Laboratorio No. 2

Programación en ensamblador MIPS

Técnica de Cifrado Afín (Affine Cipher)

Autor: Santiago Correa Marulanda

C.C: 1033486496

Arquitectura de Computadores y Laboratorio

Fredy Alexander Rivera Velez

22/09/2024

Contenido

INTRODUCCION

En esta practica se implementará la técnica o modelo se cifrado afín con ayuda del lenguaje de programación MIPS Assembler (Microprocessor without Interlocked Pipeline Stages).

La técnica de cifrado afín se basa en un tipo de cifrado por sustitución, donde construiremos nuestro propio alfabeto, con un orden y cada elemento le daremos un valor numérico, luego, por medio de una transformación lineal en aritmética modular, reemplazaremos el valor del numerico del elemento o carácter por el valor recién calculado.

OBJETIVO

Con este laboratorio se busca que el estudiante adquiera habilidades y conocimiento acerca de las instrucciones nativas y pseudo Intrucciones que contiene la arquitectura MIPS Assembler, para llevar a cabo el desarrollo de programas y todo esto, llevarlo al diseño e implementación de la técnica de cifrado afín.

DESCRIPCIÓN

Antes de explicar el planteamiento del problema, es necesario tomar en cuenta lo siguiente.

Para que el programa funcione, debemos crear un alfabeto que contenga los caracteres que vamos a usar, en este caso, haremos uso de 98 caracteres, donde contiene las letras del alfabeto (a-z) minúsculas y mayúsculas (A-Z), números (0-9), símbolos y signos de puntuación (=,",·,$,%, etc) y 3 caracteres especiales, tabulación "\t", new line "\n" y return carriage "\r" además de incluir, el espacio. Luego a cada carácter le asignaremos un valor numerico, de la siguiente manera

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t

20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39
u	v	w	x	y	z	A	B	C	D	E	F	G	H	I	J	K	L	M	N

40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59
O	P	Q	R	S	T	U	V	W	X	Y	Z	\t	\n	\r		!	"	#	$

60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8

80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97
9	:	;	<	=	>	?	@	[	\	]	^	_	`	{	|	}	~

Allí están los 98 caracteres imprimibles que haremos uso en esta práctica, luego con los dos números ingresados por el usuario, que llamaremos a, b, serán nuestras claves multiplicativas y aditivas respectivamente, a debe ser coprimo con 98, mientras b debe ser un numero entre 0 y 97, luego el carácter a cifrar, lo identificaremos con su numero asignado y le llamaremos x, con estos parametros llevamos a cabo la siguiente ecuación:

$$c(x) = (ax + b)\ mod\ 98$$

Este nos dará un numero como resultado, buscaremos el numero en el alfabeto anteriormente diseñado y observaremos el carácter que se le asigna a este valor y este será el carácter "a" cifrado,

por ejemplo:
Carácter a cifrar: “a”

“a” => x = 0

a = 5 (valor ingresado por usuario)

b = 8 (valor ingresado por usuario)

C(0) = (5*0 + 8) mod 98 = 8

8 => i (Carácter cifrado)

Con esto ya claro, podemos dar paso a el problema

El problema plantea lo siguiente:
El programa debe abrir un archivo .txt de nombre “input”, el cual contendrá un texto en inglés, de máximo 1 KiB (1024 Bytes), una vez lo abierto debe leer cada carácter y para realizar su respectivo patrón de cifrado y llevar este mensaje a un archivo .txt de nombre “criptogram”, a continuación deberá descifrar este texto cifrado y llevarlo a un archivo .txt “decoded” cuya información debe ser exactamente igual al archivo original “input”.

Para realizar el descifrado, hará uso de la siguiente formula

$$p(x) = a^{-1}(C(x) - b)) \, mod \, 98$$

Donde a^{-1} será el inverso multiplicativo de la clave multiplicativa a ingresada por el usuario en modulo 98 ($a.a^{-1}$ mod 98 ≡ 1) y C(x) es el valor hallado para el carácter ya cifrado

DESARROLLO

Para implementar el problema en Lenguaje ensamblador MIPS se ha usado el simulador MARS y se ha diseñado cada función o procedimiento que se encargará cada uno de realizar una tarea, usando la técnica de principio de única responsabilidad. Para comenzar, iniciaremos explicando en primer lugar la fase de descrifrado y finalmente la parte de cifrado.

DESCIFRADO

Find_char

Para el desarrollo del código, se ha decidido empezar con la función Find_char.

Esta se encargá de recibir un carácter almacenado en el registro $a1 y buscarlo en el alfabeto diseñado, en el que su dirección base será almacenado en el registro $a0, para finalmente retornar la posición de este carácter en el registro $v0

```
Find_char:
        addi $sp, $sp, -12
        sw $a0, 0($sp)
        sw $t1, 4($sp)
        sw $t0, 8($sp)

        addi $v0,$zero, -1      # Inicializacion de $v0 = -1
        add $t0, $zero, $zero   # Inicializacion indice de barrido
Loop1:  lbu $t1, 0($a0)         # $t1 = String[i]
        beq $t1, $zero, End_FC  # String[i] = NULL?
        bne $t1, $a1, Next_item # El caracter a buscar NO es el mismo que el caracter del string donde estamos ubicados?
        move $v0, $t0           # Copia indice de barrido en resultado
        j End_FC
Next_item:
        addi $a0, $a0, 1        # puntero a String[i + 1]
        addi $t0, $t0, 1        # Actualizar indice de barrido
        j Loop1
End_FC:


        lw $t0, 8($sp)
        lw $t1, 4($sp)
        lw $a0, 0($sp)
        addi $sp, $sp, 12

Exit_FC: jr $ra
```

Create_decoded

Esta función se encarga de recibirá un valor numérico, entre 0 y 97, almacenado en el registro $a1, luego recorrerá el alfabeto que nuevamente su base estará en el registro $a0, al recorrer esté vector, se posicionará en el carácter, este carácter tendrá asignado el valor numerico almacenado en $a1, para llevarlo a un buffer, que será el que escribiremos en el texto “decoded”.

Si $a1 = 24, recorrerá el vector alfabeto hasta que llegue a la posición 24 y retornará el carácter de esta posición = y. luego este carácter se almacenará en el buffer y se imprimirá en “decoded”.

```
.data
decrypted_file: .asciiz "C:\\Users\\HP\\Desktop\\Assembler\\Mi desarrollo\\decoded.txt"
alfabeto: .asciiz "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ\t\n\r !\"#$%&'()*+,-./0123456789:;<=>?@[\\]^_`{|}~"

decrypted_buffer: .space 1 # Espacio para almacenar el carácter encontrado
```

```
    la $t0, alfabeto # Puntero a la cadena alfabeto
    li $t1, 0 # Inicializamos el contador de índice en 0

Loop_decrypted:
    lb $t2, 0($t0) # Cargar el byte actual de alfabeto en $t2
    beq $t2, $zero, End_decrypted # Si es el fin de la cadena (byte 0), salir
    beq $t1, $a1, Encontrado_decrypted # Si el índice coincide con $a1, encontramos el carácter
    addi $t1, $t1, 1 # Aumentamos el índice
    addi $t0, $t0, 1 # Avanzamos en la cadena alfabeto
    j Loop_decrypted

Encontrado_decrypted:
    lb $t2, 0($t0) # Cargar el carácter encontrado en $t2
    sb $t2, decrypted_buffer # Guardar el carácter en el buffer

    li $v0, 13 # Syscall para abrir el archivo
    la $a0, decrypted_file # Dirección del archivo
    li $a1, 9 # Modo de apertura (9 = agregar)
    li $a2, 0 # Sin permisos especiales
    syscall
    move $a0, $v0 # Guardar descriptor del archivo en $a0

    li $v0, 15 # Syscall para escribir en el archivo
    la $a1, decrypted_buffer # Cargar el mensaje a escribir (carácter en buffer)
    li $a2, 1 # Escribir solo un byte (el carácter)
    syscall

    li $v0, 16 # Syscall para cerrar el archivo
    syscall

End_decrypted:
```

En la anterior imagen se puede ver la base de la función (no se puede observar la reserva y carga de datos en la pila y demás)

Ecuación P(x)

El valor numerico que se envia a la función anterior, no es más que el valor resultante de aplicar la ecuación:

$$p(x) = a^{-1}(C(x) - b)) \, mod \, 98$$

Para aplicarla a lenguaje ensamblador asignaremos los valores a algunos registros:

$a1 = Posición del carácter a descifrar = C(x)

$a2 = Inverso multiplicativo de la clave multiplicativa = a^{-1}

$a3 = Clave aditiva = b

$t1 = 98 = modulo

Implementado en ensamblador MIPS seria de la siguiente manera:

```
        # P(x) = (a^(-1)*(C(x)-b)) mod 98
        sub $t0, $a1, $a3 # C(x) - b
        bltz $t0, Negativo # Si (C(x) - b) < 0 debemos sumarle el modulo (98)

Multiplicacion_decode:
        mul $t0, $t0, $v0 # a^(-1)*(C(x)-b)
        div $t0, $t1 # (a^(-1)*(C(x)-b)) mod 98
        mfhi $a1 # Guardamos la anterior operacion en $a1, para pasarla como parametro a Create_decoded
```

```
Negativo:
         add $t0, $t0, $t1
         j Multiplicacion_decode
```

Note que si el valor de C(x) – b es un numero negativo, debemos sumarle el modulo (98) para continuar con el proceso correctamente, de lo contrario arrojará valores fuera del rango (0-97)

Luego este valor en $a1 se enviará a Create_decoded anteriormente mencionada.

Inverso

La función inverso se hará cargo de calcular el inverso multiplicativo de la clave multiplicativa “a” ingresada por el usuario. Para el calculo de esta, hemos evitado usar el algoritmo de Euclides por su complejidad y se ha decidido buscar uno a uno el inverso multiplicativo de este.

La función recibirá como parámetros:

$t0 = 98 = modulo

$a2 = clave multiplicativa = a (se buscara su inverso multiplicativo)

Luego inicializaremos el valor del inverso multiplicativo:

$t2 = 1 = inverso multiplicativo

El proceso a seguir será el siguiente

Multiplica la clave aditiva por el valor actual del inverso (al inicio será 1), luego, divide el resultado por 98 y obtiene el residuo. Si el residuo es 1, significa que se ha encontrado el inverso multiplicativo, y el programa termina, si no es 1, incrementa el valor del inverso y repite el proceso hasta encontrar el valor adecuado.

```
    li $t2, 1 # Inicializar inverso multiplicativo

Calculo_inverso:
    mul $t4, $t1, $t2 # Multiplicamos la clave aditiva por el inverso
    div $t4, $t0 # calculamos el residuo modulo 98
    mfhi $t4 # almacenamos este residuo en $t4
    beq $t4, 1, Fin # Si el residuo es 1, hemos encontrado el inverso
    addi $t2, $t2, 1 # sino, incrementamos el inverso en 1
    j Calculo_inverso

Fin:
    move $v0, $t2 # Movemos a $v0 el inverso
```

Ciclo de lectura de descrifado

Ya tenemos estructurado el flujo del descifrado del texto luego de ser cifrado, aun no falta explicar la parte de cifrado y como se repetirá el descifrado para finalmente descifrar el archivo por completo.

Para esto necesitaremos que una vez abierto el archivo (que almacenaremos en $a0 por medio de la etiqueta “codificado”), lea carácter por carácter, esto se logra al volver a llamar el buffer que llamaremos codificado_buffer sin cerrar el archivo, lo que permite que avance al siguiente carácter automáticamente, luego, verificamos si el byte leído es igual a cero, si es así, se ha llegado al final del archivo y saltaremos a cerrar el archivo, dentro de estas instrucciones, podemos realizar las operaciones que ya hemos mencionado anteriormente como Find_char y demás, al terminar volveremos al inicio del loop para avanzar al siguiente carácter.

Implementación, llamaremos a esta parte de descifrado “Ejecutar_decode”

```
.data
codificado: .asciiz "C:\\Users\\HP\\Desktop\\Assembler\\Mi desarrollo\\criptogram.txt"
.align 2
codificado_buffer: .space 1024
.align 2
```

```
Ejecutar_decode:
        la $a0, codificado # Direccion del archivo a abrir
        li $v0, 13
        li $a1, 0 # modo lectura
        syscall
        move $s0, $v0 # Movemos el descriptor a $s0

        la $s1, codificado_buffer # Puntero al buffer donde se cargará el contenido

Loop_lectura:
        # Leer un caracter del archivo
        li $v0, 14
        move $a0, $s0 # Pasamos el descriptor a $a0
        addi $a1, $s1, 0 # Puntero al buffer
        li $a2, 1 # Tamaño (leer unicamente un byte)
        syscall

        beq $v0, $zero, Close # Verificar si llegamos al final del archivo

        # Guardar el carácter leido en $a1 para usarlo en find_cahr
        lbu  $a1, 0($s1) # Cargar el byte leído

        # Alfabeto en $a0
        la   $a0, alfabeto

        jal Find_char
```

```
        move $a1, $v0 # Movemos a $a1, el valor que hay en $v0 (la posicion del caracter)
        # $a2 y $a3 deberian tener almacenado el valor de la clave multiplicativa y aditiva correspondientemente

        move $a2, $t4
        move $a3, $t2

        addi $t1, $zero, 98 # inicializamos el modulo en 98

        jal Inverso
        # Retorna $v0 = a^(-1)

        # P(x) = (a^(-1)*(C(x)-b)) mod 98
        sub $t0, $a1, $a3 # C(x) - b
        bltz $t0, Negativo # Si (C(x) - b) < 0 debemos sumarle el modulo (98)

Multiplicacion_decode:
        mul $t0, $t0, $v0 # a^(-1)*(C(x)-b)
        div $t0, $t1 # (a^(-1)*(C(x)-b)) mod 98
        mfhi $a1 # Guardamos la anterior operacion en $a1, para pasarla como parametro a Create_decoded:

        jal Create_decoded # Creamos el archivo decoded.txt pasando como parametro $a1 que será el caracter a imprimir

        j Loop_lectura # repetimos el bucle
```

```
Negativo:
        add $t0, $t0, $t1
        j Multiplicacion_decode

Close:
        # Cerrar el archivo
        li   $v0, 16 # Syscall para cerrar archivo
        move $a0, $s0 # Descriptor del archivo
        syscall

        li $v0, 10
        syscall
```

CIFRADO

Create_criptogram

Esta función es muy similar a Create_decoded, la diferencia es que creará un archivo de nombre distinto, “criptogram” que será el carácter cifrado en lugar de descifrado, lo pasará igualmente al respectivo buffer para imprimirlo en el archivo “criptogram”.

```
.data
output_file: .asciiz "C:\\Users\\HP\\Desktop\\Assembler\\Mi desarrollo\\criptogram.txt"
buffer: .space 1 # Espacio para almacenar el carácter encontrado
```

```
        li $t0, 0 # Inicializamos el contador o indice en 0
Loop:
        lb $t1, 0($a0) #cargar el byte actual en $t1
        beq $t1, $zero, End # Si alfabeto[i] == Null. saltamos a end

        beq $t0, $a1, Encontrado # si i = $a1, saltamos a encontrado
        addi $t0, $t0, 1 # aumentamos en 1 el indice o contador
        addi $a0, $a0, 1 # aumentamos en uno la posicion del vector de caracteres alfabeto
        j Loop
Encontrado:
        lb $t2, 0($a0)  # Cargar el carácter encontrado en $t2. movemos a $t2 el byte resultante al recorrer el vector de caracteres alfab
        sb $t2, buffer  # Guardar el carácter en el buffer. movemos a $t2 el C(x)
        li $v0, 13 # abrir archivo
        la $a0 output_file # direccion archivo
        li $a1, 9 # modo de apertura
        li $a2, 0 # Sin permisos especiales
        syscall
        move $a0, $v0  # Guardar descriptor del archivo en $a0

        li $v0, 15 # escritura
        la $a1, buffer # mensaje a escribir ($t2 = byte encontrado al recorrer el vector de caracteres alfabeto)
        li $a2, 1 # tamaño del mensaje
        syscall

        li $v0, 16 # cerramos el archivo
        syscall
```

Como se observa recibe un número ($a1), buscará a que carácter se asocia este numero y lo llevará al buffer para ser impreso.

Ecuacion C(x)

El valor anteriormente llevado a la función Create_criptogram es el resultado de la ecuación:

$$c(x) = (ax + b)\ mod\ 98$$

Que recordemos, es la posición del carácter a cifrar, la implementación de esta en ensamblador MIPS se ha adaptado de la siguiente manera:

```
# C(x) = (a*x + b) mod 98
addi $t1, $zero, 98 # inicializamos el modulo en 98
mul $t0, $a1, $a2 # a*x
add $t0, $t0, $a3 # a*x + b
div $t0, $t1 # ($a1*$a0 + $a2) mod $t1
mfhi $a1 # Guardamos el resultado de la operacion anterior en $a1
```

$t1 = 98 = modulo

$a1 = Posición del carácter original a cifrar

$a2 = clave multiplicativa

$a3 = clave aditiva

Algo a tomar en cuenta es que como se mencionó anteriormente, los valores que hay en $a2 y $a3 serán las claves multiplicativas y aditivas respectivamente, estas están almacenadas en los registros $t4 y $t2 respectivamente (esto lo veremos más adelante en la sección de Verofy_coprimo e Inicio_aditiva) por lo que será necesario moverlos a los registros $a:

```
# por si acaso move $a1, $t0
move $a2, $t4 # Guardamos la clave multiplicativa en $a2 para pasarla como parametro
move $a3, $t2 # guardamos la clave aditiva en $a3 para pasarla como parametro
```

Inicio_aditiva

Aquí, solicitaremos al usuario que ingrese un numero entre 0 y 97, que usaremos como clave aditiva, para desarrollar este código correctamente, debemos indicarle al usuario que ingrese un numero en el rango indicado y en caso de no hacerlo, solicitárselo nuevamente, hasta que el número ingresado esté dentro del rango.

Primero prepararemos los mensajes que se mostrarán al usuario:

```
.data
clave_user: .asciiz "\nIngresa la clave aditiva (numero entre 0 y 97): "
fallo: .asciiz "\nNumero fuera de rango, intentelo de nuevo\n"
```

Luego imprimimos clave_user, para luego guardar el numero en el registro $v0, con ayuda de los diferentes llamados a sistema (syscalls), a continuación, verificar si el numero es menor a 0 o mayor a 97 y volver a pedir el número si alguna de estas es verdadera.

```
Solicitar_aditiva:
        la $a0, clave_user # mensaje para ingresar la clave aditiva
        li $v0, 4
        syscall
        li $v0, 5 # leer numero
        syscall # El numero es almacenado en $v0
        li $al, 97 # Inicializacion del rango maximo del numero a ingresar
        bgt $v0, $al, Mensaje_error # si num ingresado > 97, saltar a mensaje error
        bgt $zero, $v0, Mensaje_error # si 0 > num_ingreado, saltar a mensaje error
        j Exit
Mensaje_error:
        la $a0, fallo
        li $v0, 4 # mostrar el mensaje de error
        syscall
        j Solicitar_aditiva # volver a pedir el numero
Exit:
        # carga de datos de la pila
        lw $al, 4($sp)
        lw $a0, 0($sp)
        addi $sp, $sp, 8
        jr $ra
```

Verify_coprimo

Aquí solicitamos al usuario un número coprimo con 98 y verificamos que este cumpla con lo indicado.

Como primer paso, preparamos los mensajes a imprimir:

```
.data
num_user: .asciiz "\nIngresa la clave multiplicativa: " # Mensaje para pedir numero al usuario
error: .asciiz "\nEl numero ingresado no es coprimo con 98, intentelo nuevamente.\n"
```

Solicitar al usuario el número:

```
la $a0, num_user # Guardamos en $a0 el mensaje
li $v0, 4 # 4 en $v0 para imprimir una cadena (el mensaje)
syscall # llamado de sistema
li $v0,5 # 5 en $v0 para leer un entero
syscall # llamado de sistema
move $t0, $v0 # copiamos el valor de $v0 (numero ingresado por el usuario) y lo pegamos en $t0 para hacer las operaciones
move $t3, $v0 # Guardamos el valor del numero en $t3 para retornarlo en caso de ser coprimo con 98
```

Para verificar que el numero ingresado es coprimo o no se hará uso del algoritmo de Euclides para calcular el MCD y comprobar que cumpla con lo impuesto.

```
Coprimo:
        addi $s0, $zero, 98 # Guardamos en $s0 el valor 98, ya que el numero ingresado debe ser coprimo a este
        addi $s1, $zero, -1 # Inicializamos en 1 el residuo entre el num ingresado y 98
Find_mcd:
        beq $s0, $zero, End_While # Si $s0 (al inicio 98) es igual a cero, salimos del ciclo while
        div $t0, $s0 # num ingresado % $s0 (al inicio 98)
        mfhi $t1 # guardamos en $t1 el residuo de la anterior operacion
        add $t0, $zero, $s0 # Guardamos en $t0 (num ingresado) el valor que hay en $s0 (al inicio será 98). Este guardara el MCD
        add $s0, $zero, $t1 # Guardamos en $s0 (al inicio es 98) el valor que hay en $t1 (el residuo entre num ingresado y 98)
        j Find_mcd
End_While:
        li $t2, 1 # Guardamos en $t1 el valor de 1, para hacer la comparacion de MCD
        beq $t0, $t2, Es_coprimo #(MCD == 1) si es true, saltamos a es_coprimo, sino continuamos
        la $a0, error # Mensaje advirtiendo no es coprimo
        li $v0, 4
        syscall
        j Verify_coprimo # Volver a solicitar el numero
Es_coprimo:
        move $v0, $t3 # Llevamos el valor de $t3 (numero coprimo con 98) a $v0 para poder retornarlo
```

Finalmente el numero coprimo con 98 ingresado por el usuario, se llevará al registro $v0 para retornarlo.

Los valores de Verify_coprimo (clave multiplicativa) e Inicio_aditiva (clave aditiva), los guardaremos en los registros $t4 y $t2 respectivamente, luego de que estas dos funciones sean ejecutadas:

```
jal Verify_coprimo
move $t4, $v0 # Movemos a $t1, el valor que hay en $v0 (la clave multiplicativa)

jal Inicio_aditiva
move $t2, $v0 # Movemos a $t2, el valor que hay en $v0 (la clave aditiva)
```

Ciclo de lectura de cifrado

Luego de todo esto, se repetirá el mismo esquema de ciclo de lectura de descifrado en este de cifrado, donde guardaremos en $a0 por medio de la etiqueta “input_file” el archivo a abrir y se leerá carácter por carácter en cada iteración gracias al buffer que llamaremos “input_buffer” y a que no cerraremos el archivo hasta que verifiquemos que hemos llegado al final con ayuda del byte leído, podremos realizar las diferentes operación de cifrado dentro de este loop y luego de ejecutarlas volver a repetirlo, a este lo llamaremos “Ejecutar_encode”

```
input_file:     .asciiz "C:\\Users\\HP\\Desktop\\Assembler\\Mi desarrollo\\input.txt"
.align 2
input_buffer:   .space 1024
.align 2
```

```
Ejecutar_encode:
        addi $sp, $sp, -36
        sw $a0, 0($sp)
        sw $a1, 4($sp)
        sw $a2, 8($sp)
        sw $a3, 12($sp)
        sw $s0, 16($sp)
        sw $s1, 20($sp)
        sw $t0, 24($sp)
        sw $t1, 28($sp)
        sw $ra, 32($sp)

        # Abrir archivo en modo lectura
        la   $a0, input_file # Nombre del archivo
        li   $v0, 13 # Syscall para abrir archivo
        li   $a1, 0 # Modo lectura (read-only)
        syscall

        move $s0, $v0 # Guardar descriptor del archivo en $s0

        # Inicialización del puntero de lectura
        la   $s1, input_buffer # Puntero al buffer donde se cargará el contenido

        jal Verify_coprimo
        move $t4, $v0 # Movemos a $t1, el valor que hay en $v0 (la clave multiplicativa)
```

```
        jal Inicio_aditiva
        move $t2, $v0 # Movemos a $t2, el valor que hay en $v0 (la clave aditiva)

Read_loop:
        # Leer un carácter del archivo
        li   $v0, 14 # encode Syscall para leer del archivo
        move $a0, $s0 # Descriptor del archivo
        addi $a1, $s1, 0 # Puntero al buffer (almacenamos en $s1)
        li   $a2, 1 # Leer un byte (carácter)
        syscall

        # Verificar si alcanzamos el final del archivo
        beq  $v0, $zero, Close_file

        # Guardar el carácter leido en $a1 para usarlo en find_char
        lbu  $a1, 0($s1) # Cargar el byte leído

        # alfabeto en $a0
        la   $a0, alfabeto

        # Llamada a la función find_char
        jal  Find_char

        move $a1, $v0 # Movemos a $a1, el valor que hay en $v0 (la posicion del caracter)
```

```
        move $a2, $t4 # Guardamos la clave multiplicativa en $a2 para pasarla como parametro
        move $a3, $t2 # guardamos la clave aditiva en $a3 para pasarla como parametro

        # C(x) = (a*x + b) mod 98
        addi $t1, $zero, 98 # inicializamos el modulo en 98
        mul $t0, $a1, $a2 # a*x
        add $t0, $t0, $a3 # a*x + b
        div $t0, $t1 # ($a1*$a0 + $a2) mod $t1
        mfhi $a1 # Guardamos el resultado de la operacion anterior en $a1

        jal Create_criptogram # Creamos el archivo criptograma.txt

        # Volver al siguiente carácter
        j    Read_loop

Close_file:
        # Cerrar el archivo
        li   $v0, 16 # Syscall para cerrar archivo
        move $a0, $s0 # Descriptor del archivo
        syscall

        lw $ra, 32($sp)
        lw $t1, 28($sp)
        lw $t0, 24($sp)
        lw $s1, 20($sp)
        lw $s0, 16($sp)
```

Ejecucion

Para la ejecución de este programa debe seguir el siguiente flujo:

Ejecutar_encode

- Inicio_aditiva
- Verify_coprimo
- Find_char
- Create_criptogram
- Repetir hasta llegar al final
- Close_file

Ejecutar_decode

- Find_char
- Inverso
- Create_decoded
- Repetir hasta llegar al final
- Close

Conclusiones

En este laboratorio se ha implementado con éxito la técnica de cifrado afín por medio del lenguaje de programación Ensamblador o Assembler en la arquitectura MIPS, consiguiendo satisfactoriamente la encriptación y desencriptación de textos con ayuda de instrucciones y pseudo instrucciones brindadas por el simulador MARS y el lenguaje usado, además de operaciones de aritmética modular y conocimientos adquiridos en cursos como matemáticas discretas y demás materias acerca de programación.

Fue una práctica que necesitó de gran razonamiento y conocimientos adquiridos en clase con ayuda del profesor y aportaciones de los compañeros (RIvera, 2024)

Exposición y simulación

Para observar la explicación y simulación del programa, puede revisarlo aquí. IMPORTANTE: Basta con tener el archivo input.txt y los 7 archivos .asm en la carpeta que hay en escritorio, el archivo criptogram y decoded se generan desde cero y en caso de ya estar creados, únicamente se agrega más texto, por lo que es importante eliminarlos cada que se realice una prueba, el archivo que ejecuta todo el programa es progama.asm.

Bibliografia y referencias

Bibliografía

RIvera, F. A. (2024). *Diapositivas Programación en bajo nivel.* Medellín.