r0 y le guardo r1, donde se ubica el valor de clave_texto en entero

entero_a_ascii: función que se encarga de convertir un número entero en un ascii para poder imprimirlo por pantalla (devuelve 2 registros con un carácter ascii que luego son utilizados para imprimir los caracteres procesados)

entero_a_ascii:

pongo r2 en 0
comparo r11 con 10, si es menor termino la función
sino, entro en un bucle
le resto 10 a r11
le sumo 1 a r2
pregunto si r11 es mayor a 10, si lo es, vuelvo al bucle
al terminar la función
le sumo a r2 y a r11 un 0x30 para conseguir su carácter en ascii

conseguir_bit_paridad: función que se utiliza dentro de un bucle de la función codificar para evaluar cada carácter del mensaje y verificar la paridad total de la suma de los caracteres de la cadena.

conseguir_bit_paridad:
entro en **bucle**:
le **resto** #2 al **registro auxiliar**
comparo el **registro auxiliar** con 1
si es **mayor** vuelvo al **bucle**
si es **igual** me voy a **sumar_bit_paridad**
si no, termino la función
sumar_bit_paridad:
si **r9** es un 0, lo pongo en 1 y termino
si **r9** es un 1, lo pongo en 0 y termino

conseguir_desplazamiento: función que se encarga de conseguir el desplazamiento que se utilizó para codificar el mensaje, en función de la palabra clave dada.

conseguir_desplazamiento:
inicializo en **r2** el **desplazamiento**
entro en **bucle_1**:
direcciono en **r0** la **palabra**
direcciono en **r1** el **mensaje**
le **sumo** 1 al **desplazamiento** en **r2**

entro en **bucle_2**:
cargo en **r3** el primer **carácter** de la **palabra**
cargo en **r4** el primer **carácter** del **mensaje**
si **r3** es un espacio, **termino** la función
si **r4** es un salto de línea, vuelvo al **bucle_1** para reiniciar
sino, le **resto** el **desplazamiento** al **carácter** de **r4**
si **r4** es **menor** que **'a'**, le **sumo** #26 para **voltear abecedario**
comparo **r3** con **r4**, si son **iguales** vuelvo al **bucle_2**
sino, **reinicio_palabra**
reinicio_palabra:
direcciono en **r0 clave_texto** para **reiniciar** la **palabra**
vuelvo al **bucle_2**

termino_la_funcion:
direcciono en **r0 clave_numero**
guardo el **desplazamiento** en **r0**

codificar: función que se encarga de codificar el mensaje, dependiendo de la clave y la opción que se obtuvo en la entrada del programa. Esta función se reutiliza para la decodificación, ya que lo único que cambia es el signo de la clave (si se suma o se resta) ya que decodificar un mensaje que fue codificado en clave 5, por ejemplo, es lo mismo que codificar el mensaje en clave – 5, ya que estaríamos logrando que el mensaje vuelva a su estado original.

codificar:
cargo la dirección de la **clave** en **r0** y su valor en **r1**
direcciono en **r2** el **mensaje**
direcciono en **r3** la **salida**
inicializo un **contador** en **r11** para cantidad de caracteres
inicializo en **r9** un **contador** para la **paridad**
entro en **bucle**:
cargo en **r4** el digito
lo comparo con un salto de línea, si es igual, termino la función
sino, **aumento** el contador en 1
muevo el carácter a **r6** para tener un carácter **auxiliar**
llamo a la función **conseguir_bit_paridad**
comparo **r4** con un espacio
si es un espacio, cargo el carácter sin modificarlo y vuelvo al bucle
sino, me voy a cambiar el digito
cambiar:
comparo la **bandera r12** con 1
si es 1, me voy a **es_negativo**
sino, me voy a **es_positivo**
es_negativo:
le **resto** a **r4** la **clave**
si es **menor** que el valor de **“a”**
me voy a **menor_que_a**
sino, lo cargo en **salida** y vuelvo al **bucle**

menor_que_a:
le sumo a **r4** un **26** (para dar la vuelta al abecedario)
lo cargo en la **salida**
vuelvo al **bucle**

es_positivo:
le **sumo** a **r4** la **clave**
si es **mayor** que el valor de **“z”**
me voy a **mayor_que_z**
sino, lo cargo en **salida** y vuelvo al **bucle**
mayor_que_z:
le resto a **r4** un **26**
lo cargo en **salida**
vuelvo al **bucle**

fin:
agrego un espacio a la cadena
le sumo 0x30 al **bit de paridad** y lo **agrego** a la **cadena**
termino la función

***MAIN*:**

- Pido la **entrada** al usuario
- Invoco las funciones necesarias para **cargar** todos los **datos necesarios** (extraer_mensaje, extraer_clave, etc)
- Cargo en registro el carácter de **opción** y lo comparo para saber si se debe codificar o decodificar el mensaje
- Si se debe codificar me voy a la etiqueta **opción_codificar**
  opción_codificar:
  - llamo a la función **codificar** para generar la salida
  - llamo a la función **entero_a_ascii** para convertir a ascii el numero guardado en **r11** el cual contaba los caracteres procesados
  - guardo los caracteres numéricos en **c1 y c2,** los cuales eran .asciz para mostrar en pantalla la cantidad de caracteres procesados
  - imprimo la **salida**
  - imprimo la cantidad de caracteres procesados de **c1** y **c2**
  - me voy a la etiqueta **salir**, termino el programa

- Si se debe decodificar me voy a la etiqueta **opción_decodificar**
  opción_decodificar:
  - Comparo el registro bandera r12, si es 0 lo pongo en 1 y si es 1 lo pongo en 0 (esto lo hago porque al decodificar, si se codifico en 7, para volver al estado original se debe invertir el signo, es decir, en vez de sumar 7 le tengo que restar 7)
  - Una vez invertido el signo, llamo a la función codificar para que devuelva la cadena a su estado original
  - llamo a la función **entero_a_ascii** para convertir a ascii el numero guardado en **r11** el cual contaba los caracteres procesados
  - guardo los caracteres numéricos en c1 y c2, los cuales eran .asciz para mostrar en pantalla la cantidad de caracteres procesados
  - Imprimo por pantalla el .asciz **encriptado**
  - Imprimo por pantalla el .asciz **cartel_mensaje**
  - Posteriormente imprimo por pantalla el .asciz **salida** donde está el código decodificado
  - Imprimo por pantalla la cantidad de caracteres procesados de **c1** y **c2**
  - Me voy a la etiqueta **salir**, termino el programa

- Si no hay opción, quiere decir que se debe conseguir el desplazamiento con la palabra clave, me voy a la etiqueta **opción_decodificar_desplazamiento**
  - Llamo a la función conseguir_desplazamiento
  - Llamo a la función entero_a_ascii para convertir el desplazamiento conseguido en la función anterior guardo los caracteres en el asciz desplazamiento_usado
  - Pongo la bandera de r12 en 1 para indicar negativo llamo a la función codificar
  - Llamo a la función entero_a_ascii para conseguir los caracteres procesados
  - Guardo los caracteres procesados en c1
  - Imprimo por pantalla encriptado, desplazamiento_usado, cartel_mensaje, salida, c1 y c2
  - Termino el programa

Dificultades:

Para empezar, he de decir que es un trabajo que se pudo desarrollar mucho mejor, pero se vio ajustado ante los tiempos de múltiples inconvenientes y su realización debía ser grupal, no obstante, este trabajo fue realizado por una sola persona y la confianza en la división de tareas de cada integrante lo complicó aún más, ya que solo una parte del cuerpo estaba cumpliendo con su responsabilidad y se dio visibilidad de eso en una fecha bastante moderada.

En cuanto a las dificultades encontradas en el desarrollo del trabajo, no hubo ningún tipo de problema con el código o la sintaxis del mismo, ni con los algoritmos, los cuales fueron todos resueltos de manera sencilla. Todos los problemas encontrados fueron de memoria.

El problema más importante fue el hecho de que se sobrescribía la clave del mensaje con la opción, es decir, cuando se extraía la clave, se guardaba en **clave_numero** su valor numérico, luego, se debía extraer la opción. Al cargar en memoria la opción de la entrada del usuario, esta sobrescribía el .int **clave_numero**, convirtiéndola en una “c” o una “d”, 0x63 o 0x64.
Este problema se solucionó creando múltiples .word y cargándolos en memoria, logrando así hacer el espacio suficiente para que no se sobrescriban

Otro problema importante fue que las salidas por consola del programa se imprimían muy mal, por ejemplo, se imprimía el mensaje codificado y luego la clave, la cual no debería salir por pantalla.
El problema con esto era que la cantidad de espacios del .asciz y el largo de la salida el cual se ubica en r2, no eran el mismo, la cantidad de espacios era menor que la cantidad de caracteres de la salida, por lo que el programa terminaba tomando otros espacios de la memoria y los imprimía por pantalla.