UNSAM
UNIVERSIDAD NACIONAL DE SAN MARTÍN

Escuela de Ciencia y Tecnología

MATERIA: Sistema de Procesamiento de Datos

Profesores: Fabio Bruschetti
Pedro Irisso

Turno del TP: 18:00 a 22:00 hs

Cursada: 1er Cutrimestre del 2020

Fecha de entrega: 24/06/2020

Integrantes:

- Villamonte, Nicolás Matías
- Ibarra Bilbao, Adrián Joel
- Pastorini, Mateo Norl
- Oyola, Ezequiel Agustín
- Frechou, Lucas Osvaldo

TRABAJO PRÁCTICO FINAL:

"PONG by GruPong"

PONG Datasheet

Caracteristicas:

- **GROUND.asm, KEYBOARD.asm, TIMER.asm, TIMER2.asm PALETTE.asm, BALL.asm and SCORE.asm son archivos de tipo librería utilizado para la creación del videojuego PONG utilizando el MSDOS.**

El archivo PONG.EXE es un juego programado en lenguaje ensamblador para computadores de arquitectura 8086. Utiliza, internamente, al menos 47 subrutinas separadas en diferentes archivos que se complementan en el archivo principal para el correcto funcionamiento del videojuego.

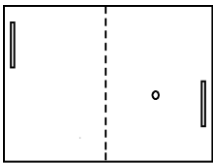
- **Manejo del teclado mediante el puerto 60h, utilizando Scancodes con facilidad.**
- **Conversion de Scancodes a Ascii**
  SC = 91h RL = 77h RH = 01
- **Funciones de control de tiempo incluido y de fácil utilización.**
- **Utilización del modo de video 12h para el menú y del modo de video 03h para el videojuego en algunos casos.**
- **Uso automatizado de sistema de coordenadas.**
- **Conversores de coordenadas a posición en un vector.**

pos(X,Y) = Y*(BYTESPL+3)+X

posY = abs[(POS-(3*Abs(Pos/ BYTESPL)))/ BYTESPL]

posX = mod[(POS-(3*Abs(Pos/ BYTESPL)))/ BYTESPL]

- **Su tamaño no supera los 15 Kb por versión de Pong, lo que no superaría los 75kb en total.**
- **La memoria necesaria para correr cada versión no superará los 500kb.**



Gru
Pong

Indice de los temas a abarcar en esta hoja de datos:

Archivo GROUND.ASM

Subrutina Draw

Dibuja o refresca en el DOS la interfaz del PONG actual.

LLAMADA:
screen db ….
….
call Draw

DEVUELVE:
Nada.

EFECTO: Se limpia la pantalla y se imprime el vector llamado SCREEN en la misma. El vector debe tener obligatoriamente 80x25 espacios de caracteres ASCII imprimibles ocupados en memoria.

Subrutina setCoord

Setear un valor ASCII en coordenadas XY.

LLAMADA:
AL = Valor ASCII a posicionar.
DL = Coordenada en Y
DH = Coordenada en X
call setCoord

DEVUELVE:
Nada.

EFECTO: Se reemplaza o sobrescribe el valor especificado en las coordenadas (X,Y) por el valor indicado en el vector SCREEN. Para que el cambio sea visible, se debe refrescar la pantalla con la subrutina Draw.

Subrutina getCoord

Obtener el valor ASCII en coordenadas XY.

LLAMADA:
DL = Coordenada en Y
DH = Coordenada en X
call getCoord

DEVUELVE:
AL = Valor ASCII en esa posicion.

EFECTO: Se busca en la posición especificada por las coordenadas (X,Y) en el vector SCREEN y se devuelve el hexadecimal obtenido en el mismo.

Subrutina chgCoord

Intercambiar los valores ASCII de dos coordenadas XY

LLAMADA:
PUSH1 = Coordenada X1
PUSH2 = Coordenada Y1
PUSH3 = Coordenada X2
PUSH4 = Coordenada Y2
call chgCoord

DEVUELVE:
AL = Valor ASCII en esa posicion.

EFECTO: Intercambia el valor ASCII de la coordenada (X1,Y1) con el valor ASCII de la coordenada (X2,Y2). Para que el cambio sea visible, se debe refrescar la pantalla con la subrutina Draw.

Subrutina getPos

Obtener el valor convertido de la posición de (X,Y) en el vector.

LLAMADA:
DL = Coordenada en Y
DH = Coordenada en X
call getPos

DEVUELVE:
DX = Valor convertido.

EFECTO: Convierte el sistema de coordenadas (X,Y) en el sistema de posicionamiento en el vector devolviendo, mediante estos datos, a cuantos bytes se encuentra la posición de memoria seleccionada del offset del vector SCREEN.

Subrutina getPosXY

Obtener las coordenadas (X,Y) mediante la posición.

LLAMADA:
DX = Posición desde el offset de SCREEN.
call getPosXY

DEVUELVE:
DL = Coordenada en Y
DH = Coordenada en X

EFECTO: Convierte la posición del byte seleccionado en el vector SCREEN al sistema de coordenadas (X,Y).

Subrutina fillSq

Setear varios valores ASCII en un conjunto de coordenadas XY

LLAMADA:
PUSH1 = Coordenada X1 (Parte alta) e Y1 (Parte baja)
PUSH2 = Coordenada X2 (Parte alta) e Y2 (Parte baja)
PUSH3 = Valor ASCII (Parte baja)
call fillSq

DEVUELVE:
Nada.

EFECTO: Convierte la posición del byte seleccionado en el vector SCREEN al sistema de coordenadas (X,Y).

Archivo KEYBOARD.ASM

Subrutina getScanCode

Recibe el Scancode del puerto del teclado.

LLAMADA:
call getScanCode

DEVUELVE:
AL = ScanCode

EFECTO: Devuelve el código escaneado en el puerto 60h de la interfaz de E/S, perteneciente al teclado.

Subrutina SCtoASCII

Convierte de SCANCODE del teclado a ASCII.

LLAMADA:
AL = ScanCode
AH = Modo (02h si es segunda lectura, solo para teclas especiales. Cualquier otro valor en caso contrario)
call SCtoASCII

DEVUELVE:
AL = Código ASCII (Las letras se devuelven todas en minúscula), en caso de modo 02h o tecla especial devuelve ScanCode.
AH = $\overline{Pressed}$ /$Realesed$ (Devuelve 02h si fue una tecla especial)

EFECTO: Se realiza una conversión del SCANCODE al código ASCII y se devuelve este mismo código junto con un valor que representa si la tecla esta presionada o se ha soltado. Si a la subrutina se la configura en modo 2, entonces el programa reconocerá que está trabajando con una tecla especial anteriormente ya reconocida y solamente devolverá si esa tecla esta presionada o se soltó, devolviendo el SCANCODE de la tecla cuando esta presionada (menor a 80h).

Devolución de teclas especiales en modo **NO** 02h:

TECLA	AH	AL	
		Presionado	Soltado
HOME	02h	47h	C7h
END	02h	4Fh	CFh
PGUP	02h	49h	C9h
PGDN	02h	51h	D1h
DEL	02h	53h	D3h
R-SHIFT	02h	36h	B6h
L-SHIFT	02h	2Ah	AAh
CTRL	02h	1Dh	9Dh
ALT	02h	38h	B8h

Devolución en modo 02h:

TECLA	AH	AL
HOME	$\overline{Pressed}$ /$Realesed$	47h
END	$\overline{Pressed}$ /$Realesed$	4Fh
PGUP	$\overline{Pressed}$ /$Realesed$	49h
PGDN	$\overline{Pressed}$ /$Realesed$	51h
DEL	$\overline{Pressed}$ /$Realesed$	53h
R-SHIFT	$\overline{Pressed}$ /$Realesed$	36h
L-SHIFT	$\overline{Pressed}$ /$Realesed$	2Ah
CTRL	$\overline{Pressed}$ /$Realesed$	1Dh
ALT	$\overline{Pressed}$ /$Realesed$	38h

Archivo TIMER.ASM

Subrutina TimRst

Resetea el contador de clocks del sistema.

LLAMADA:
call TimRst

DEVUELVE:
Nada.

EFECTO: Pone en cero al contador del clocks del sistema, el cual comenzará a contar nuevamente sin necesidad de enviarle otra instrucción.

Subrutina TimSet

Setea el contador de clocks del sistema utilizando el sistema sexagesimal.

LLAMADA:
PUSH1 = Horas - Minutos
PUSH2 = Segundos - Centésimo
call TimSet

DEVUELVE:
Nada.

EFECTO: Setea el valor del contador de clocks del sistema según horas, minutos, segundos y centésimas.

Subrutina TimSetC

Setea el contador de clocks del sistema utilizando cantidad de clocks.

LLAMADA:
PUSH1 = Parte más significativa del contador
PUSH2 = Parte menos significativa del contador
call TimSetC

DEVUELVE:
Nada.

EFECTO: Setea el valor del contador de clocks del sistema según cantidad de clocks. En cada segundo hay 18,2 ciclos.

Subrutina getTim (NO FUNCIONAL)

Recibe el valor del contador de clocks del sistema utilizando el sistema sexagesimal.

LLAMADA:
call getTim

DEVUELVE:
CH = Horas
CL = Minutos
DH = Segundos
DL = Centésimas

EFECTO: Devuelve el tiempo actual del contador en horas, minutos, segundos y centésimas.

Subrutina getTimC

Recibe el Scancode del puerto del teclado utilizando cantidad de clocks.

LLAMADA:
call getTimC

DEVUELVE:
CX = Parte más significativa del contador
DX = Parte menos significativa del contador

EFECTO: Devuelve el tiempo actual del contador en ciclos del clock. Se producen 18,2 ciclos por segundo.

Subrutina cmpMiliSec (NO FUNCIONAL)

Realiza una comparación de cumplimiento del tiempo transcurrido.

LLAMADA:
AX = Milisegundos
call cmpMicroSec

DEVUELVE:
Una comparación lista para hacer un salto condicional.

EFECTO: Si el tiempo pasado en el contador de clocks del sistema superó el tiempo recibido en el registro en AX entonces podrá utilizar un JE para saltar a otra parte del programa, en caso contrario el JE será salteado. Para realizar esto el proceso modifica los flags directamente.

Archivo TIMER2.ASM

Este archivo trabaja con una estructura de TIMER, la cual tiene guardado como vectores, de forma separada, el horario inicial, el día inicial, el tiempo pasado y ese tiempo pasado convertido en milisegundos, segundos y minutos.

Subrutina TimRst

Resetea el timer.

LLAMADA:
DI = Offset del objeto Timer.
call TimRst

DEVUELVE:
Nada.

EFECTO: Setea el tiempo transcurrido del Timer seleccionado en cero para comenzar a contar desde el momento en que se llama a la subrutina.

Subrutina refreshPassedTim

Actualizar el tiempo transcurrido en el Timer.

LLAMADA:
DI = Offset del objeto Timer.
call refreshPassedTim

DEVUELVE:
Nada.

EFECTO: Actualiza el tiempo transcurrido desde el reset del Timer hasta el momento actual mediante una serie de cálculos que realiza internamente la subrutina. Sin la actualización del Timer el mismo podría mantener su ultimo refresco y no el tiempo real que ha pasado.

Subrutina getPassedMilisec

Obtener el tiempo transcurrido en milisegundos.

LLAMADA:
DI = Offset del objeto Timer.
call getPassedMilisec

DEVUELVE:
CX = Parte alta de los milisegundos.
DX = Parte baja de los milisegundos.

EFECTO: Devuelve, en cantidad de milisegundos, el tiempo transcurrido desde el Reset del Timer seleccionado. No es necesario refrescar el Timer para su correcto funcionamiento.

Subrutina getPassedSeconds

Obtener el tiempo transcurrido en segundos.

LLAMADA:
DI = Offset del objeto Timer.
call getPassedSeconds

DEVUELVE:
CX = Parte alta de los segundos.
DX = Parte baja de los segundos.

EFECTO: Devuelve, en cantidad de segundos, el tiempo transcurrido desde el Reset del Timer seleccionado. No es necesario refrescar el Timer para su correcto funcionamiento.

Subrutina getPassedMinutes

Obtener el tiempo transcurrido en minutos.

LLAMADA:
DI = Offset del objeto Timer.
call getPassedMinutes

DEVUELVE:
DX = Minutos pasados.

EFECTO: Devuelve, en cantidad de minutos, el tiempo transcurrido desde el Reset del Timer seleccionado. No es necesario refrescar el Timer para su correcto funcionamiento.

Subrutina cmpMilisec

Compara si el timer seleccionado ha pasado un tiempo dado en milisegundos.

LLAMADA:
DI = Offset del objeto Timer.
DX = Milisegundos de 0000h a FFFFh.
call cmpMilisec

DEVUELVE:
Zero Flag en 1 si el tiempo introducido en DX ya pasó.
Zero Flag en 0 si el tiempo introducido en DX no pasó.

EFECTO: Realiza una comparación entre el TImer seleccionado y los valores de tiempo en milisegundos obtenidos. Si el temporizador superó o igualó los valores de tiempo pedidos entonces la comparación será verdadera, en caso contrario será falsa. Solo se podrán adoptar valores de 0 a 65535 milisegundos. No es necesario refrescar el Timer para su correcto funcionamiento.

Subrutina cmpSeconds

Compara si el timer seleccionado ha pasado un tiempo dado en segundos.

LLAMADA:
DI = Offset del objeto Timer.
DX = Segundos de 0000h a FFFFh.
call cmpSeconds

DEVUELVE:
Zero Flag en 1 si el tiempo introducido en DX ya pasó.
Zero Flag en 0 si el tiempo introducido en DX no pasó.

EFECTO: Realiza una comparación entre el Timer seleccionado y los valores de tiempo en segundos obtenidos. Si el temporizador superó o igualó los valores de tiempo pedidos entonces la comparación será verdadera, en caso contrario será falsa. Solo se podrán adoptar valores de 0 a 65535 segundos. Los valores siempre se redondean para abajo. No es necesario refrescar el Timer para su correcto funcionamiento.

Subrutina cmpMinutes

Compara si el timer seleccionado ha pasado un tiempo dado en minutos.

LLAMADA:
DI = Offset del objeto Timer.
DX = Minutos de 0000h a 059Fh.
call cmpSeconds

DEVUELVE:
Zero Flag en 1 si el tiempo introducido en DX ya pasó.
Zero Flag en 0 si el tiempo introducido en DX no pasó.

EFECTO: Realiza una comparación entre el Timer seleccionado y los valores de tiempo en segundos obtenidos. Si el temporizador superó o igualó los valores de tiempo pedidos entonces la comparación será verdadera, en caso contrario será falsa. Solo se podrán adoptar valores de 0 a 1439 minutos. Los valores siempre se redondean para abajo. No es necesario refrescar el Timer para su correcto funcionamiento.

Subrutina delayMilisec

Hacer una espera de tiempo en milisegundos.

LLAMADA:
PUSH = Milisegundos de 0000h a FFFFh.
call delayMilisec

DEVUELVE:
NADA.

EFECTO: Realiza una espera de X milisegundos. Los valores pueden rondar entre 0 y 65535 milisegundos. No es necesario haber creado un Timer para ejecutar esta instrucción.

Subrutina delaySeconds

Hacer una espera de tiempo en segundos.

LLAMADA:
PUSH = Segundos de 0000h a FFFFh.
call delaySeconds

DEVUELVE:
NADA.

EFECTO: Realiza una espera de X segundos. Los valores pueden rondar entre 0 y 65535 segundos.

Subrutina delayMinutes

Hacer una espera de tiempo en minutos.

LLAMADA:
PUSH = Minutos de 0000h a 059Fh.
call delayMinutes

DEVUELVE:
NADA.

EFECTO: Realiza una espera de X milisegundos. Los valores pueden rondar entre 0 y 65535 milisegundos.

Estructura utilizada:

```
TIMER STRUC
        initH           db          ?
        initM           db          ?
        initS           db          ?
        initC           db          ?
        initDate        db          ?
        passedH         db          ?
        passedM         db          ?
        passedS         db          ?
        passedC         db          ?
        miliSec         dw          2 dup (?)
        Seconds         dw          2 dup (?)
        Minutes         dw          ?
ENDS
```

Archivo PALETTE.ASM

Este archivo tiene, dentro del mismo, definida una estructura de datos que refiere a las características de las paletas. Con un valor de coordenada en X, el cual será fijo a lo largo de todo el programa, un valor de coordenada en Y, el cual variará, un valor del ASCII que representará a la paleta dibujada en pantalla y un largo de la paleta. Las coordenadas (X,Y) se referirán al primer valor de la paleta buscando de arriba para abajo.

Subrutina initializatePalette

Inicializa el objeto PALETTE para evitar futuros problemas.

LLAMADA:
DI = Offset del objeto pelota.
call initializateBall

DEVUELVE:
Nada.

EFECTO: En forma de constructor, es imprescindible utilizar esta función antes de comenzar a realizar cambios en el objeto para evitar futuros problemas.

Subrutina refreshPalette

Dibuja o refresca, en el vector SCREEN, los cambios realizados en la paleta.

LLAMADA:
DI = Offset del objeto paleta.
call refreshPalette

DEVUELVE:
Nada.

EFECTO: Cambia, según los valores de coordenadas obtenidos en el objeto PALETTE, los valores ASCII del vector SCREEN de la librería GROUND.asm para dibujar o refrescar la paleta.

Subrutina setPalettePos

Cambia de posición la paleta con el sistema de coordenadas (X,Y).

LLAMADA:
DI = Offset del objeto paleta.
DH = Coordenada x (Si es FFh se mantiene la posición actual).
DL = Coordenada y (Si es FFh se mantiene la posición actual).
call setPalletePos

DEVUELVE:
Nada.

EFECTO: Se cambian las coordenadas de la paleta seleccionada según las coordenadas recibidas como argumento. Las coordenadas representarán el primer carácter ASCII que conforme la paleta de arriba para abajo. Para que esta operación sea visible se debe llamar a la subrutina refreshPalette.

Subrutina getPalettePos

Obtener de posición la paleta con el sistema de coordenadas (X,Y).

LLAMADA:
DI = Offset del objeto paleta.
call getPalletePos

DEVUELVE:
DH = Coordenada x.
DL = Coordenada y.

EFECTO: Se reciben las coordenadas (X,Y) de la paleta seleccionada. Las mismas representan el primer carácter ASCII que la conforme de arriba para abajo.

Subrutina goUp

Incrementar en uno la coordenada Y de la paleta seleccionada.

LLAMADA:
DI = Offset del objeto paleta.
call goUp

DEVUELVE:
Nada.

EFECTO: Desplaza la paleta seleccionada un espacio para arriba. Para que esta operación sea visible se debe llamar a la subrutina refreshPalette.

Subrutina goDown

Decrementar en uno la coordenada Y de la paleta seleccionada.

LLAMADA:
DI = Offset del objeto paleta.
call goDown

DEVUELVE:
Nada.

EFECTO: Desplaza la paleta seleccionada un espacio para abajo. Para que esta operación sea visible se debe llamar a la subrutina refreshPalette.

Subrutina getPaletteLarge

Obtener la longitud de la paleta.

LLAMADA:
DI = Offset del objeto paleta.
call getPalleteLarge

DEVUELVE:
DL = Largo de la paleta.

EFECTO: Devuelve la longitud en unidades de caracteres de la paleta seleccionada.

Subrutina setPaletteLarge

Setear la longitud de la paleta.

LLAMADA:
DI = Offset del objeto paleta.
DL = Nuevo largo de la paleta.
call setPalleteLarge

DEVUELVE:
Nada.

EFECTO: Cambia la longitud en unidades de caracteres de la paleta seleccionada.

Subrutina isPaletteAt

Verificar si se encuentra una porción de la paleta en las coordenadas (X,Y).

LLAMADA:
DI = Offset del objeto paleta.
DH = Coordenada x.
DL = Coordenada y.
call isPalletteAt

DEVUELVE:
DL = $\overline{No\ esquina}$ /$Esquina$.
DH = Posición de la paleta contando desde 1 (0 si no existe).
Zero Flag = $\overline{No\ existe}$ /$Existe$.

EFECTO: Se verifica si en las coordenadas existe una porción de la paleta seleccionada y cambia el flag de comparación según esa operación. Para esto se toman en cuenta las coordenadas (X,Y) enviadas como parámetros y el largo de la paleta. También se devuelve, en caso de que exista una porción de la paleta, si la misma es una de sus esquinas.

Estructura utilizada:

```
PALETTE STRUC
        coordx          db      ?
        coordY          db      ?
        graf            db      ?
        largo           db      ?
        p_old_coordX    db      ?
        p_old_coordY    db      ?
ENDS
```

Archivo BALL.ASM

Este archivo tiene, dentro del mismo, definida una estructura de datos que refiere a las características de la pelota. Con un valor de coordenada en X, un valor de coordenada en Y, un valor del ASCII que representará a la pelota dibujada en pantalla y un modo de dirección con la que se moverá la misma, la cual puede variar dependiendo dónde rebote. Existirán 3 tipos de modo de direccionamiento y una combinación de 4 tipos de modos de Sentido. Con las siguientes tablas se define la dirección y sentido del objeto BALL.

Modo de direccionamiento	Cantidad de mov en coord. X	Cantidad de mov en coord. Y
01h	+1	+1
02h	+2	+1
03h	+3	+1

Modo de Sentido		Direccion de mov en coord. X	Direccion de mov en coord. Y
Parte Alta	Parte Baja		
00h	**00h**	Izquierda (-)	Abajo (-)
00h	**01h**	Izquierda (-)	Arriba (+)
01h	**00h**	Derecha (+)	Abajo (-)
01h	**01h**	Derecha (+)	Arriba (+)

Subrutina initializateBall

Inicializa el objeto BALL para evitar futuros problemas.

LLAMADA:
DI = Offset del objeto pelota.
call initializateBall

DEVUELVE:
Nada.

EFECTO: En forma de constructor, es imprescindible utilizar esta función antes de comenzar a realizar cambios en el objeto para evitar futuros problemas.

Subrutina refreshBall

Cambia las coordenadas del objeto de tipo BALL.

LLAMADA:
DI = Offset del objeto pelota.
call refreshBall

DEVUELVE:
Nada.

EFECTO: Cambia, según los valores de coordenadas obtenidos en el objeto BALL, los valores ASCII del vector SCREEN de la librería GROUND.asm para dibujar o refrescar la pelota.

Subrutina moveBall

Cambia las coordenadas del objeto de tipo BALL.

LLAMADA:
DI = Offset del objeto pelota.
call moveBall

DEVUELVE:
Nada.

EFECTO: Cambia las coordenadas de la pelota teniendo en cuenta el modo de direccionamiento y sentido que tiene seleccionado el objeto BALL. No es necesario refrescar el objeto BALL al utilizar esta función.

Subrutina initialPosBall

Mueve la pelota a la posición central de una paleta.

LLAMADA:
DI = Offset del objeto pelota.
SI = Offset del objeto paleta.
DL = $\overline{Lado\ Izquierdo}$ /$Lado\ Derecho$.
call initialPosBall

DEVUELVE:
Nada.

EFECTO: Mueve la pelota a la posición central de la paleta seleccionada.

Subrutina getBallPos

Devuelve las coordenadas del objeto BALL seleccionado en sistema de coordenadas (X,Y).

LLAMADA:
DI = Offset del objeto pelota.
call getBallPos

DEVUELVE:
DH = Coordenada X de la posición del objeto.
DL = Coordenada Y de la posición del objeto.

EFECTO: Devuelve los valores de las coordenadas (X,Y) de la posición del objeto de tipo BALL seleccionado.

Subrutina setBallPos

Setea las coordenadas del objeto BALL seleccionado en sistema de coordenadas (X,Y).

LLAMADA:
DI = Offset del objeto pelota.
DH = Coordenada X de la nueva posición del objeto (Si es FFh se mantiene la posición actual).
DL = Coordenada Y de la nueva posición del objeto (Si es FFh se mantiene la posición actual).
call setBallPos

DEVUELVE:
NADA.

EFECTO: Cambia las coordenadas (X,Y) del objeto de tipo BALL seleccionado a los valores ingresados.

Subrutina getBallDirec

Obtener el modo de dirección y sentido actual del objeto BALL seleccionado.

LLAMADA:
DI = Offset del objeto pelota.
call getBallDirec

DEVUELVE:
AL = Modo de direccionamiento.
DH = Parte Alta del modo de sentido.
DL = Parte Baja del modo de sentido.

EFECTO: Devuelve todos los modos de direccionamiento del objeto tipo BALL, los cuales son el “Modo de direccionamiento” (El cuál decide el ángulo con el que se mueve el objeto) y el “Modo de sentido” (El cuál decide el sentido en el que se moverá el objeto).

Subrutina setBallDirec

Setea el modo de dirección y sentido del objeto BALL seleccionado.

LLAMADA:
DI = Offset del objeto pelota.
PUSH1 = Modo de direccionamiento. (Si es FFh se mantiene el modo de ángulo actual).
PUSH2 = Parte Alta del modo de sentido. (Si es FFh se mantiene la parte alta del modo de sentido actual).
PUSH3 = Parte Baja del modo de sentido. (Si es FFh se mantiene la parte baja del modo de sentido actual).
call setBallDirec

DEVUELVE:
NADA.

EFECTO: Cambia los modos de direccionamiento y sentido del objeto de tipo BALL seleccionado a los valores ingresados.

Estructura utilizada:

```
BALL STRUC
        coordx          db      ?
        coordY          db      ?
        graf            db      ?
        direccion       db      ?
        sentido         db      2 dup (?)
        old_coordX      db      ?
        old_coordY      db      ?
        backup_ASCII    db      ?
ENDS
```

Archivo SCORE.ASM

Subrutina ShowScore

Muestra la puntuación en pantalla.

LLAMADA:
AH = Puntuacion del jugador 1.
AL = Puntuacion del jugador 2.
call ShowScore

DEVUELVE:
NADA.

EFECTO: Dibuja en pantalla la puntuación sin borrar lo que se encuentra en el fondo. Los valores a dibujar en pantalla deben ser argumentados para llamar a esta subrutina. Los valores no deben superar al número 99 en sistema decimal.

Subrutina EraseScore

Borra la puntuación mostrada en pantalla.

LLAMADA:
call EraseScore

DEVUELVE:
NADA.

EFECTO: Borra la puntuación dibujada en pantalla, sin borrar nada más. Para borrarlo se reemplazan los caracteres ASCII imprimidos por espacios nuevamente.

Archivo SOUND.ASM

Subrutina StartSound

Habilita el parlante del dispositivo para emitir el sonido.

LLAMADA:
call StartSound

DEVUELVE:
NADA.

EFECTO: Comienza a sonar un sonido en el parlante con una frecuencia de 200Hz por defecto.

Subrutina StopSound

Deshabilita el parlante del dispositivo para dejar de emitir el sonido.

LLAMADA:
call StopSound

DEVUELVE:
NADA.

EFECTO: Deja de emitir el sonido que anteriormente se ha comenzado a emitir. Si ningún sonido se emitió anteriormente entonces no realiza nada, aunque podrían sucederse algunos problemas si esto se lleva a cabo, por lo que no es recomendable usarla sin haber utilizado la función StartSound antes.

PONG.ASM

Cada uno de los integrantes del grupo del GruPong creará un archivo “PONG.asm” diferente utilizando las librerías anteriormente mencionadas. Por lo que deberían existir en total 5 versiones diferentes del Pong. Para poder ejecutar las diferentes versiones del Pong se instalará junto con todos los archivos “PONG.asm” un archivo bat en la carpeta del TASM quien se encargará de correr el juego y en el que el jugador seleccionará cuál de todas las versiones del Pong quiere jugar. Este archivo, programado en batch, se utilizará en la consola como un comando a ejecutar, el cual si se lo ejecuta de manera incorrecta lanzará la ayuda de cómo utilizarlo correctamente.

Como cada versión del Pong corre a cuenta propia de cada uno de los programadores del GruPong, ellos mismos son los que decidirán qué librerías utilizar de las creadas anteriormente, de que forma va a ser el juego, cómo serán sus controles e incluso puede haber decidido crear nuevas librerías o utilizar otras librerías externas, por lo que tal vez este Datasheet no contenga absolutamente todo lo necesario para entender el juego en su totalidad, pero si todo el contenido creado por la corporación GruPong.

El juego en sí se podrá obtener utilizando un instalador, el cual se encargará de pegar todos los archivos necesarios en la carpeta donde se encuentra el TASM. Una vez instalado se puede utilizar simplemente abriendo el TASM y tipeando el comando “PONGEXE [Versión]”, siendo la versión el nombre del participante del GruPong que se quiere ver cuya versión. Si necesita ayuda con el comando puede tipear “PONGEXE HELP” o simplemente “PONGEXE” sin ningún argumento. Allí le mostrará una lista de versiones del Pong disponibles.

Este trabajo se le atribuye a todo el Grupong que esta conformado por Fabio Bruschetti y Pedro Iriso como creadores del proyecto. Nicolás Villamonte como el organizador del proyecto y programador. Adrián Ibarra, Mateo Pastorini, Lucas Frechou y Ezequiel Oyola como programadores claves del mismo.

Gru
Pong

Utilizaciones

Se utilizaron las siguientes interrupciones para llevar a cabo el proyecto:

- Interrupcion 10h: Servicio de video (BIOS).
- Interrupcion 1Ah: Manejo del clock.
- Interrupcion 21h: Servicios múltiples del sistema operativo MS-DOS.

Se utilizaron los siguientes puertos de entrada y salida para llevar a cabo el proyecto:

- Puerto 21h: Registro de mascara de interrupciones del 8259A.
- Puerto 60h: Puerto de estado del teclado.
- Puerto 61h: Puerto de control del parlante.