UNIVERSIDAD NACIONAL DE GENERAL SARMIENTO

LICENCIATURA EN SISTEMAS

ORGANIZACIÓN DEL COMPUTADOR

TRABAJO PRÁCTICO - ARM

PROFESORES:

VELCIC, FERNANDO

MALLERET, LAUTARO

ALUMNOS:

GOMEZ, MELINA BELEN

2024

INTRODUCCIÓN

Juego del Ahorcado

Este trabajo práctico consistió en crear el conocido juego del “Ahorcado” en el lenguaje de programación Assembler con arquitectura ARM. El juego del “Ahorcado” consiste en intentar adivinar la palabra oculta, arriesgando una letra por cada intento. Si el jugador adivina la palabra oculta y tiene al menos una vida entonces el jugador gana la partida. Asimismo, por cada error que comete el jugador comenzaran a aparecer en la horca las partes del cuerpo del ahorcado, si todas las partes del cuerpo son dibujadas el jugador perdió la partida, sin embargo, tiene posibilidades aun de ganar el juego cumpliendo con la segunda parte de dicho trabajo, la misma se basa en disparar a la cuerda en las coordenadas exactas que liberarán al ahorcado. En cuanto al repertorio seleccionado para el juego, se decidieron palabras sin relación entre ellas para sumar un punto de dificultad al momento de adivinarlas.

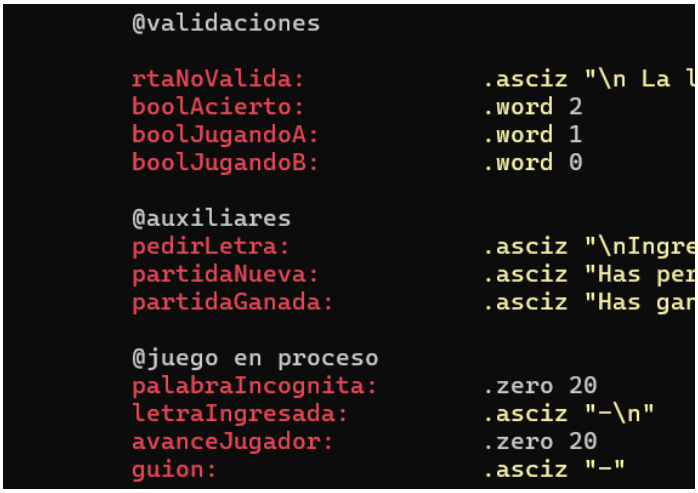
Se brinda un pseudocódigo para comprender la arquitectura del mismo y se implementó main al principio del mismo para una mejor lectura del código.

- Link de acceso a Pseudocódigo

A continuación, se detallará como se llevó a cabo dicho proyecto.

IMPLEMENTACIÓN

Para asegurar un código ordenado, en el mismo se hicieron divisiones con comentarios que ordenaban las etiquetas utilizadas según la función en la que se utilizaban, también se utilizó esta lógica para separar las subrutinas en base a lo que sería la salida visual, la lógica del juego y las funciones auxiliares. Esto nos permite tener un código más ordenado para entenderlo tanto desde uno como autor, como tercero que consulta el código.

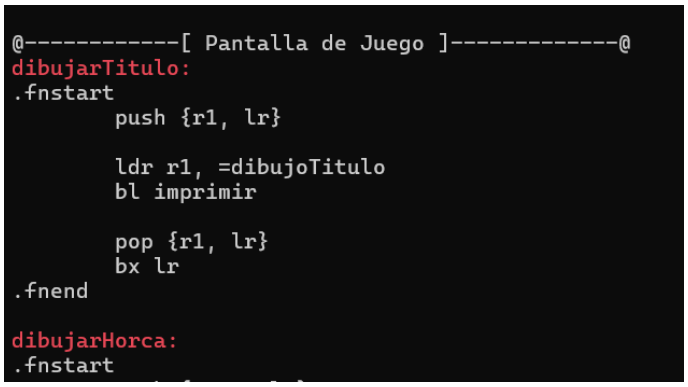


```
        @validaciones

        rtaNoValida:            .asciz "\n La l
        boolAcierto:            .word 2
        boolJugandoA:           .word 1
        boolJugandoB:           .word 0

        @auxiliares
        pedirLetra:             .asciz "\nIngre
        partidaNueva:           .asciz "Has per
        partidaGanada:          .asciz "Has gar

        @juego en proceso
        palabraIncognita:       .zero 20
        letraIngresada:         .asciz "-\n"
        avanceJugador:          .zero 20
        guion:                  .asciz "-"
```



```
@------------[ Pantalla de Juego ]-------------@
dibujarTitulo:
.fnstart
        push {r1, lr}

        ldr r1, =dibujoTitulo
        bl imprimir

        pop {r1, lr}
        bx lr
.fnend

dibujarHorca:
.fnstart
```

Si bien todas las subrutinas existentes son necesarias para la correcta implementación del juego, indagaremos en las subrutinas que implementaron una lógica “nueva”, que fueron aprendidas principalmente en el transcurso de este trabajo:

- **Uso de Random para seleccionar una palabra al azar:**

Se llama a la subrutina random desde seleccionarPalabra como primer paso (la subrutina random se encuentra junto con las demás auxiliares).

En la subrutina random primero se cargan los datos necesarios para informarle al sistema que la interrupción es para generar un random y se informa que queremos que se guarde en la etiqueta semilla.

Luego se carga en un registro la etiqueta semilla y aca es muy importante conocer cuanto espacio ocupamos y cuantas palabras tenemos. Entonces tenemos 4 bytes de números random pero 15 palabras, para simplificar, 1 byte. Entonces en este caso con ayuda de un AND nos quedamos con los últimos 4 bits que serán el número que seleccione la palabra, luego de esto guardamos el valor en semilla.

Luego regresamos a seleccionarPalabra que con ayuda de un contador y recorriendo cada palabra nos dará la palabra n° (el nro que tenga almacenado "semilla.")

```
seleccionarPalabra:
.fnstart
        push {r0,r1,r2,r3,r4,r5,lr}
        bl random
        ldr r0, =semilla
        ldr r0, [r0]
        ldr r1,=palabras
        ldr r3,=palabraIncognita
        mov r4, #0
        mov r5, #0

        cicloSeleccionarPalabra:
```

```
random:
.fnstart
        push {r1,r2,r7,lr}
        mov r7, #384
        ldr r0, =semilla
        mov r1, #4
        mov r2, #0
        swi 0
        ldr r1, =semilla
        ldr r0, [r1]
        and r0, r0, #15
        str r0, [r1]
        pop  {r1,r2,r7,lr}
        bx lr
.fnend
```

Dato: La cantidad de palabras se eligió a propósito, tenía que ser potencia de 2.

- **Conversión de números a carácter de dicho número:**

La conversión de números al char de dicho numero aparece en las subrutinas convertirCoordenadaX y convertirCoordenadaY a las cuales se llega cuando se solicita al jugador que “dispare” de la siguiente manera:

Una vez que el jugador perdió sus vidas, desde la subrutina de derrota se llama a ofrecerDisparo.

En ofrecerDisparo: Primero consulta al jugador si quiere disparar, si este ingresa 'n' continua con los mensajes de derrota y cierre de juego, ingresado 's' se deriva a pedirCoordenadasDisparo.

pedirCoordenadasDisparo: Primero solicita al jugador que ingrese el valor de la coordenada x a la cual desea disparar, una vez ingresada llama a la subrutina ingresarCoordenadaX.

ingresarCoordenadaX: En esta subrutina (Que se encuentra junto con las demás auxiliares) se guardan los 2 caracteres ingresados por el usuario en la etiqueta coorXingresada.

Regresando a pedirCoordenadasDisparo, luego llamará a que se ingrese la coordenada Y con la misma lógica y continua naturalmente a la siguiente subrutina convertirCoordenadaX.

convertirCoordenadaX: Se carga primero el primer byte de coorXingresada (En un caso hipotético suponemos que ingresa el 17, por lo que el primer byte sería el 1 en Hexa el 31). Entonces se le restan 30 y se multiplica por 10 (por ser el valor en la decena): 1 -> 31 en hexa -> (31-30)*10= 10. Luego tomamos el 2do byte y tambien le restamos 30, solo que como en este caso se omite la multiplicacion que sería por 1 ya que es la unidad. Entonces tendríamos: 7 -> 37 en hexa -> (37-30)=7.

Luego se suman ambos valores obteniendo el valor 17 (nótese valor y no número). Luego continúa a convertirCoordenadaY. (Misma dinámica que la mencionada anteriormente la cual no se repetirá para evitar redundancia).

```
ciclo_OfrecerDisparo:
bl ingresarLetra
ldr r1, =letraIngresada
ldrb r1, [r1]
cmp r1, #0x73   //s
beq pedirCoordenadasDisparo
cmp r1, #0x6E   //n
beq salirDeOfrecerDisparo
ldr r1, =rtaNoValida
bl imprimir
bal ciclo_OfrecerDisparo

pedirCoordenadasDisparo:
ldr r1, =pedirCoordenadaX
bl imprimir
bl ingresarCoordenadaX

ldr r1, =pedirCoordenadaY
bl imprimir
bl ingresarCoordenadaY

convertirCoordenadaX:
mov r0, #0
ldr r1, =coorXingresada
ldrb r2, [r1]
sub r2, #0x30
mov r3, #10
mul r0, r2, r3

add r1, #1
ldrb r2, [r1]
sub r2, #0x30
add r0, r2

ldr r1, =coordenadaX
str r0, [r1]
```

Continuamos con comprobarPunteria: que carga ambos valores ya convertidos y los compara con las coordenadas correctas, en caso de coincidir informa un mensaje de victoria. Caso contrario se pierde la partida.

```
comprobarPunteria:
ldr r1, =coordenadaX
ldr r1, [r1]
ldr r2, =coordenadaY
ldr r2, [r2]
cmp r1, #9
bne erroDisparo
cmp r2, #7
bne erroDisparo
ldr r1, =ganoParteB
bl imprimir
bal salirDeOfrecerDisparo

erroDisparo:
ldr r1, =perdioParteB
bl imprimir
```

- **Apertura de archivos, lectura y sobreescritura:**

Se aplica la apertura y lectura de archivos al momento de importar el lemario de palabras y la apertura y lectura/escritura al momento de importar y sobreescribir el ranking, por lo que se procede a explicar este segundo.

Desde el main se llamará a preguntarVerRanking que como su nombre lo indica, consulta al usuario si desea ver el ranking de las ultimas top 3 personas que jugaron. En caso de que la respuesta sea si nos lleva a mostrarRanking, una subrutina que como primer paso importa el ranking desde la subrutina importarRanking (En funciones auxiliares). La misma tiene respectivos comentarios en cada paso:

```
importarRanking:
.fnstart
        push {r0,r1,r2,r7,lr}
        mov r7, #5
        ldr r0, =rankingArchivo
        mov r1, #2
        mov r2, #438
        swi 0

        @Proceso de lectura del archivo

        mov r7, #3
        ldr r1, =ranking_1
        mov r2, #250
        swi 0

        @cerrar el archivo
        mov r0, r6
        mov r7, #6
        swi 0

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

Luego imprimirá por pantalla el ranking que se guardó en la etiqueta ranking_1.

En cuanto al momento en el que se lo escribe, se creó la subrutina agregarJugadorARanking que tendrá diversos bloques dentro de la misma. Comenzamos con un ciclo en el cual se cargará el dato de nombre de jugador y su puntaje (las vidas) en la etiqueta ranking_2, de la misma manera lo hará de ranking_1 donde teníamos los anteriores. Dentro de la subrutina que ingresa el puntaje se lleva un contador que tiene de información cuantos saltos de línea se ingresaron, como comienza en cero, saldremos de la subrutina cuando éste llegue a 2. Luego completamos con ceros el espacio libre disponible en la etiqueta (tambien identificados como fines de cadena) para evitar que queden caracteres que ensucien nuestro código y generen problemas en tiempo de ejecución.

```
agregarJugadorARanking:
.fnstart
        push {r1,r2,r3,lr}
        ldr r1, =nombreJugador
        ldr r3, =ranking_2

        cicloAgregarJugador:
        ldrb r2, [r1]
        cmp r2, #0
        beq agregarPuntaje
        strb r2, [r3]
        add r1, #1
        add r3, #1
        bal cicloAgregarJugador

        agregarPuntaje:
        mov r2, #0x20
        strb r2, [r3]
        add r3, #1
        ldr r1, =vidasJugadorTexto
        ldr r2, [r1]
        str r2, [r3]
        add r3, #2
        ldr r1, =ranking_1
        mov r4, #0

        cicloRankingAnterior:
        ldrb r2, [r1]
        cmp r2, #10
        beq siguienteNombre
        strb r2, [r3]
        add r1, #1
        add r3, #1
        bal cicloRankingAnterior
```

```
        siguienteNombre:
        strb r2, [r3]
        add r1, #1
        add r3, #1
        add r4, #1
        cmp r4, #2
        beq completarConCeros
        bal cicloRankingAnterior

        completarConCeros:
        mov r2, #0
        strb r2, [r3]
        ldr r1, =ranking_2
        bl calcularCaracteres
        mov r4, #250
        sub r4, r2
        mov r2, #0x20

        escribirCero:
        cmp r4, #0
        beq salirActualizarRanking
        strb r2, [r3]
        add r3, #1
        sub r4, #1
        bal escribirCero

        salirActualizarRanking:
        bl exportarRanking
        pop {r1,r2,r3,lr}
        bx lr
.fnend
```

Una vez que modificamos el ranking (la etiqueta) debemos sobreescribir el archivo y lo hacemos en la subrutina exportarRanking donde se busca el archivo por el nombre de la etiqueta y se setea r1 con #1 (escritura) y luego el resto de la parametrización de dicho bloque. Se sobreescribe el archivo y se cierra el mismo, entonces cuando se vuelva a consultar ya estará actualizado.

```
exportarRanking:
.fnstart
        push {r0,r1,r2,r6,r7,lr}
        mov r7, #5
        ldr r0, =rankingArchivo
        mov r1, #1
        mov r2, #438
        swi 0

        mov r6, r0

        mov r0, r6
        mov r7, #4
        ldr r1, =ranking_2
        mov r2, #250
        sub r2, #1
        swi 0

        @cerrar el archivo
        mov r0, r6
        mov r7, #6
        swi 0
        pop {r0,r1,r2,r6,r7,lr}
        bx lr
.fnend
```

CONCLUSION

En el código implementado hay muchas subrutinas funcionales y etiquetas cuyos datos permiten la correcta implementación del juego. Se mencionaron las principales que resultaron un desafío en este camino de aprendizaje. Siendo un lenguaje de bajo nivel fue clave nombrar las subrutinas específicamente por su función para poder interpretar de una mejor manera lo que hacían. Se utilizaron subrutinas de bloques de código para evitar acoplamiento, tales como imprimir y calcularCaracteres, lo cual también ayudó a tener un código prolijo. También se implementó un **catch** de error si el usuario responde cualquier letra en una respuesta del tipo (s/n), en ese caso le pide que ingrese una respuesta válida.

Este trabajo me permitió comprender más sobre los usos de espacio en memoria y pensar desde otras perspectivas al tener que realizar métodos de manera “manual”. Si bien fue difícil de entender al principio por ser la primera vez trabajando con lenguaje Assembler, es satisfactorio ver ahora un bloque de código y poder interpretarlo siendo que antes se veía muy difícil.

```
main:
    dibujarTitulo()        // Muestra el título del juego
    obtenerNombreDeJugador() // Pide al jugador que ingrese su nombre
    preguntarVerRanking()  // Pregunta si el jugador quiere ver el
ranking
    seleccionarPalabra()   // Selecciona una palabra aleatoria para el
juego
    reiniciarAvance()      // Reinicia el progreso del jugador
    reiniciarJugador()     // Pregunta si el mismo jugador sigue
jugando y muestra el ranking si es necesario
    dibujarHorca()         // Dibuja la horca inicial
    mostrarJugadorActual() // Muestra el nombre del jugador actual
    mostrarVidas()         // Muestra las vidas restantes del jugador

    while (jugando):
        pedirLetra()       // Pide al jugador que ingrese una letra
        analizarIngreso()  // Analiza la letra ingresada
        comprobarVictoria() // Comprueba si el jugador ha ganado
        comprobarDerrota()  // Comprueba si el jugador ha perdido
        dibujarHorca()      // Dibuja la horca actualizada
        mostrarVidas()      // Muestra las vidas restantes
actualizadas

Funciones:
dibujarTitulo():
    imprimir(dibujoTitulo)

dibujarHorca():
    imprimir(separador)
    obtener vidasJugador
    seleccionar y mostrar dibujo de la horca basado en las vidas
restantes

mostrarJugadorActual():
```

```
    imprimir(txt_jugandoAhora)
    imprimir(nombreJugador)

mostrarVidas():
    imprimir(txt_vidas)
    imprimir(vidasJugadorTexto)

mostrarRanking():
    importarRanking()
    imprimir(ranking_txt)
    imprimir(ranking_1)
    imprimir(ranking_txt2)

preguntarVerRanking():
    imprimir(txt_verRanking)
    while (respuesta no válida):
        ingresarLetra()
        if letraIngresada == 's':
            mostrarRanking()
        else if letraIngresada == 'n':
            salir

obtenerNombreDeJugador():
    imprimir(txt_ingreseNombre)
    leerNombre(nombreJugador)
    limpiar entrada
    imprimir(txt_saludo1)
    imprimir(nombreJugador)
    imprimir(txt_saludo2)

seleccionarPalabra():
    random()
```

```
    seleccionar palabra aleatoria de la lista de palabras y copiar a
palabraIncognita


reiniciarAvance():
    for cada letra en palabraIncognita:
        si no es el fin de línea:
            reemplazar con guion en avanceJugador


reiniciarJugador():
    imprimir(mismoJugador)
    while (respuesta no válida):
        ingresarLetra()
        if letraIngresada == 's':
            salir
        else if letraIngresada == 'n':
            obtenerNombreDeJugador()


comprobarVictoria():
    if avanceJugador no contiene guiones:
        agregarJugadorARanking()
        imprimir(avanceJugador)
        imprimir(partidaGanada)
        while (respuesta no válida):
            ingresarLetra()
            if letraIngresada == 's':
                reiniciar juego
            else if letraIngresada == 'n':
                terminar juego


comprobarDerrota():
    if vidasJugador == 0:
        agregarJugadorARanking()
```

```
        dibujarHorca()
        mostrarVidas()
        ofrecerDisparo()
        imprimir(partidaNueva)
        while (respuesta no válida):
            ingresarLetra()
            if letraIngresada == 's':
                reiniciar juego
            else if letraIngresada == 'n':
                terminar juego

ofrecerDisparo():
    imprimir(ofrecerDisparo_txt)
    while (respuesta no válida):
        ingresarLetra()
        if letraIngresada == 's':
            pedirCoordenadasDisparo()
            if coordenadaX == 9 y coordenadaY == 7:
                imprimir(ganoParteB)
            else:
                imprimir(perdioParteB)

analizarIngreso():
    if letraIngresada está en palabraIncognita:
        actualizar avanceJugador
        imprimir(letra correcta)
    else:
        restar vida a vidasJugador
        actualizar vidasJugadorTexto
        imprimir(letra incorrecta)
```