Universidad Nacional de General Sarmiento

TP FINAL ARM

Juego “Ahorcado”

- **ORGANIZACIÓN DEL COMPUTADOR 1**
- **Com 6 (Turno Mañana)**
- **Grupo 16**
- **Alumnos:**

  - Leonardo Mendoza
  - Elias Carrasco
  - Tobias Veron

Universidad Nacional de General Sarmiento

Objetivos:

En el presente tp se buscó replicar el popular juego del ahorcado en lenguaje ensamblador arquitectura ARM para ***Raspberry Pi*** (Arm11- ARMv8-A), aplicando los conocimientos adquiridos en la segunda parte de la materia.

La mecánica del juego es sencilla, se presenta el dibujo de una horca y en su base una palabra de 5 letras de longitud, estas se encuentran ocultas por lo que el jugador debe adivinar en orden de obtener la palabra. Cada jugador tiene 6 vidas al inicio del juego, al cometerse un error se sumará una parte de la persona en la horca y se restará una vida.

En el presente informe se expondrán las **dificultades** durante el desarrollo del programa a su vez que su **funcionamiento** en profundidad.

Directorio del archivo ejecutable: home/occ6g16@Alysa:~/tpfinal/tp

1ra parte: “El ahorcado”

Dificultades

En esta primera parte se encuentra la lógica central del juego, una de las principales dificultades fue la sintaxis de las subrutinas.

En general nuestro enfoque fue siempre el mantener un main con la menor cantidad de operaciones posibles, y durante la mayoría del desarrollo esto significó intentar lograr llamadas a subrutinas autoconclusivas, es decir prácticamente no operando en el main. Si bien en los papeles no parecía un mal camino a tomar este “formato” daba lugar a muchos errores de ejecución. Por ello se decidió cambiar a un modelo de syscalls reducidas y un main más presente en las operaciones lógicas, lo cual derivó en la traducción casi completa del código trabajado, pero también una sustancial mejora en los resultados de ejecución.

Por otro lado, un problema mas especifico llegó durante la creación de las 2 subrutinas centrales de la lógica del juego, siendo estas **“comprobar_letra”** y **“resultado_intento”**.

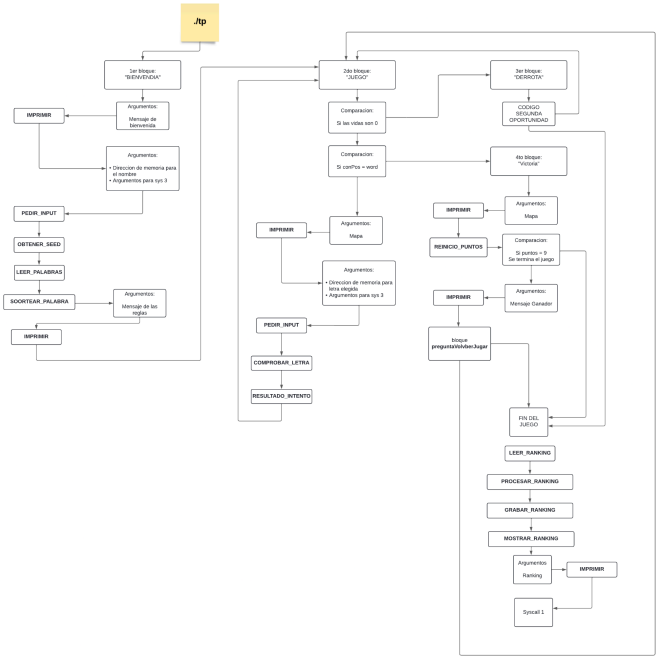
Al desarrollar *comprobar_letra* se debía comprobar consistentemente la existencia de una o más coincidencias de la letra elegida en la palabra. Esto nos dejaba 3 problemáticas a resolver, estas fueron: el **si**, el **no**, y el **¿dónde?.**

El método elegido para saber si existe una coincidencia fue una variable .byte llamada *hayCon*, que a modo de **flag** si se encuentra en 1 o 0 nos indicará si hay o no hay coincidencia respectivamente, y por otro lado, para saber en qué posición de la palabra se encuentra la coincidencia, y al enfrentar varias posibles coincidencias en simultáneo, se optó por implementar una variable **"espejo"** llamada *conPos*, de modo que a medida que se recorre la palabra y se encuentran las coincidencias estas se van reflejando en esta variable. Este método no solo nos permite imprimir de manera correcta las coincidencias, sino que también nos da un método para saber si el usuario adivina la palabra y por tanto es una victoria.

Gracias a eso r*esultado_intento* es una función mucho más manejable, al utilizar un flag *hayCon* podemos descartar el error como primera opción. Si el flag no es 1 entonces no hubo acierto y se debe actualizar como un error, se resta una vida y en base a la cantidad de vidas se imprime una parte de la persona en la horca. De haber coincidencia se recorre la variable espejo *conPos* y en cada coincidencia de la letra elegida se reemplaza y se actualiza la cadena con la misma en la posición correspondiente.

Funcionamiento:

El funcionamiento se explica en el siguiente Workflow:



Universidad Nacional de General Sarmiento

Guia de subrutinas parte 1:

intentando traducir la lógica antes mencionada se realizaron las siguientes subrutinas:

pedir_input: Llama a la syscall 3.

imprimir: Llama a la syscall 4 e imprime.

actualizar_vidas: Actualiza la cadena que muestra la cantidad de vidas.

actualizar_puntos: Actualiza la cadena que muestra la cantidad de vidas.

comprobar_letra: Comprueba si la letra elegida está incluida en la palabra.

resultado_intento: Actualiza el mapa en los aciertos o errores.

reinicio_conPos: Reinicia la variable espejo a su estado original.

reinicio_horca: Reinicia el mapa a su estado original

reinicio_puntos: Incrementa la cantidad de puntos

reinicio_vidas: Reinicia la cantidad de vidas a su estado original.

comparar_conPos: Compara la variable espejo, si esta es igual a la palabra levanta un flag que indica la victoria.

2da parte: “La Puntería”

En la subrutina Derrota, además de volver a imprimir la horca, se imprime el mensaje consultando si quiere hacer **“la puntería”**. Posterior a imprimirse se pasa a confirmar el input y se compara con el ascii correspondiente a “**s”** o “**n”**.

En caso de ser “**s”**, pasamos al bloque *respuestaSi*, en caso de ser **“n”**, nos dirigimos al bloque *jugarOtraVez*, y en caso de no ser ninguno de los dos caracteres, se llama al bloque *preguntarOtraVez*, que imprime un mensaje de error, y devuelve al usuario a re ingresar el input hasta que ingrese el input deseado.

Ya en *respuestaSi*, se imprime un mensaje explicando lo que se tiene que hacer (ingresar las coordenadas x-y para poder cortar la cuerda y ganar). Posteriormente se solicita el input al usuario para la coordenada x, la cual es cargada en r6 con ldrb, y se convierte el primer dígito de ascii a decimal. Comparamos si es válido o no (es decir si está entre 0 y 9) luego cargamos en r8 el 10, se multiplica por el dígito. Posteriormente se lee el segundo dígito, lo transformamos de ascii a decimal para luego sumar a ambos.

Finalmente comparamos con el número deseado (la coordenada x=38) y en caso de ser correcto, pasamos a *primerAcierto* y en caso contrario a *noAcierto.*

En caso de que el input no sea válido (no sea un número entre 0 y 9) pasa a *errorInputX* y le manda un mensaje de error y le pide al usuario que ingrese un número válido.

En primerAcierto repetimos el proceso realizado en *respuestaSi*, y en caso de que el número sea el deseado (coordenada Y = 4) pasamos al bloque *salvado*, que imprime un mensaje de victoria y pregunta si desea volver a jugar. En caso de que sea otro número pasamos a *falloB*, que imprimirá un mensaje de derrota, y preguntará si se quiere volver a jugar.

En *noAcierto*, realizamos lo mismo que *primerAcierto*, solo que en caso de que el número sea el deseado (4) imprime el mensaje de *falloA*, es decir que imprime el mensaje de cuando fallas X. En caso de fallar ambas salta a *falloC*, que imprime el mensaje de que fallaste ambas y también pregunta si quieres volver a jugar

Subrutinas especiales:

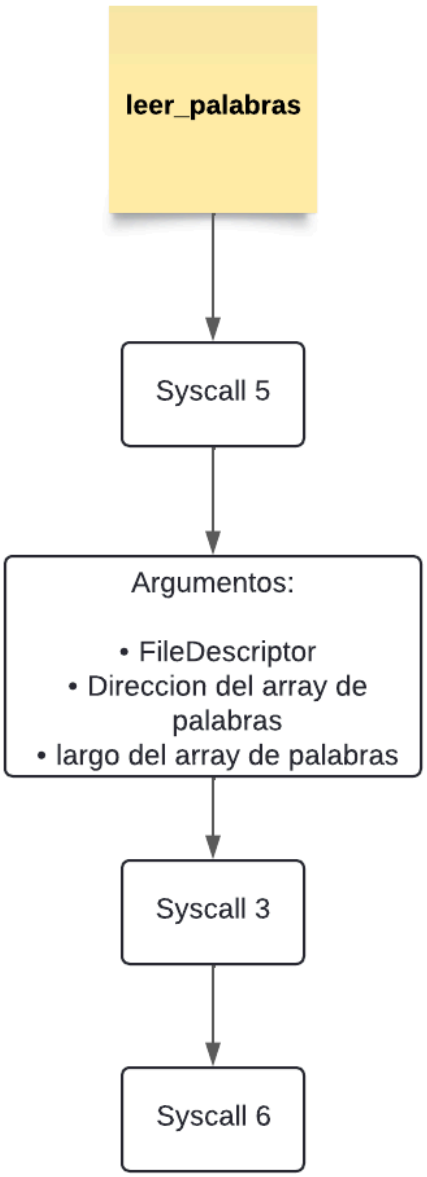
Subrutina leer_palabras:

```
leer_palabras:

        .fnstart
        push {r0, r1, r2, r3, r7, r8, lr}

        mov r7, #5              @ Sys 5 - Buscar en memoria
        ldr r0, =docPalabras    @ nombre del archivo
        mov r1, #2              @ Permisos de apertura
        mov r2, #0              @ Permisos generales, no aplica para un archivo creado
        swi 0                   @ interrupcion

        mov r8, r0              @ Mover el descriptor a r8
        ldr r3, =palabrasLen

        mov r7, #3              @ Sys 3 - Leer
        mov r0, r8              @ Leer el contenido del FileDescriptor
        ldr r1, =palabras               @cargamos en r1 la lista (ahora mismo vacia)
        mov r2, r3              @largo de la palabra
        swi 0

        mov r7, #6
        swi 0

        pop {r0, r1, r2, r3, r7, r8, lr}
        bx lr
        .fnend
```

Funcionalidad:

Esta subrutina se llama antes de que empiece el ciclo juego y sirve para guardar el contenido del archivo ‘palabras’ en un buffer, el cual después es utilizado para escoger las palabras que se desean utilizar para adivinar.

workflow:


leer_palabras
Syscall 5
Argumentos:
• FileDescriptor
• Direccion del array de palabras
• largo del array de palabras
Syscall 3
Syscall 6

Generar índices aleatorios:

Para que funcionen los índices aleatorios se utilizó la fórmula de un Generador lineal congruencial, la fórmula tiene la siguiente forma:

$$X_i = (aX_{i-1} + c) \bmod m$$

Las constantes son:
a (el multiplicador), c (el incremento), m (el módulo),

X_0 (la semilla /valor inicial): Este valor es necesario para seguir creando números pseudo-aleatorios.

Basándonos en esto se creó un código que utiliza los valores ASCII que ingresa el usuario al momento de escribir su nombre, estos valores son transformados en la semilla inicial de la siguiente forma:

subrutina obtener_seed:

```
obtener_seed:                   @Esta funcion crea una seed a partir de los ascii del nombre ingresado

        .fnstart
        push {r0, r1, r2, r3, lr}

        ldr r0, =nombre
        ldr r0, [r0]            @agarra el valor de los ascii que forman el nombre del jugador

        mov r1, #200

        udiv r3,r0,r1           @divido por 200 para obtener un entero mas manejable

        ldr r2, =indice_rand
        str r3, [r2]            @guardo el valor obtenido en la variable indice_rand para generar indices aleatorios

        pop {r0, r1, r2, r3, lr}
        bx lr
        .fnend
```

Pseudocódigo:

Se carga el contenido ASCII de la dirección 'nombre' en r0,

Se establece que r1 valga 200 en decimal

Dividir el contenido ASCII por 200 y lo guardar en el registro r3

Guardar el resultado en la dirección 'indice_rand'

Finaliza la función.

Funcionalidad:

Esta subrutina es llamada luego de que el usuario ingrese su nombre para generar la primera raíz / valor inicial

Luego de que se crea la primera raíz, se pasa a generar un índice random:

Subrutina generar_indice (es una usada en sortear_palabra):

```
generar_indice:

	.fnstart
	push {r0, r1, r2}

	mov r0, #1103		@ multiplicador

	ldr r1, =indice_rand
	ldr r1, [r1]		@ Utilizo la variable indice como raiz

	mov r2, #12341		@ incremento

	mul r0,r1		@ multiplico por la raiz y lo guardo en r0

	add r0, r2		@ sumo el incremento a r0

	@MODULO 50		calculo el modulo 50 del numero que se genero

	mov r1, #50

	udiv r2, r0, r1		@ divide a r0 por 50 y guarda el resultado en r2

	mls r0, r2, r1, r0	@ multiplica r2 por 50 y el resultado resta a r0 y lo guardo en r0

	ldr r1, =indice_rand
	str r0, [r1]		@ guardo el numero como nueva raiz, asi cuando llamen a la funcion devuelta, dara otro numero aleatorio

	pop {r0, r1, r2}
	bx lr
	.fnend
```

Esta subrutina es llamada en ‘sortear_palabra’

Utiliza la raíz generada por la anterior subrutina (obtener_seed)

Pseudocódigo:

Se establece el multiplicador en 1103 (es un número primo)
Utilizamos el valor de la variable índice_rand (raíz inicial)
Se establece el incremento (12341)
Se multiplica el índice_rand por el multiplicador y se guarda en r0
Se suma el incremento al r0

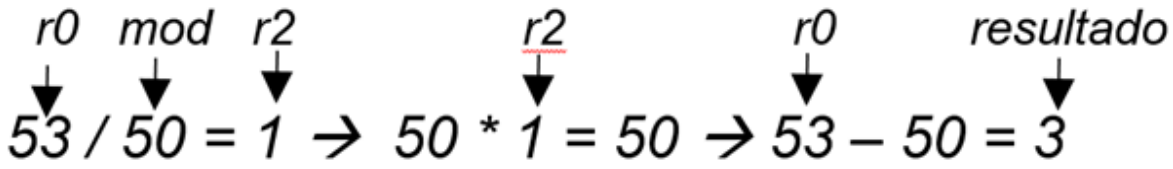
Ahora se debe pasar el valor del módulo al resultado, pero como arm no contiene una instrucción de módulo se que busco la siguiente solución:

Se establece el módulo en 50 (r0; este módulo fue elegido por la cantidad de palabras en el listado)

Se divide a r0 por 50 (se almacena el resultado en r2)

Realizamos producto de r2 por 50 y se resta a r0 el resultado.

Ejemplo:



Guardamos el numero generado en índice_rand (para luego utilizarlo nuevamente para generar otro índice aleatorio)

Funcionalidad:

Utiliza la raíz creada con anterioridad y lo pasa por la fórmula del Generador lineal congruencial.

Subrutina sortear_palabra :

```
sortear_palabra:
        .fnstart
        push {r0, r1, r2, r3, r4, r5, r7, lr}

        bl generar_indice            @ Genero un indice random para seleccionar palabras


        ldr r0, =palabras           @ Cargo la lista de palabras

        ldr r1, =indice_rand         @ Cargo la direccion del indice
        ldr r1, [r1]                @ Guardo el numero del indice en r1

        mov r2, #6              @ 6 es la cantidad de caracteres que se usan para pasar de palabra en palabra (5 caracteres + 1 '\n')
        mul r3, r1, r2          @ Multiplico el indice por 6 asi puedo moverme a la palabra indicada por el indice

        add r0, r0, r3          @ Muevo el puntero r0 a la posición de la palabra seleccionada
        ldr r4, =word

        mov r5, #5
        bl copy_loop

        pop {r0, r1, r2, r3, r4, r5, r7, lr}
        bx lr
        .fnend
```

Pseudocódigo:

Llamamos a la subrutina generar_indice y luego guardamos el número que generó en r1

Almacenamos la dirección de palabras en r0 (contiene todas las palabras)

Se establece que r2 valga #6 (este número se debe a que cada palabra tiene 5 letras + \n)

Se multiplica el índice obtenido aleatoriamente por 6 y lo almacenamos en r3

Se añade el valor de r3 a la dirección de las palabras (r0) para empezar a recorrer la palabra que toco

Llamamos a la dirección de 'word' para almacenar la palabra que se eligió

Se establece el contador en r5 como #5

Llamamos a ***copy_loop*** *para copiar la palabra que salió a la variable 'word'*

Copy_loop:

```
copy_loop:
        .fnstart
        ldrb r6, [r0], #1           @ Cargar byte de palabra y aumentar r0
        strb r6, [r4], #1           @ Almacenar byte en word y aumentar r4
        subs r5, r5, #1             @ Decrementar contador
        bne copy_loop               @ Si no es cero, repetir

        bx lr
        .fnend
```

Funcionalidad:

Se encarga de almacenar byte a byte la palabra en la variable word, que se utilizara en el juego.

Subrutina Leer_ranking:

```
leer_ranking:
        .fnstart
        push {r0, r1, r2, r3, r7, r8, lr}

        mov r7, #5              @ Sys 5 - Buscar en memoria
        ldr r0, =rankfile       @ nombre del archivo
        mov r1, #2              @ Permisos de apertura
        mov r2, #0              @ Permisos generales, no aplica para un archivo creado
        swi 0                   @ interrupcion

        mov r8, r0              @ Mover el descriptor a r8
        ldr r3, =ranklen

        mov r7, #3              @ Sys 3 - Leer
        mov r0, r8              @ Leer el contenido del FileDescriptor
        ldr r1, =rankings       @ cargamos en r1 la lista (ahora mismo vacia)
        mov r2, r3              @ largo de la palabra
        swi 0

        mov r7, #6              @ Syscall para cerrar
        swi 0

        pop {r0, r1, r2, r3, r7, r8, lr}
        bx lr
        .fnend
```

Pseudocódigo:

Se abre el archivo txt rankfile con Syscall 7

Se asegura de que los permisos sean generales (escritura/lectura)

Software interrupt

Luego se guarda la dirección donde se encuentra el nombre del archivo rankfile (descriptor)

Se carga la longitud del archivo ranking (ranklen)

Se realiza el llamado a la Syscall 4 para extraer el texto que está dentro del archivo txt y guardarlo en un buffer

Aseguramos la carga correcta del buffer en el registro r1 (rankings) y el largo del buffer en r2 (ranklen)

Software interrupt

Se realiza el llamado a la Syscall 6 que es para cerrar el archivo

Software interrupt

Finaliza la función

Funcionalidad:

Busca el archivo últimos.txt con el syscall 7 y extrae el contenido del txt, luego lo almacena en un buffer el cual después lo modificamos para generar la lista que queremos. Cierra el archivo.

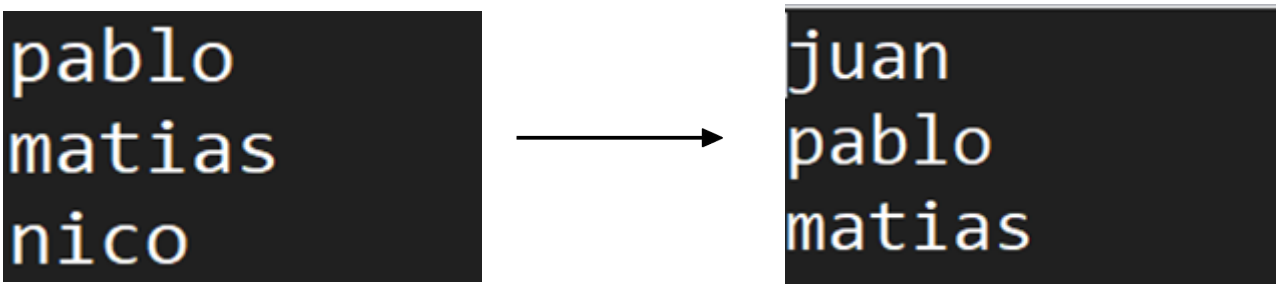
Subrutina Procesar_ranking:

Esta subrutina utiliza otras subrutinas para realizar su función

Funcionalidad:

Esta función lo que hace es, en una lista de 3 nombres mover los nombres un lugar más abajo para dejar espacio para el nuevo nombre que ingresa el usuario. Ejemplo:

Nombre nuevo: Juan




```
procesar_ranking:               @ Mueve el segundo nombre a la tercera posicion de nombres y mueve el primer nombre a la segunda posicion de la fila
        .fnstart
        push {r0, r1, r2, r3, r4, r5, r7, lr}

        bl mover_nombre1        @muevo el segundo nombre de la lista a donde va el tercer nombre de la lista
        bl mover_nombre2        @muevo el primer nombre de la lista a donde ve el segundo nombre de la lista, reemplazandolo

        @ Ingreso el nombre
        @ Preparo los registros para el loop que se encarga de escribir el nombre del actual jugador en la primera fila
        ldr r0, =rankings

        ldr r1, =nombre
        ldr r2, =nombreLen        @ Longitud del nombre
        mov r3 , #0               @ Indice

        b loop_nombre
```

Usa la subrutina mover_nombre1 y 2:

```
mover_nombre1:
    push {r0, r1, r2, r3, r4, r5, r7, lr}

    ldr r0, =rankings

    mov r1, #20                 @ Usar el indice 20 ya que esa seria la posicion del 2do nombre
    mov r4, #40                 @ indica el indice donde debe de parar de reemplazar bytes
    ldr r5, =ranknombre2
    bl cambiar_nombre           @ Mueve el 2ndo nombre a donde se guarda el 3er nombre

    pop {r0, r1, r2, r3, r4, r5, r7, lr}
    bx lr


mover_nombre2:
    push {r0, r1, r2, r3, r4, r5, r7, lr}
    ldr r0, =rankings
    mov r1, #0                  @ Uso el indice 0 que es la posicion del 1er nombre
    mov r4, #20                 @ indice donde deberia terminar de escribir
    bl cambiar_nombre           @ Mueve el 1er nombre a donde se guarda el 2do nombre

    pop {r0, r1, r2, r3, r4, r5, r7, lr}
    bx lr
```

Mover nombre1:

Esta subrutina se encarga de mover el segundo nombre de la lista a donde se encuentra el tercer nombre.

Mover_nombre2:

Esta subrutina se encarga de mover el primer nombre de la lista a donde se encuentra el segundo nombre de la lista.

Estas subrutinas asignan valores a los registros para que funcione la subrutina que intercambia los nombres de lugar, la subrutina es esta:

```
cambiar_nombre:                 @ esta funcion sirve para pasar un nombre de una fila a la siguiente

    // registros inputs:
    // r1 = donde comienza el puntero
    // r4 = donde debe finalizar el recorrido de la palabra

    cmp r1, r4                  @ si el indice esta sobre la posicion r4, debe dejar de escribir
    beq volver

    ldrb r2, [r0,r1]            @ carga el byte indicado por el indice

    add r3 , r1 ,#20            @ le suma 20 al indice para asi indicar donde queda la posicion del 3er nombre

    strb r2, [r0,r3]            @ muevo la letra cargada en r2 a la tercera fila donde se encuentra el indice

    mov r3, #0                  @ reseteo r3 para la proxima iteracion
    add r1, #1                  @ le sumo 1 al indice

    b cambiar_nombre

volver:
    bx lr
```

Lo que hace esta subrutina es asignar valores inputs a los registros r0, r1 y r4 siendo r0 el buffer que contiene el listado de nombres de los últimos nombres

Pseudocódigo:

Se compara el índice que va aumentando con el índice final, (donde tiene que dejar de copiar). Si es igual, tiene que dejar de copiar y retornar al lr

Se carga un byte de la lista de últimos jugadores y utiliza de índice el ingresado por el jugador, este índice va aumentando

Incrementa 20 al índice y se almacena en r3, lo que hace esto es saltar de línea a la próxima posición donde se encuentra el otro nombre, es decir, lo reemplaza byte a byte (Cada nombre ocupa 20 bytes en cada línea donde se guarden los nombres.)

Se Almacena en memoria el byte que se sacó de la línea anterior

Se restablece el registro r3

Se incrementa 1 al índice r1

Reinicia el loop

```
procesar_ranking:               @ Mueve el segundo nombre a la tercera posicion de nombres y mueve el primer nombre a la segunda posicion de la fila
        .fnstart
        push {r0, r1, r2, r3, r4, r5, r7, lr}

        bl mover_nombre1        @muevo el segundo nombre de la lista a donde va el tercer nombre de la lista
        bl mover_nombre2        @muevo el primer nombre de la lista a donde ve el segundo nombre de la lista, reemplazandolo

        @ Ingreso el nombre
        @ Preparo los registros para el loop que se encarga de escribir el nombre del actual jugador en la primera fila
        ldr r0, =rankings

        ldr r1, =nombre
        ldr r2, =nombreLen        @ Longitud del nombre
        mov r3 , #0               @ Indice

        b loop_nombre
```

*Ahora prepara los registros para entrar a la función **loop_nombre**, la cual se encarga de escribir el nombre del nuevo jugador*

R0 = Dirección donde está la lista de nombres

R1 = Dirección del buffer donde se guarda el nombre

R2 = Longitud del nombre

R3 = Índice (se establece en 0)

```
// Subrutinas utilizadas para escribir el nombre del nuevo jugador en el ranking
loop_nombre:
    cmp r3, #19                @ Si el indice es 19 (bytes) entonces ya deberia terminar de copiar
    beq fin_copia

    ldrb r4, [r1, r3]           @ Cargo la letra del nombre indicada por el indice
    cmp r4, #'\n'              @ si el caracter ingresado en el nombre es un salto de linea significa que
    beq seguir_copia

    strb r4, [r0, r3]          @ Guarda la letra en el ranking indicado por el indice

    add r3, #1                 @ Aumento el indice

    b loop_nombre

seguir_copia:
    cmp r3, #19                @ si el indice es 20 termina de copiar
    beq fin_copia

    mov r5, #' '               @ caracter vacio "espacio"

    strb r5, [r0,r3]           @ lo agrego al ranking en su respectivo indice

    add r3, #1

    b seguir_copia

fin_copia:
    pop {r0, r1, r2, r3, r4, r5, r7, lr}
    bx lr
    .fnend
```

Compara si el índice llegó a 19 y si llego a 19 termina de copiar el nombre y vuelve a main

Carga un byte del nombre indicado por el índice r3 (lo carga en r4)

Para evitar bugs *Se compara si el byte elegido es un salto de línea, si es añadir espacios vacíos hasta que deje de copiar. (subrutina seguir_copia)*

Guardar el byte en el listado de nombres, indicado por el índice que está en r3

Continuar el loop

Subrutina grabar_ranking:

```
grabar_ranking:
        .fnstart
        push {r0, r1, r2, r7, lr}

        @ Reabrir el archivo para escribir la lista actualizada
        mov r7, #5                      @ Syscall number for open
        ldr r0, =rankfile
        mov r1, #577                    @ escritura y formatea
        mov r2, #0
        swi 0

        mov r4, r0                      @ Guardar descriptor de archivo

        @ Escribir la lista actualizada en el archivo
        mov r7, #4                      @ Syscall para escribir
        mov r0, r4                      @ Descriptor de archivo
        ldr r1, =rankings
        mov r2, #60                     @ Número de bytes a escribir (3 líneas de 22 bytes + 3 \n)
        swi 0

        @ cerrar el archivo
        mov r7, #6                      @ Syscall para cerrar
        mov r0, r4                      @ Descriptor de archivo
        swi 0

        pop {r0, r1, r2, r7, lr}
        bx lr
        .fnend
```

<u>Pseudocódigo:</u>

Se realiza el llamado a la syscall 5 para abrir el archivo, se asegura que r0 tenga la dirección del archivo a abrir

Se establece r1 en 577, que lo que hace es formatear el archivo (dejarlo vacío) y dejarlo listo para escribir en el

Software interruption

Luego, Se realiza otro llamado a una Syscall, esta vez la 4, para escribir dentro del archivo

Se utiliza:

r0 = dirección del archivo

R1 = dirección de donde esta lo que queremos guardar

R2 = cantidad de caracteres a escribir

Software interruption

Se realiza el llamado a la syscall 6 para cerrar el archivo

Termina la función

Subrutina mostrar_ ranking

```
mostrar_ranking:
        .fnstart
        push {r0, r1, r2, r3, r4, r7, lr}

        bl rank_nombres
        ldr r0, =msgrank          @Tablero del ranking

        mov r1, # 186             @Posicion del cartel ascii
        ldr r2, =ranknombre1      @Nombre que quiero escribir en el cartel
        mov r3, #0                @Indice
        bl loop_ranking

        mov r1, # 274             @Posicion del cartel ascii
        ldr r2, =ranknombre2      @Nombre que quiero escribir en el cartel
        mov r3, #0                @Indice
        bl loop_ranking


        mov r1, # 362
        ldr r2, =ranknombre3
        mov r3, #0
        bl loop_ranking

        pop {r0, r1, r2, r3, r4, r7, lr}
        bx lr
        .fnend
```

Salta a la subrutina ***Rank_nombres***

```
rank_nombres:
       .fnstart
       push {r0, r1, r2, r3, r4, r5, r7, lr}

       ldr r1, =rankings            @Este es el buffer que contiene la lista actualizada, toma los nombres desde aca

       ldr r0, =ranknombre3         @Ingreso el buffer donde quiero guardar el nombre
       mov r2, #40                  @Indice desde donde se empiezan a tomar letras del nombre
       mov r3, #0                   @Indice usado para ir guardando las palabras extraidas en el buffer (ranknombre)
       bl conseguir_nombre

       ldr r0, =ranknombre2         @Ingreso el buffer donde quiero guardar el nombre
       mov r2, #20                  @Indice desde donde se empiezan a tomar letras del nombre
       mov r3, #0                   @Indice usado para ir guardando las palabras extraidas en el buffer (ranknombre)
       bl conseguir_nombre

       ldr r0, =ranknombre1
       mov r2, #0
       mov r3, #0
       bl conseguir_nombre

       pop {r0, r1, r2, r3, r4, r5, r7, lr}
       bx lr
       .fnend
```

Esta subrutina llama a la subrutina conseguir_nombre 3 veces, para guardar en memoria los 3 nombres que aparecen en la lista

Ranknombre3 guarda el 3er nombre de la lista

Ranknombre2 guarda el 2do nombre de la lista

Ranknombre1 guarda el 1er nombre de la lista

```
conseguir_nombre:              @Esta subrutina se encarga de guardar una palabra del listado de nombres en el buffer indicado
       // r0 = buffer donde se va a guardar el nombre deseado
       // r2 = indice donde empieza el nombre en el listado
       // r3 = indice para ir guardando la letra en el buffer (r0) (debe ser iniciado en 0)

       cmp r3, #20                 @si es 20, vuelve donde se llamo
       beq volver

       ldrb r4, [r1,r2]            @Carga un byte del listado de nombres, indicado por el indice r2
       cmp r4, #'\n'               @Si este byte es un salto de linea, significa que es el final de la palabra
       beq volver

       strb r4, [r0, r3]           @Se guarda el byte del listado en el buffer deseado, el indice es r3

       add r3, #1                  @le sumo 1 al indice que carga la letra en el buffer
       add r2, #1                  @le suma 1 al indice que indica donde se toman las letras en el listado

       b conseguir_nombre
```

En r1 se encuentra la dirección del registro donde está el listado de nombres

Usa 3 registros input:

R0 = dirección del buffer donde quiero que se guarde el nombre

R2 = índice donde se encuentra el nombre que quiero guardar

R3 = índice (se empieza 0)

Pseudocódigo:

Compara al índice con 20 que son la cantidad de bytes por nombre, si es igual entonces vuelve a la subrutina

Carga en r4 un byte de la lista indicado por el índice r2

Comparo si es un salto de línea vuelve a la subrutina

Guardo el byte agarrado de la lista en el buffer que quiero, indicado por r3

Aumento r3 por 1

Aumento r2 por 1

Vuelvo al loop

Luego de recorrer los 3 nombres los guarda en memoria para después utilizarlos y mostrarlos en el cartel ASCII de los últimos jugadores

```
mostrar_ranking:
        .fnstart
        push {r0, r1, r2, r3, r4, r7, lr}

        bl rank_nombres
        ldr r0, =msgrank          @Tablero del ranking

        mov r1, # 186             @Posicion del cartel ascii
        ldr r2, =ranknombre1      @Nombre que quiero escribir en el cartel
        mov r3, #0                @Indice
        bl loop_ranking

        mov r1, # 274             @Posicion del cartel ascii
        ldr r2, =ranknombre2      @Nombre que quiero escribir en el cartel
        mov r3, #0                @Indice
        bl loop_ranking


        mov r1, # 362
        ldr r2, =ranknombre3
        mov r3, #0
        bl loop_ranking

        pop {r0, r1, r2, r3, r4, r7, lr}
        bx lr
        .fnend
```

Luego de obtener todos los nombres y guardarlos en memoria

Establezco los registros para luego pasarlos por el loop que los añade al tablero final de últimos jugadores

```
loop_ranking: //Ingresa los nombres deseados en el cartel ascii de ranking
       // r0 = Direccion del cartel ascii
       // r1 = Posicion del cartel ascii
       // r2 = Direccion del nombre que quiero agregar
       // r3 = Contador / indice (siempre arranca en 0)
       cmp r3, #19
       beq volver

       ldrb r4, [r2, r3]        @cargo la palabra de nombre con indice r3 en el registro r4
       cmp r4, #'\n'
       beq volver

       strb r4, [r0,r1]         @la guardo en la posicion del ascii para el primer nombre
       add r3, #1               @sumo 1 al indice
       add r1, #1               @sumo 1 a la posicion del ascii
       b loop_ranking
```

Pseudocódigo:

Compara el índice r3 con 19 si son iguales termina de copiar y vuelve al link register

Carga un byte del nombre indicado por el índice

Compara el byte con un salto de línea para evitar errores gráficos

Guardo el byte en el cartel ASCII, indicado por el índice que está en la posición que quiero reemplazar en el ASCII (r1)

Aumento r3 en 1 (índice)

Aumento r1 en 1 (posición en el cartel ASCII)

Vuelve al loop

Luego de agregar los 3 nombres al tablero ya estaría listo para printear y mostrar los últimos 3 jugadores, también luego de todo este proceso el archivo.txt almacenaría el último jugador y quedaría actualizada