

MANUAL TÉCNICO

201602782 – Sergio Fernando Oztzy Gonzalez
CUARTA PRACTICA DE LABORATORIO
ARQUITECTURA DE COMPUTADORAS Y ENSAMBLADORES 1

TABLA DE CONTENIDO

1	Objetivos	2
1.1	Generales.....	2
1.2	Específicos.....	2
2	Alcance	2
3	Panorama general de la aplicación	3
4	Requisitos técnicos	3
5	Funcionalidad de la aplicación	5
5.1	Función toLower.....	5
5.2	Funcion compareStr.....	6
5.3	Flujo de menu principal	7
5.4	Flujo Play	8
5.4.1	Flujo verifyCatch.....	10
5.4.2	Flujo VerifySuicide.....	11
5.5	Flujo PASS.....	12
5.6	Flujo Load	13
5.7	Flujo Save	14
5.8	Flujo Show	15

1 OBJETIVOS

1.1 GENERALES

1. Dar una explicación breve y concisa de aspectos técnicos del programa

1.2 ESPECÍFICOS

1. Explicar requerimientos mínimos de instalación del programa. Tales como localizaciones de archivos.
2. Utilizar diagramas para explicar procedimientos y rutinas seleccionadas del programa.
3. Ofrecer una explicación sencilla sobre el código de ensamblador en el que está escrito el programa

2 ALCANCE

Este manual técnico está dirigido a personas que posean un conocimiento sobre ensamblador, DOS y Microsoft® Macro Assembler (MASM); es necesario una noción básica sobre los registros y segmentos del procesador Intel® 80186 (16bytes).

El programa está escrito en MASM. Se utiliza DOSBox como entorno de pruebas. El bus de datos, el tamaño y uso de los registros/segmentos y la sintaxis utilizada está basada y limitada por la arquitectura del procesador Intel® 80186 (16bytes).

Se utiliza la versión 6.1 de MASM. No se utiliza ninguna librería. Todo el código está escrito con instrucciones de ensamblador y estructuras de control de alto nivel que el compilador de MASM ofrece¹.

¹ Según manual del programador: <http://people.sju.edu/~ggrevera/arch/references/MASM61PROGUIDE.pdf>

3 PANORAMA GENERAL DE LA APLICACIÓN

La aplicación consiste en recrear el juego *go*. El *go* es un juego de tablero de estrategia para dos personas.

El objetivo del juego es controlar una cantidad de territorio mayor a la del oponente. Para controlar un área debe rodearse con las piedras. De esta forma, el jugador que controla la mayor cantidad de territorio al finalizar la partida gana.

La aplicación emula algunas reglas del juego *go*. Las reglas del juego se explican de forma más detallada en el manual de usuario.

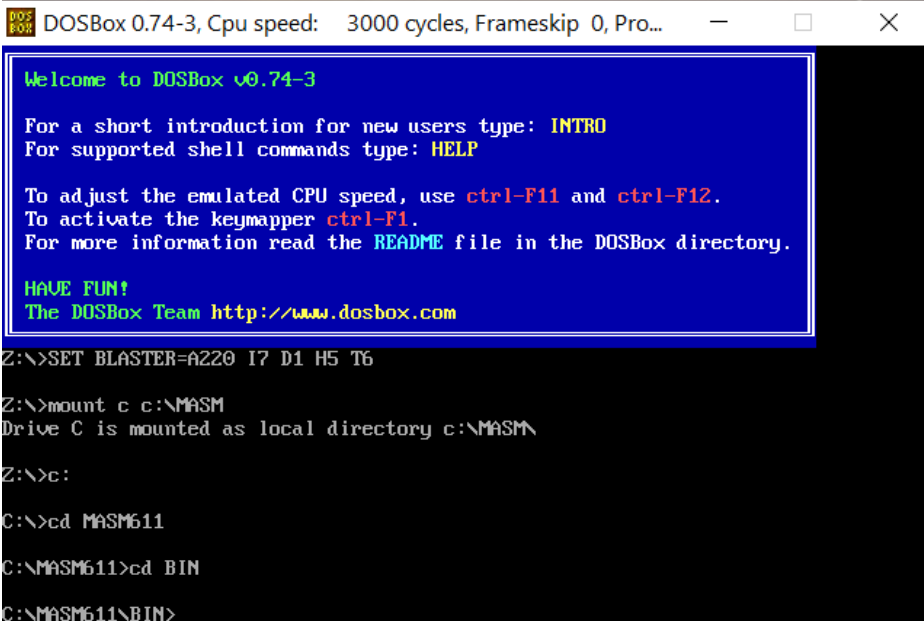
En esta aplicación no se utiliza el modo video, todo se maneja a través de la consola de la aplicación de DOSBox

4 REQUISITOS TÉCNICOS

Para el correcto funcionamiento y ejecución de la aplicación es necesario poseer el programa DOSBox instalado en el ordenador, y tenerlo debidamente configurado para compilar código en ensamblador o ejecutar el archivo .exe.

Para compilar el código ensamblador es necesario tener instalado MASM v6.1.

1. Deberá ejecutar el programa DOSBox:



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Welcome to DOSBox v0.74-3
For a short introduction for new users type: INTRO
For supported shell commands type: HELP

To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.

HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6

Z:\>mount c c:\MASM
Drive C is mounted as local directory c:\MASM\

Z:\>c:

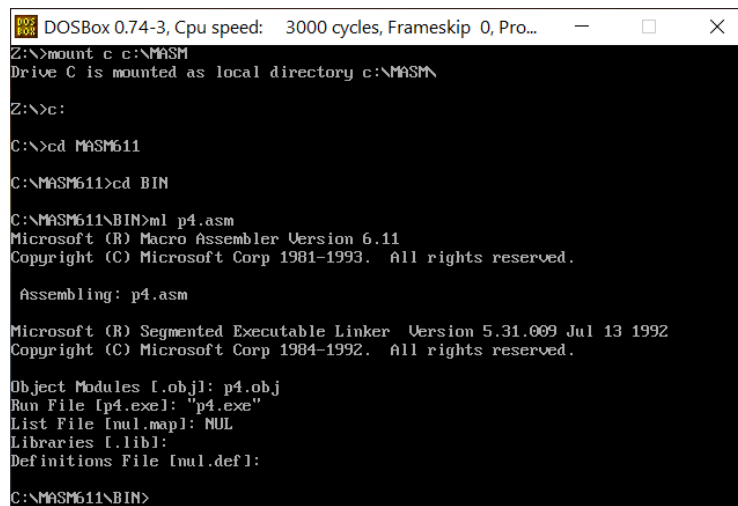
C:\>cd MASM611

C:\MASM611>cd BIN

C:\MASM611\BIN>
```

El archivo a compilar deberá estar en la carpeta montada

2. Compile el archivo utilizando el comando **ml**



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Pro...
Z:\>mount c c:\MASM
Drive C is mounted as local directory c:\MASM\
Z:\>c:
C:\>cd MASM611
C:\MASM611>cd BIN
C:\MASM611\BIN>ml p4.asm
Microsoft (R) Macro Assembler Version 6.11
Copyright (C) Microsoft Corp 1981-1993. All rights reserved.

Assembling: p4.asm

Microsoft (R) Segmented Executable Linker Version 5.31.009 Jul 13 1992
Copyright (C) Microsoft Corp 1984-1992. All rights reserved.

Object Modules [.obj]: p4.obj
Run File [p4.exe]: "p4.exe"
List File [nul.map]: NUL
Libraries [lib1]:
Definitions File [nul.def]:
C:\MASM611\BIN>
```

De esta forma el compilador generará el archivo ejecutable y estará disponible para su posterior uso².

El programa fue exitosamente ejecutado en un computador con las siguientes especificaciones:

- Windows 10 x64
- 8 GB RAM
- Intel® Core™ i7-8550U
- DOSBox 0.74-3
- MASM v6.1

Sin embargo, se asegura su correcta ejecución con las siguientes especificaciones:

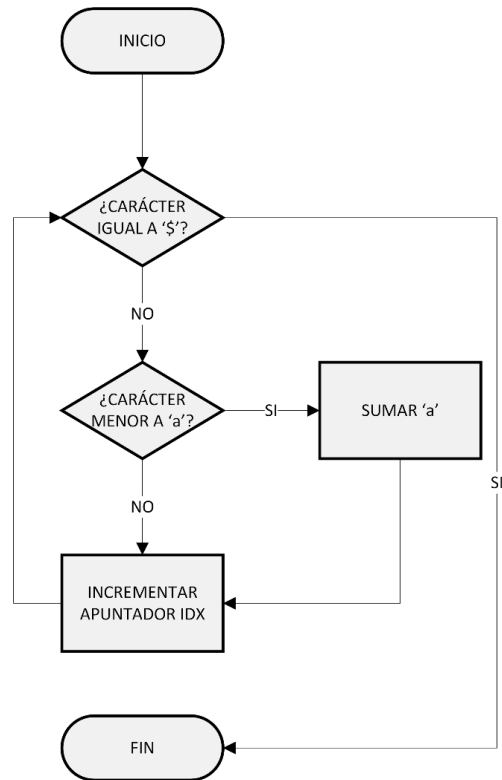
- Windows 8 o superior (x64 o x32 bits)
- Cualquier versión de DOSBox compatible con MASM 6.1
- MASM v6.1
- 2 GB RAM
- Intel® Core™ i3 (cualquier nomenclatura) o superior

² Ver manual técnico

5 FUNCIONALIDAD DE LA APLICACIÓN

5.1 FUNCIÓN TOLOWER

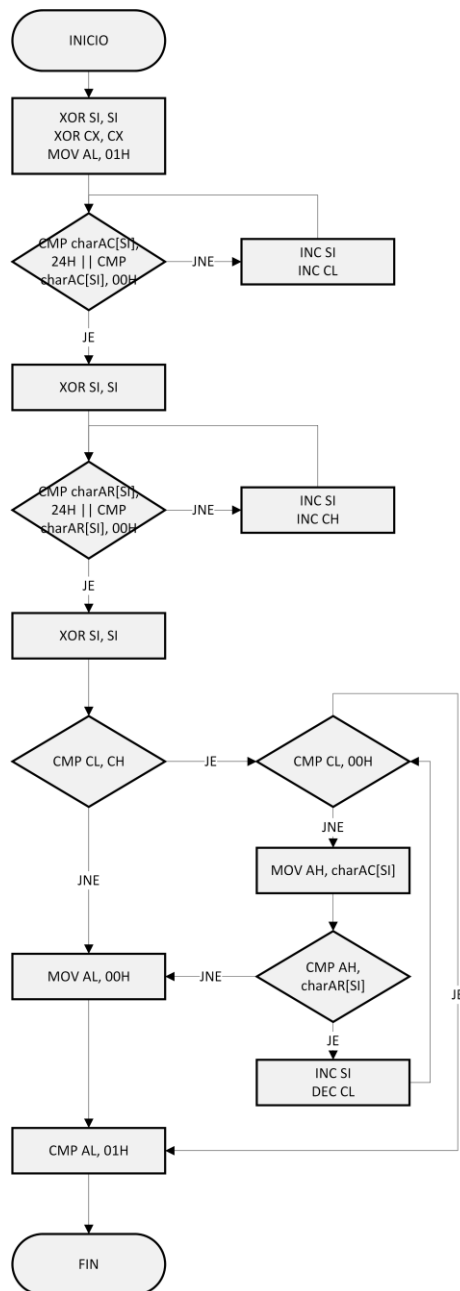
Dado un *string* de **letras** se pasa a minúscula



```
toLower MACRO charAC
LOCAL _1, _2, _3, _4
XOR SI, SI
_1:
    CMP charAC[SI], 24H ;ES IGUA
    JE _4              L A '$'
    CMP charAC[SI], 00H ;ES IGUAL A NUL
    JE _4              L
_2:
    CMP charAC[SI], 61h ;
    JAE _3             ;SI ES MAYOR O IGUA
    ADD charAC[SI], 20h L A
_3:
    INC SI
    JMP _1
_4:
ENDM
```

5.2 FUNCION COMPARESTR

Dados dos *string* de **letras**, se comparan y se determina si son iguales o no. Esta función tiene un comportamiento similar a la función *compare()* de C/C++. Primero cuenta el número de caracteres de ambos arreglos, si son iguales compara caractere por carácter utilizando su código ASCII. El registro *bandera* alojará el resultado luego de comparar el registro *acumulador* con el valor inmediato *01H*.



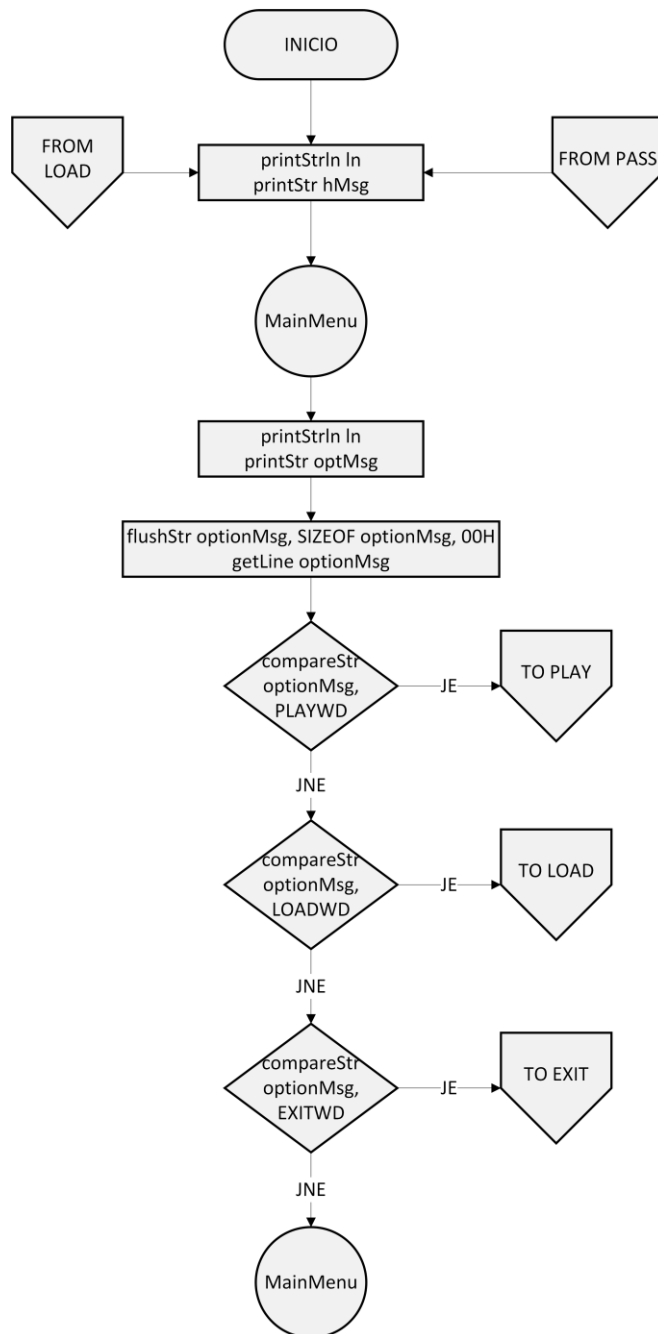
```

compareStr MACRO charAC, charAR
LOCAL _1, _2, _3, _4, _5, _6, _7, _8, _9
    XOR SI, SI
    XOR CX, CX
    MOV AL, 01H
    _1:
        CMP charAC[SI], 24H
        JE _2
        CMP charAC[SI], 00H
        JE _2
        INC SI
        INC CL
        JMP _1
    _2:
        XOR SI, SI
    _3:
        CMP charAR[SI], 24H
        JE _4
        CMP charAR[SI], 00H
        JE _4
        INC SI
        INC CH
        JMP _3
    _4:
        XOR SI, SI
    _5:
        CMP CL, CH
        JNE _8
    _6:
        CMP CL, 00H
        JE _9
    _7:
        MOV AH, charAC[SI]
        CMP AH, charAR[SI]
        JNE _8
        INC SI
        DEC CL
        JMP _6
    _8:
        MOV AL, 00H
    _9:
        CMP AL, 01H
ENDM

```

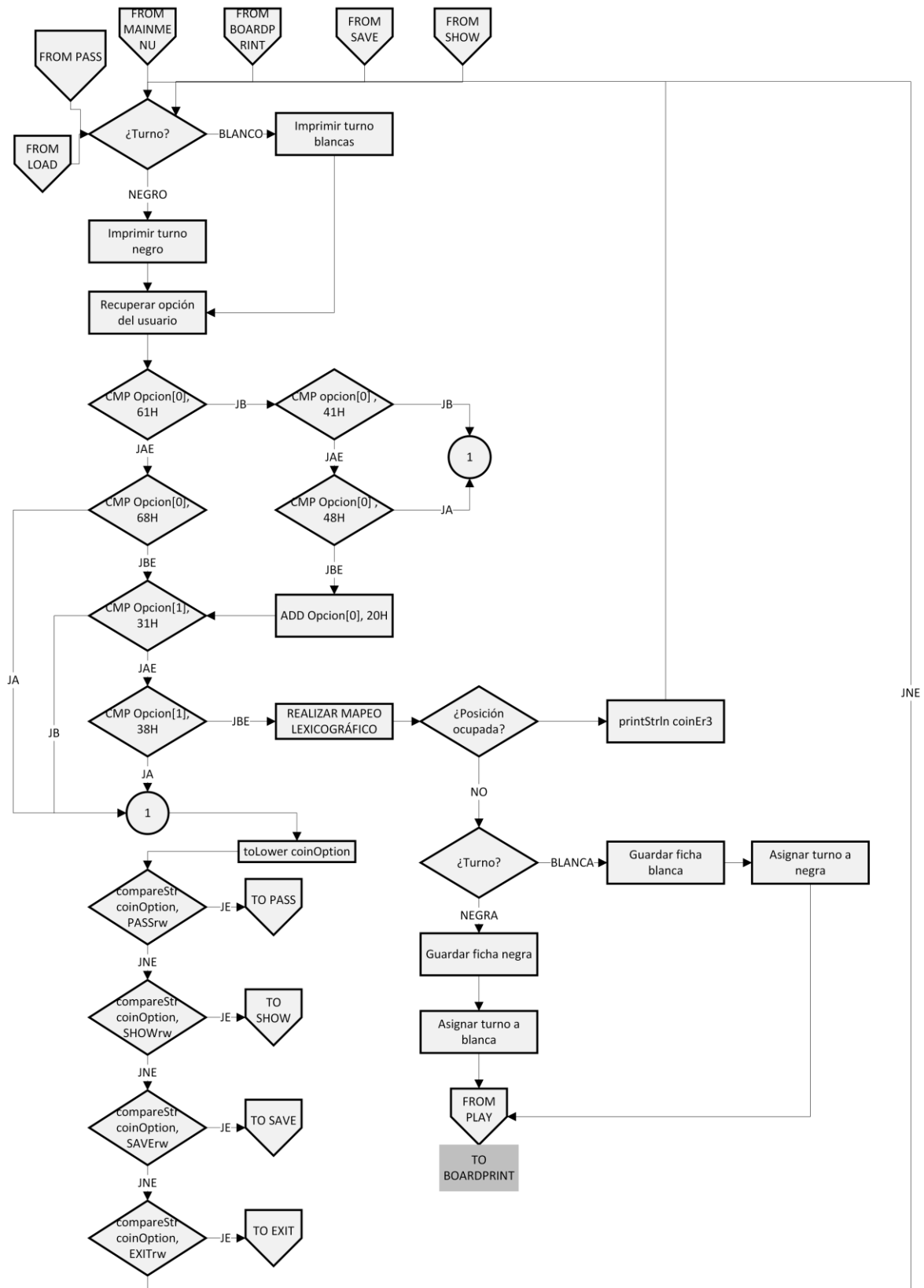
5.3 FLUJO DE MENU PRINCIPAL

El menú principal provee al usuario las rutas necesarias para acceder a las diferentes opciones el juego. El flujo del menú principal se explica en el diagrama de flujo. Este diagrama de flujo contiene etiquetas de referencia externa a otros diagramas que se explicarán más adelante.



```
MainMenu:
    printStrLn ln
    printStr optMsg
    flushStr optionMsg, sizeof optionMsg, 00H
    getline optionMsg
    compareStr optionMsg, PLAYWD
    JE Play
    compareStr optionMsg, LOADWD
    JE Load
    compareStr optionMsg, EXITWD
    JE Exit
    JMP MainMenu
```


5.4 FLUJO PLAY



```

Play:
    CMP actTurn, 01H                ;DETERMINA DE QUIÉN ES EL TURN
    JNE _printWhit                 ;NO ES IGUAL A 1,
    printStr trnMsg1                ;ES IGUAL A 1, ENTONCES ES EL TURNO DE LAS NEGRA
    JMP _play1                      ;SIGUE EL CURSO DE LA SECCIÓN PLA
    _printWhit                      Y
e:   printStr trnMs 2                ;ENTONCES ES EL TURNO DE LAS BLANCA
    _play1:                          S
        flushStr coinOptio    sizeof coinOption, 00H
        getline coinOptio     ;RECUPERA LA OPCIÓN DEL USUARI
n,   XOR AX, AX                     ;LIMPIA EL ACUMULADO
    CMP coinOption[0], 'a'         R
    JB _uppe                       ;CODIGO ASCII ES MENO
    CMP coinOption[0], 'h'         R A 'a'
    JBE _digi                       ;CODIGO ASCII ES MENOR O IGUA
    JMP _play2                     ;CODIGO ASCII ES MAYO
    _upper:                         R A 'h'
        CMP coinOption[0], 'A'
        JB _play2                 ;CODIGO ASCII ES MENO
        CMP coinOption[0], 'H'    R A 'A'
        JA _play2                 ;CODIGO ASCII ES MAYO
        ADD coinOption[0], 20H     ;SUBTRAHOCODIGO 32 para lograr una minúscul
    _digi:                          a
        CMP coinOption[1], 'i'
        JB _play2                 ;CODIGO ASCII ES MENO
        CMP coinOption[1], '8'    R A 'i'
        JBE _play3                 ;CODIGO ASCII ES MENOR O IGUA
    _play2:                         L A '8'
        toLower coinOptio
n     compareStr coinOption, PASSr
w     JE Pass
        compareStr coinOption, SAVER
w     JE Save
        compareStr coinOption, EXITr
w     JE ExitPlay
        compareStr coinOption, SHOWr
w     JE Reporte
        JMP Play
    _play3:
        SUB coinOption[0], 61H     ;OBTIENE UN INDICE DE COLUMNA, BASE 0 => coinOption[0] <- coinOption[0] - 61
        MOV AL, 38H                ;MUEVE UN OCHO ASCII AL ACUMULADOR-L, (PARA LUEGO MULTIPLICARLO) => AL <- 38
        SUB AL, coinOption[1]      ;OBTIENE UN INDICE DE FILA, BASE 0 => AL <- AL - coinOption
        XOR AH, AH                 ;LIMPIA AH
        SHL AX, 3                  ;MULTIPLICA POR OCHO
        ADD AL, coinOption[0]      ;SUMA LA COLUMNA
        XOR BH, BH                 ;LIMPIA EL INDICE BASE SUPERIO
        MOV BX, AX                 ;MUEVE EL RESULTADO A UN REGISTRO BAS
        CMP LOGICM[BX], 20H        ;COMPARA SI LA POSICION ES IGUAL A UN ESPACIO
        JE _play4                 ;ES UN ESPACIO DISPONIBL
        printStrln coinE 3        ;INFORMA AL USUARIO QUE ESA POSICIÓN YA ESTÁ OCUPAD
        JMP Pla                    ;REGRESA AL FLUJO DE PLA
r     _play4:y                      Y
        CMP actTurn, 01H          ;¿TURNO?
        JNE _playWhit            ;TURNO DE BLANCAS
        MOVELOGICM[BX], 4EH       ;GUARDA FICHA NEGRA
        DEC actTur                ;ASIGNA TURNO A BLANC
        INCnscoreN                A
        CALL verifyCatch
        JMP BoardPrint            ;IMPRIME TABLER
    _playWhite:                    O
        MOV LOGICM[BX], 42H       ;GUARDA FICHA BLANC
        INC actTur                ;ASIGNA TURNO A NEGR
        INCnscoreB                A
        CALL verifyCatch
        JMP BoardPrint

```

En este flujo, primero se limpia la entrada del usuario y luego se solicita la información. Luego se verifica si la información proporcionada por el usuario es una posición en el tablero o es un comando a ejecutar. Si es una posición, realiza un mapeo lexicográfico por filas y posiciona una 'B' o una 'N' (según el turno actual) en el arreglo lógico de posiciones (el cual simula un arreglo 2D). Si es un comando, primero lo pasa a minúsculas y luego lo compara, si existe un acierto el flujo normal se interrumpe y ocurre un salto hacia el flujo correspondiente.

En caso ser una posición, al finalizar la colocación de la ficha se verifica si este es un movimiento válido o bien si la colocación de la ficha desencadena la captura de una formación de ficha.

5.4.1 FLUJO VERIFYCATCH

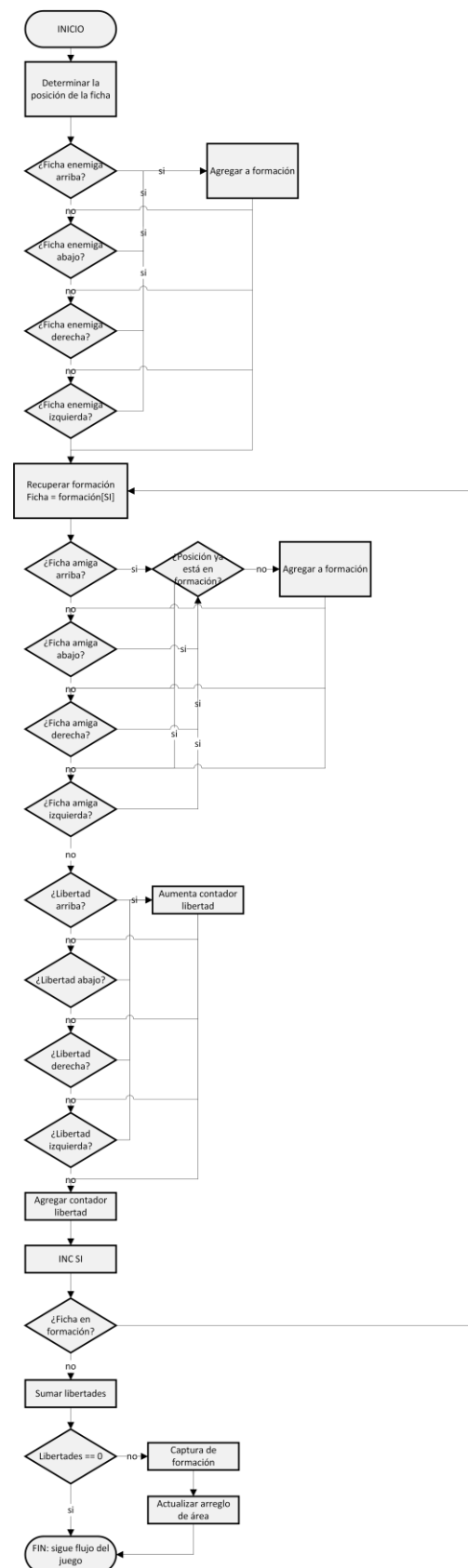
Durante este flujo y el flujo de verifySuicide se conserva el valor del índice base, el cual contiene el valor de la posición que se recuperó luego de realizar el mapeo lexicográfico.

En este flujo se verifica que cualquier formación de ficha “enemiga” de la ficha recién colocada no posea libertades en sus periferias.

Primero se cuenta y se guarda las fichas enemigas que se encuentren en la periferia de la ficha recién colocada. Luego, a través de un ciclo, se verifica si alrededor de las fichas enemigas guardada y contadas existen otras fichas enemigas, y así consecutivamente hasta no encontrar otra ficha enemiga. En cada iteración de esta búsqueda se contará las libertades de las fichas

Al finalizar este ciclo se suman las libertades de la formación identificada. Si la suma de libertades es 0 entonces, se captura toda la formación y es marcada como territorio ocupada por la ficha que ocasionó la captura, el punteo y el arreglo lógico es actualizado.

Si las libertades no suman 0 se procede a verificar si existe suicidio.



5.4.2 FLUJO VERIFYSUICIDE

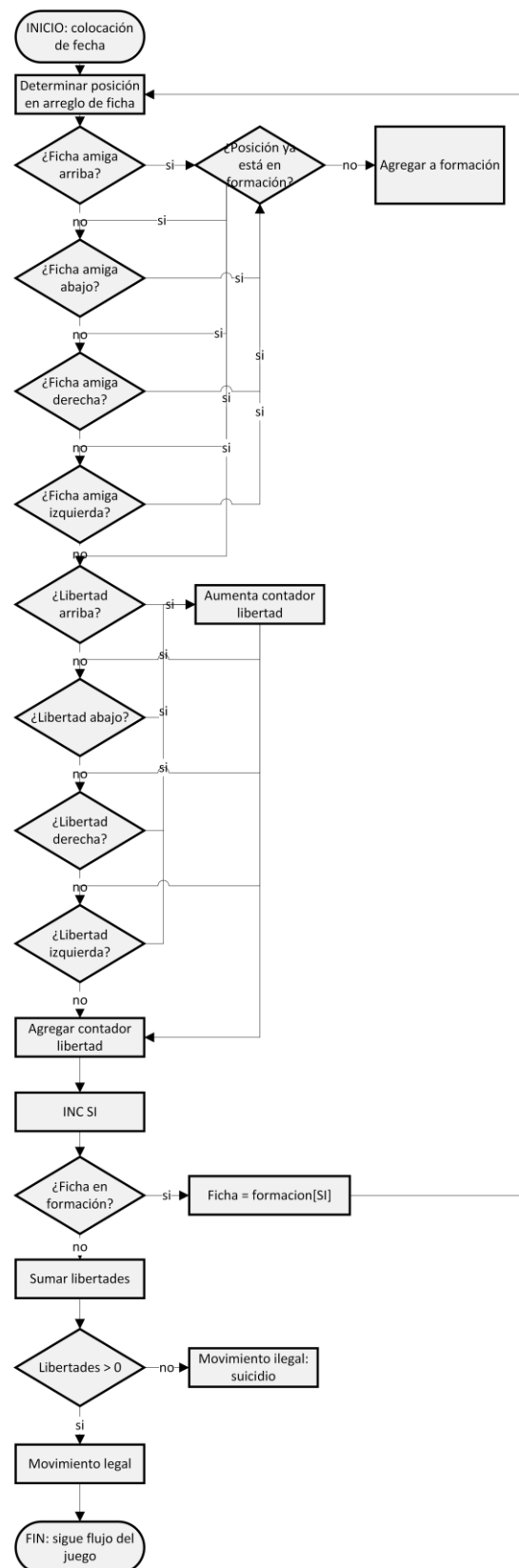
Luego de que una captura de fichas no ocurre se verifica que la colocación de ficha no incurra en una ilegalidad al ocasionar un suicidio.

Realiza una verificación similar a la que se realiza para verificar si existe una captura de formación.

A través de un ciclo se verifican las periferias de la ficha recién colocada, se cuenta y almacena cualquier ficha amiga encontrada. Al mismo tiempo se cuentan el número de libertades de cada ficha contada y almacenada.

Al finalizar el ciclo si la suma de las libertades de la formación identificada es igual a 0 entonces, se revierte la colocación de la ficha y se solicita al usuario que vuelva a ingresar una coordenada válida.

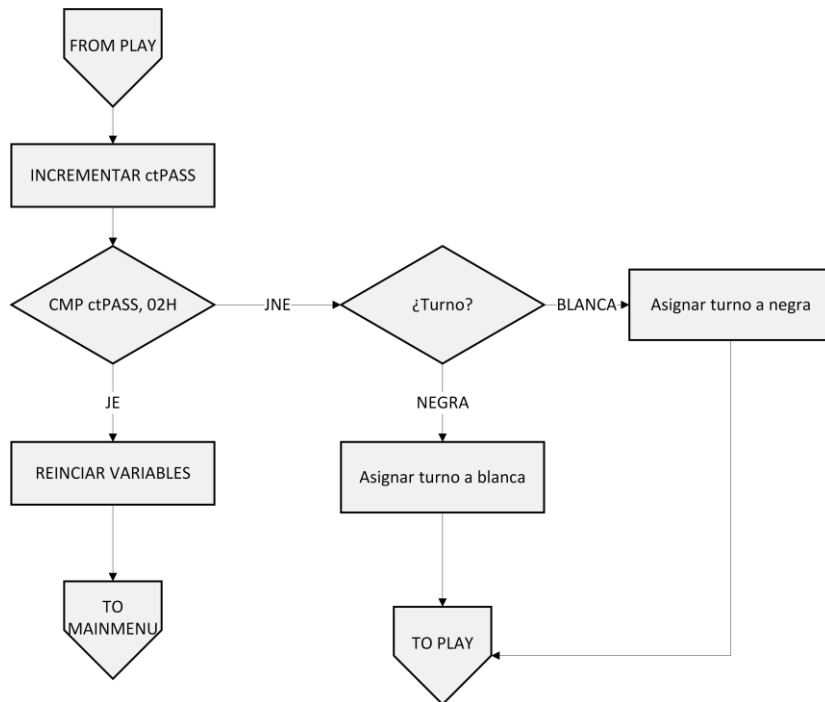
Si la suma es mayor a cero se considera un movimiento legal.



5.5 FLUJO PASS

Al ejecutarse este flujo automáticamente se incrementa el valor que contiene la variable ctPASS.

Luego se verifica si ese valor es igual a 2. Esto significa que no es necesario que PASS se ejecute consecutivamente si no que un jugador puede utilizarlo y tiempo después volver a utilizarlo y esto haría que el flujo del juego se interrumpiera hacia el Menú principal



```

Pass:
    INC ctPAS                      ;INCREMENTA EL CONTADO
    CMPSctPASS, 02H                R
    JNE _passTurn
    JMP ExitPlay
_passTurn:
    CMP actTurn, 01H
    JNE _whiteTurn
    DEC actTur
    ;DISMINUYE actTurn, ESO HARÁ QUE SEA EL TURNO DE LAS BLANCA
    S    JMP Pla
    ;REGRESA A YA SECCIÓN DE JUEG
0 _whiteTurn:
    INC actTur
    ;INCREMENTA actTurn, ESO HARÁ QUE SEA EL TURNO DE LAS NEGRA
    S    JMP Pla
    ;REGRESA A YA SECCIÓN DE JUEG
0
  
```

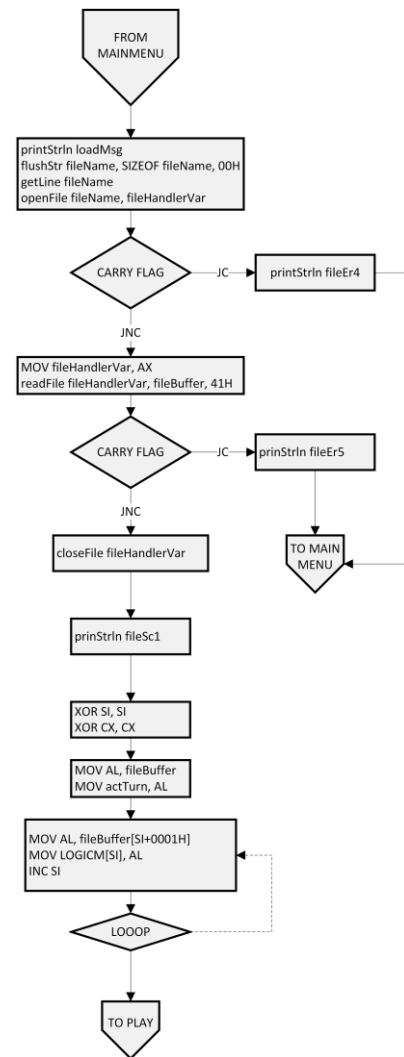
5.6 FLUJO LOAD

Cuando este comando se reconoce se procede a buscar el archivo con el nombre especificado en la **misma carpeta** desde donde se está ejecutando la aplicación. Si al momento de abrir el archivo ocurre un error se informará al usuario.

Si no existe error se leerá la información del archivo. La información alojada en un buffer se distribuirá en reconocer cual es el turno que prosigue y en llenar el arreglo lógico de posiciones con las fichas tal y como se encontraban cuando el juego se almacenó.

El archivo se cierra luego de finalizar su lectura

Al terminar de recuperar la información sobre el turno y las posiciones de la matriz lógica el flujo sigue en Play



```

Load:
    printStrln loadMsg          ;INFORMA AL USUARIO QUE INGRESE EL NOMBRE DEL ARCHIV
    flushStr fileName SIZEOF fileName, 00H ;LIMPIA LA CADENA EN LA QUE SE ALMACENARÁ EL NOMBRE DEL ARCHIV
    e, getline fileName         ;RECUPERA EL NOMBRE DEL ARCHIV
    e, openFile fileName, fileHandlerVa ;INTENTA ABRIR EL ARCHIV
    r JC _loadErr4              ;NO PUDO ABRIR EL ARCHIV
    MOV fileHandlerVar, AX      ;ALMACENA EL HANDLER DEL ARCHIV
    readFile fileHandlerVar, fileBuffer 41h;LEE EL CONTENIDO DEL ARCHIVO Y LO ALMACENA EN EL BUFE
    r, JC _loadErr5             ;NO PUDO LEER EL ARCHIV
    closeFile fileHandlerVa     ;CIERRA EL ARCHIV
    r printStrln fileSc1        ;INFORMA AL USUARIO QUE EL PROCESO FUE EXITOS
    c XOR SI, SI                ;LIMPIA EL REGISTRO INDIC
    XOR CX, CX                  ;LIMPIA EL REGISTRO DE CONTE
    MOV AL, fileBuffer          ;HUEVE EL PRIMER VALOR DEL BUFFER AL ACUMULADO
    SUB AL, 30H                 ;CONVIERTE EL SIMBOLO ASCII A UN NUMER
    MOV actTurn, AL             ;ALOJA EL TURNO EN EL QUE SE QUEDÓ EL JUEG
    _loadGam
e:    MOV AL, fileBuffer[SI + 0001H] ;HUEVE EL CONTENIDO DEL BUFFER AL ACUMULADO
    MOV LOGICM[SI], AL             ;HUEVE EL CONTENIDO DEL ACUMULADOR A LA MATRIZ LOGIC
    INC SI                         ;AUMENTA EL REGISTRO INDIC
    LOOP _loadGam                  ;LOOP
    JMP Pla e                      ;TERMINA EL PROCESO Y SALTA A Pla
    _loadErr4:
    printStrln fileE 4            ;INFORMA AL USUARIO QUE OCURRIÓ UN ERROR AL INTENTAR ABRIR EL ARCHIV
    r JMP MainMenu                ;
    _loadErr5:
    printStrln fileE 5            ;INFORMA LA USUARIO QUE OCURRIÓ UN ERROR AL INTENTAR LEER EL ARCHIV
    r closeFile fileHandlerVa     ;INTENTA CERRAR EL ARCHIV
    r JMP MainMenu                ;
    
```

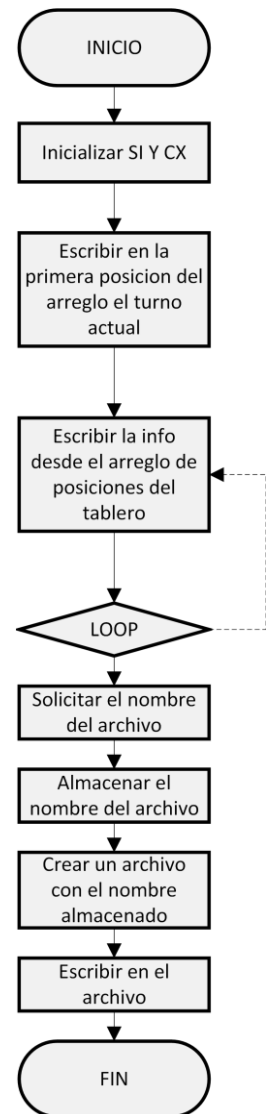
5.7 FLUJO SAVE

Al momento de ejecutar este comando se pedirá al usuario que ingrese **únicamente** el nombre del archivo y la extensión. Se creará el archivo y se procederá a alojar la información del turno actual y la información de las posiciones del tablero alojada en el arreglo lógico de posiciones.

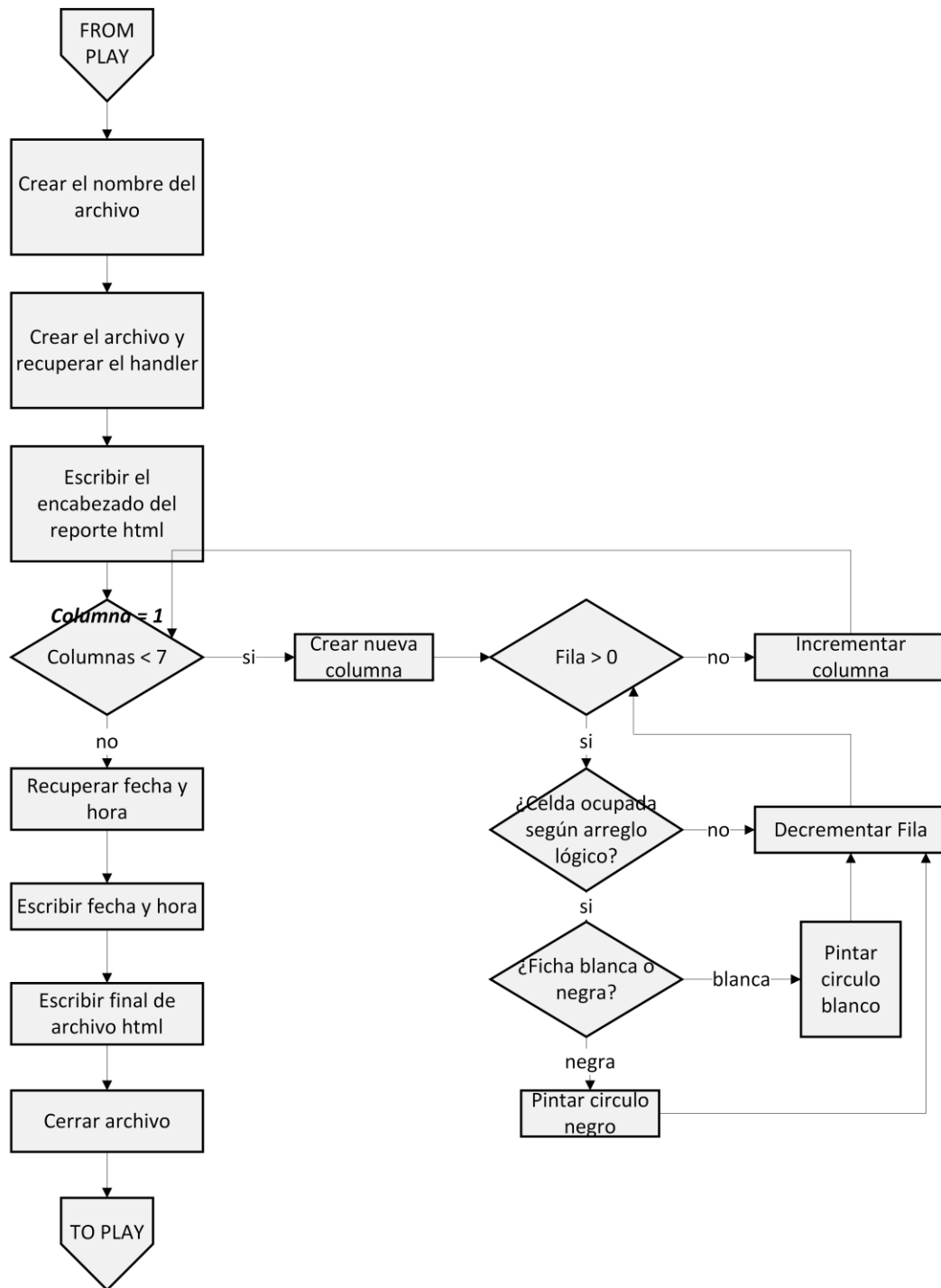
Al finalizar la composición de la información a escribir en el archivo, se escribe esta información en el archivo.

Se informa al usuario del resultado de la acción

```
Save:
XOR SI, SI          ;LIMPIA EL INDICE
XOR CX, CX          ;LIMPIA EL CONTEO
MOV CX, 0040H       ;INICIALIZA EL CONTEO (64)
MOV AL, actTur       ;ALOJA EL TURNO ACTUAL
ADD AL, 30H         ;CONVIERTE EL NUMERO EN UN CÓDIGO ASCII RECONOCIBLE
MOV fileBuffer, AL   ;ALMACENA EL CODIGO ASCII
_loopCreateBufFe
r:  MOV AL, LOGICM[SI] ;MUEVE EL VALOR DE LOGICM AL ACUMULADO
    MOV fileBuffer[SI+0001H], AL ;MUEVE EL ACUMULADOR AL BUFFER DE CONTENIDO DE ARCHIVO
    INC SI             ;INCREMENTA EL INDIC
    LOOP _loopCreateBufFe
    printStrLn svAsMs  ;SOLICITA EL NOMBRE DEL ARCHIVO
g  flushStr fileNam    SIZEOF fileName, 00H 0
e,  getline fileNam    ;RECUPERA EL NOMBRE DEL ARCHIVO
e  createFile fileNam  ;CREA EL ARCHIVO
e  JC _err1ToPla       ;EXISTIÓ UN ERROR
    MOV fileHandlerVar, AX ;ALMACENA EL HANDLE
    writeFile fileHandlerVar, fileBuffer 41H ;ESCRIBE EN EL ARCHIVO
r,  JC _err2ToPla       ;EXISTIÓ UN ERROR
    printStrLn svRsMs    ;ARCHIVO CORRECTAMENTE ESCRITO
g  closeFile fileHandlerVa ;CIERRA EL ARCHIVO
r  JMP Pla            ;REGRESA AL FLUJO DEL JUEGO
    _err1ToPla          0
y:  printStrLn fileE 1    ;NO SE PUDO CREAR EL ARCHIVO
r  JMP Pla            ;REGRESA AL FLUJO DEL JUEGO
    _err2ToPla          0
y:  printStrLn fileE 2    ;NO SE PUDO ESCRIBIR EN EL ARCHIVO
r  closeFile fileHandlerVa ;INTENTA CERRAR EL ARCHIVO
r  JMP Pla            ;REGRESA AL FLUJO DEL JUEGO
    v                  0
```



5.8 FLUJO SHOW



A través de este procedimiento se genera un archivo html que muestra el estado actual del tablero