Universidad de San Carlos de Guatemala
Arquitectura de computadores y ensambladores 1
Escuela de Sistemas
Ing. Otto Rene Escobar Leiva
Aux Frederick Jonathan Faugier Pinto

Manual de Usuario:

Proyecto 2

Nombre: **Carnet**:
John Henry López Mijangos 201710392

Guatemala, 30 de junio del 2022

Introducción:

El proyecto tuvo como objetivo la creación de un juego llamado esquivar vehículos, a través de la memoria de gráficos que en assembler se puede manejar, teniéndose además del juego un apartado para registro de usuarios, uno para login y un menu dependiendo de los tres tipos de usuarios que pueden existir (usuario normal, usuario admin, administrador general) teniendo entre las opciones que dichos menús proveen la posibilidad de manipular los datos que el juego registraba a través de un score en un archivo externo, pudiéndose consultar el top 10 de usuarios con mejor score y su partida, así como un apartado para visualizar métodos de ordenamiento, tomando como datos a analizar ya sea el score o el tiempo requerido para finalizar cada juego, además de proveer al administrador general la posibilidad de quitar bloqueos de usuarios asi como promover y quitar administrador. Se llegó a la conclusión que la memoria de gráficos facilita enormemente la necesidad de manipular objetos movibles en forma de píxeles, así como la facilidad que esta provee para guardar su estado en un archivo externo y realizar diversos gráficos con esta.

Objetivos

Objetivo General

- Aplicar todos los conocimientos adquiridos en el manejo del lenguaje ensamblador, que permitan al estudiante utilizar su intelecto y creatividad para diseñar soluciones óptimas y simples de problemas complejos

Objetivos Específicos

- Aprender a simplificar operaciones complejas a simples operadores del lenguaje ensamblador.
- Entender y manipular el uso de la memoria en los programas informáticos.
- Conocer el manejo de las interrupciones.
- Poner en práctica los conocimientos de operaciones aritméticas básicas a bajo nivel.
- Comprender el uso de la memoria de video en los computadores.
- Manejar el modo gráfico y el modo video en lenguaje ensamblador..

Código:

main: Codigo principal con el cual se llamará al resto de macros y procedimientos:

```
include macro.asm
.MODEL small
.STACK
.RADIX 16
.DATA
;APARTADO PARA LA DECLARACION DE VARIABLES Y LISTAS
mVariables
.CODE
;APARTADO PARA EL CODIGO
start:
    main proc
        call pFlujoProyecto2
    main endp
END start
```

pFlujoProyecto2: Procedimiento con el cual se procedera a llamar al resto de macros y procedimientos.

```
pFlujoProyecto2 proc
    call pAjustarMemoria
```

pMenuPrincipal: Menú principal con el cual se accedera ya se a login, a registrar o a salir del programa, cabe aclarar que las teclas Fn's en la laptop donde fue trabajado el proyecto para ser activadas es necesario apretar una tecla especial llamada Fn, por lo cual se llama dos veces a la interrupción ah 01-int 21, una para capturar el fn y otra para capturar el valor de la tecla auxiliar.

```
pMenuPrincipal proc
    ciclomenu:
        mov opcion,0
        mMostrarString Menu
```

LOGIN:

pLogin: procedimiento que presentara las opciones necesarias para logearse, teniendo como principal objetivo el capturar un usuario y contraseña, verificando en primer lugar si el usuario existe, para

posteriormente revisar si la contraseña es correcta o no y revisar si el usuario ha sido bloqueado y por lo tanto aun con la contraseña correcta no le sea posible ingresar (cuando se equivoca 3 veces un usuario con su contraseña es bloqueado).

```
pLogin proc
    call pResetFlagsE
    mOpenFile2Write usersb ;abre el archivo de users
    cicloLogin:
    call pLimpiarConsola
```

mUserExiste: macro para identificar si el usuario existe, y además posicionarlo en la fila del documento leído correcta para posteriormente tomar los datos de esta.

```
mUserExiste macro Username
    local Existe,Noexiste,salir,cicloexiste
    ;SE VERIFICA SI NO ES EL ADMIN
    mReadFile eleActual ;TOMA EL PRIMER VALOR DEL ARCHIVO
    mEncontrarId Username;lo primero en el documento de usuarios es el
admin, que siempre estara aca
    cmp idEncontrado,1 ; se encontro usuario?
    je Existe ;si se encontro se procede a decir que si existe el
```

REGISTRAR:

pRegistrar: Procedimiento con el objetivo principal de capturar un usuario y contraseña para luego verificar que cumpla con las características requeridas de estos, para hacer validas un posible registro.

Características para hacer válido un registro:

1. Se solicitará el nombre de usuario y se harán lan siguientes validaciones:

1. No puede empezar por número.
2. La longitud es libre
3. El nombre de usuario no debe existir.

2. Se solicitará el password para el usuario y se harán las siguientes validaciones.

1. Longitud máxima y unica es de 4 numeros.

```
pRegistrar proc
    mov eerror,0
    mLimpiar UsuarioRegis,25,24
    mLimpiar PasswordRegis,25,24
    call pLimpiarConsola
```

mUserInicial: verifica si la primera letra del usuario a ingresar es un numero o no.

mUserExisteR: verifica si el usuario existe o no.

mEncontrarId: busca si en una posición actual del cursor en un archivo leído existe una cadena igual a una cadena en específico.

mSizePassword: Verifica que la contraseña tenga la longitud correcta

```
mSizePassword macro
    local ciclosize,comparaciones,sentenciagood,salir,sentenciabad
    push si
    mov contadoraux,0 ; se inicializa la variable que contendra el
tamaño del password
    mov si, 0
    ciclosize:
        cmp si,25t ; si llego a 25 (maximo tamaño para una password )
pasa a proceder a verificar el tamaño
        je comparaciones ;pasa a comparar con los margenes
        mComparar PasswordRegis[si],"$" ;llego hasta $, significa que
```

MACROS Versátiles:

mEnRango: Verifica que un elemento se encuentre entre un rango especificado.

```
mEnRango macro dato,limif, limsup
    local enElrango,noEnelrango,salir
    ;ja >,jb <,  jbe<=
    mComparar dato,limif
    jb noEnelrango ; si es menor al limite inferior no esa en el rango
    mComparar dato,limsup
    jbe enElrango ; si es menor o igual al limite superior esta en el
```

mMostrarString: Imprime un string con un “$” al final.

String2Num: Convierte un string a decimal.

Num2String: Convierte un numero a un string legible.

mCapturarString: Captura una cadena de un documento que se esta leyendo actualmente, en un espacio de este en especifico.

```
mCapturarString macro variableAlmacenadora
    local salir,capturarLetras,deletCaracter
    push ax
    push si
    mov si,0
    capturarLetras:
        mov ah,01h
        int 21h
        cmp al, 0dh ;es igual a enter?
        Je salir ; una vez dado enter y capturado todo el nombre, pasar
```

mComparaStrings: Compara dos cadenas e indica si ambas son iguales o no a través de una variable llamada cadIguales como resultado (1 si son iguales, 0 no son iguales).

```
mCompararStrings macro var1, var2
    local salir,Iguales,noIguales,comparar,pfvar1,pfvar2
    push si
```

```
    mov cadIguales,0
    mov si,0
    comparar:
        mComparar var1[si],var2[si]
        je Pfvar1
        jne noIguales
    pfvar1:
        mComparar var1[si],"$" ;cadena llego al final?
        je pfvar2 ;tambien llego al final en la cadena 2?
        inc si
        jne comparar
    pfvar2:
        mComparar var2[si],"$"
        je Iguales ;si llego al final al mismo tiempo que var 1, son
iguales
        jne noIguales ;no son iguales
    Iguales:
        mov cadIguales,1
        jmp salir
    noIguales:
        mov cadIguales,0
        jmp salir
    salir:
    pop si
endm
```

mComparar: Permite comparar variables entre variables y brinda una mayor seguridad al comparar otros valores.

mMoverVariablesDw: Mueve variables de tamaño word.

mCrearFile: Macro para crear archivos externos.

mWriteToFile: Permite escribir un string en un archivo actualmente abierto.

```
mWriteToFile macro palabra
    push ax
    push bx
```

```
    push cx
    push dx
    mov bx, handler
    mov cx, LENGTHOF palabra
    mov dx, offset palabra
    mov ah,40
    int 21
    pop dx
    pop cx
    pop bx
    pop ax
endm
```

mReadFile: Permite leer un carácter de un archivo leído actualmente, con cada llamada de esta macro aumenta la posición del cursor de lo actualmente leido en uno.

```
mReadFile macro varAlmacenadora
    push ax
    push bx
    push cx
    push dx
    mov bx, handler
    mov cx, 1
    lea dx, varAlmacenadora ; esto seria igual a: mov dx, offset
lectura, "EN LA POSICION DE LECTURA GRABAR LO LEIDO"
    mov ah, 3F
    int 21
    mov posLectura, ax
    pop dx
    pop cx
    pop bx
    pop ax
endm
```

mOpenFile2Write: Permite abrir un archivo para que pueda ser leído y escrito.

```
mOpenFile2Write macro fileName
```

```
    local errorOpen,Opencorrecto,salidaOpen
    push ax
    push dx
    mov estadocarga,0
    mov al,2
    lea dx, fileName
    mov ah,3Dh
    int 21
    jc errorOpen
    mov handler, ax
    jmp Opencorrecto
    errorOpen:
        mMostrarString carbad
        mov estadocarga,0
        jmp salidaOpen
    Opencorrecto:
        ;mMostrarString cargood
        mov estadocarga,1
        jmp salidaOpen
    salidaOpen:
    pop dx
    pop ax
endm
```

mDrawPixel: macro para pintar un pixel en la ventana, con el modo vídeo ya activado.

```
mDrawPixel macro line,column,color
    push ax
    push bx
    push dx
    push si
    xor ax,ax
    xor bx,bx
    xor dx,dx
    xor si,si


    ;formula para pintar un pixel de la matriz video = ((linea-1) *
320) + (columna-1)
```

```
    mov ax,line
    dec ax
    mov bx, 320t
    mul bx
    ;en ax ya tengo el resultado del primer parentesis
    add ax, column
    dec ax


    mov si, ax
    mov bl,color
    mov es:[si],bl


    pop si
    pop dx
    pop bx
    pop ax
endm
```

mDrawRectangulo: Macro para dibujar un rectángulo en el modo video.

```
mDrawRectangulo macro x,y,ancho,alto,color
    local lineasup,barraslat,lineainf
    push cx
    push bx
    xor cx,cx
    xor bx,bx
    mov bx,y ;auxiliar que tendra almacenada la variable y
    mov cordx,x
    mov cordy,y


    mov cx,ancho
    lineasup: ;se grafica la linea superior, imprimiendo y aumentando
las columnas para generar una linea
        mDrawPixel cordx,cordy,color
        inc cordy
        loop lineasup
    mov cordy,bx ; se regresa cordy a su valor original
    inc cordx ;se pasa a la siguiente fila
```

```
    mov cx,alto ; se hara el siguiente procedimiento hasta que se
cumpla el alto establecido
    barraslat: ;se grafican las barras laterales
        mDrawPixel cordx,cordy,color
        mSumarDw cordy,ancho
        dec cordy
        mDrawPixel cordx,cordy,color
        mov cordy,bx ;una vez hecho las dos impresiones siempre volver
al valor original
        inc cordx
        loop barraslat
    mov cx,ancho
    lineainf:
        mDrawPixel cordx,cordy,color
        inc cordy
        loop lineainf
    mov cordx,0
    mov cordy,0
    pop bx
    pop cx
endm
```

mImprimirLetreros: Macro para imprimir strings en modo video.

```
mImprimirLetreros macro letrero,fila,columna,color
    push ax
    push bx
    push cx
    push dx
    push bp
    mov al,1   ;MODO DE IMPRESION CON COLOR (1), SIN COLO(0)
    mov bh,0   ;PAGINA
    mov bl,color ;COLOR  (PALETA VGA 1t-255t)
    mov cx,LENGTHOF letrero ;tamaño del letrero
    mov dl,columna ;columna
    mov dh,fila ;fila
    call pDataS_ES ;se puede realiar esto o el procedimiento de abajo
siempre y cuando ds tenga el valor de @data
```

mDrawBarra: Dibuja una barra para ser usada en los ordenamientos y aspectos visuales.

```
mDrawBarra macro x,y,alto,ancho,color
    local cicloAncho,cicloAlto,no0x,no0y
    push ax
    push dx
    push cx


    movVariablesDw cordx,x
    movVariablesDw cordy,y
        cmp cordx,0
        jne no0x
        mov cordx,1
        no0x:
        cmp cordy,0
        jne no0y
        mov cordy,1
        no0y:
```

pDelay30t: Procedimiento de 30 segundos que imprime el tiempo que lleva cada segundo.

```
pDelay30 proc
    push ax
    push dx
    mov valort1,0
    mov contadort,0
    ;SE TOMA EL VALOR DE T1
    mov ah,2Ch
    int 21h
    mov valort1,dh ;VALOR 1 TOMA UN TIEMPO INICIAL
    ciclodelay:
        mov ah,2Ch
        int 21h
        mComparar valort1,dh ;EL CICLO SE REPETIRA HASTA QUE SEAN
DISTINTOS
        jne segundo ;ES DISTINTO POR LO CUAL YA CAMBIO DE SEGUNDO
```

```
        jmp ciclodelay
        segundo:
            mLimpiar StringNumT,4,24 ;SE LIMPIA EL STRING QUE
ALMACENARA EL SEGUNDO
            Num2String contadort,StringNumT ;SE PASA EL CONTADOR ACTUAL
A STRING
            mMostrarString StringNumT ;SE IMPRIME EL STRING DEL
CONTADOR
            cmp contadort,30t ;CONTADOR ES IGUAL A 30?
            je salir ;SI, SALIR
            MovVariables valort1,dh ;NO, ENTONCES VALORT1=auxt (que
contiene el valor2 sin el efecto de mod)
            inc contadort ; SE LE SUMA UNO AL CONTADOR
            jmp ciclodelay
    salir:
        pop dx
        pop ax
    ret
pDelay30 endp
```

JUEGO

pMovimientoGame: Es el corazón del juego, le permite al juego moverse a 100 fps.

```
pMovimientoGame proc
    mov auxfpsT,0
    reset:
        call pConfigIni
    fps: ;ciclo que provoca un movimiento cada centisegundo
        mov ah,2Ch
        int 21
        cmp dl, auxfpsT
        je fps
    mov auxfpsT, dl
    call pDrawCleansCorazones
    call pDrawCorazones
    cmp nivelGame,4 ; si se finalizo el 3 nivel, nivelgame llegara a 4
indicando que finalizo el juego
```

pDrawcarro: Procedimiento para dibujar el carro principal.

```
pDrawNave proc
    push cx
    push ax
    push dx
    mov ax,cNave_x
    mov dx, cNave_y
    ;CAÑON PRINCIPAL
    mDrawPixel cNave_x,cNave_y,39t
    inc cNave_x
    mDrawPixel cNave_x,cNave_y,15t
    ;CUERPO
    inc cNave_x
    dec cNave_y
    mDrawFila cNave_x,cNave_y,15t,3t
    inc cNave_x
```

pMovEnemys: Variable que permite mover los enemigos de uno en uno para realizar el ataque kamikaze a la nave principal.

```
pMovEnemys proc
    push cx
    cmp estEnem,3
    je filaene3
    cmp estEnem,2
    je filaene2
    cmp estEnem,1
    je filaene1
    jmp salir ;SE MUEVE EL ESTADO PARA PASAR AL NIVEL 2
    filaene3:
        mov cx,nivelGame
        movi3:
            call pDestEnemA ;el enemigo fue destruido con anterioridad?
            cmp DestEnemA, 1 ;si entonces saltar a fin de movimiento
            je finMov3
            movVariablesDw borrXenemy, ce_x
            movVariablesDw borrYenemy, ce_y
            mDrawEborrado borrXenemy,borrYenemy
            call pColision
```

pTimeGame: Cronómetro del juego.

```
pTimeGame proc
    inc cengameN
    cmp cengameN,100t ;cuando llegue a 100 el contador de centisegundos volvera a 0 y se le sumara 1 a los segundos caso contrario solo sumara uno y se saldra
    jne salir
```

pPauseGame: Procedimiento para generar una pausa en medio del juego.

```
pPauseGame proc
    call pGuardarMatrizVideo ; guardar el estado de la matriz de video para posteriormente cargarla sin los letreros
    mov exitGame, 0
    mImprimirLetreros letPause,5t,25t,15t
    mImprimirLetreros letRen,12t,20t,15t
    mImprimirLetreros letExit,15t,20t,15t
    ciclo:
        mov ah, 00 ;Espera a que se presione una tecla y la lee
        int 16h
        cmp al, 27t ;escape
        je exitG
        cmp al, " " ;espacio
        je salir
        jmp ciclo ;estara en un ciclo si no es o espacio o escape
    exitG:
        mov exitGame,1
    salir:
        call pCargarMatrizVideo ;cargar la matriz de video guardada luego de los letreros
    ret
pPauseGame endp
```

ORDENAMIENTOS:

pMoveOrdenamiento: Corazón de los ordenamientos, permite de modo default mover los gráficos a 100 fps con una delay escogido bajo.

```
pMoveOrdenamiento proc
    push ax
```

```
    push dx
    mov auxfpsT,0
    reset:
        call pConfigInicOrd
        call pDrawBarras
        call pOrdMando
    fps: ;ciclo que provoca un movimiento cada centisegundo
        mov ah,2Ch
        int 21
        cmp dl, auxfpsT
        je fps
    mov auxfpsT, dl
    call pTimeOrd
    cmp EstOrd,0t
    je sinAccion
        mDrawBarra 17t,0t,170t,8t,0t;borrar flechas de pasos anteriores
        call pBubbleSort
        cmp EstOrd,0t
        je exit
    sinAccion:
    jmp fps
    exit:
        mImprimirLetreros msgPressEnd,24t,7t,15t
        ciclo: ;SALE HASTA QUE SE PRESIONE "FIN"
         mov ah, 00 ;Espera a que se presione una tecla y la lee
        int 16h
        cmp al,"O"
        je exit2
        cmp al,"1"
        je exit2
        jmp ciclo
        exit2:
        pop dx
        pop ax
    ret
pMoveOrdenamiento endp
```

pRDatosOrdPuntos: Permite guardar los 20 primeros datos del archivo score en un array asi como la posición de estos.

pDrawBarras: dibuja el estado actual de los array con los datos cargados en un diagrama de barras.

pShowtop10: Permite visualizar el top 10 de mejores tiempos.

pShowMyTop10: Permite visualizar los 10 mejores scores de un jugador.

Equipo Requerido:

- 2 gb ram
- procesador de 2 nucleos 2.3 ghz
- Dosbox 0.74-3
- Visual Code

CONCLUSIONES

- A la hora de manipular archivos externos es necesario limpiar los registros utilizados, si en algún momento llegamos a realizar una operación como la división luego de abrir dichos archivos con el programa.
- La mala utilización de registros puede dar errores como el paro del funcionamiento de un programa.
- Los lenguajes de alto nivel facilitan mucho todo lo concerniente a la programación permitiendo que nos despreocupemos del uso correcto de los registros.
- Las macros son una funcionalidad que permiten en gran medida el ahorro de líneas de código siendo estas reutilizables en otros proyectos.
- No se pueden mover dos variables al mismo tiempo, debe de existir un registro intercesor entre los dos.
- La memoria de gráficos permite manipular de forma mas sencilla los pixeles a colores para que el programa realice distintas reacciones dependiendo de estos, además de permitirse guardar esta en archivos externos.