TECNOLÓGICO NACIONAL DE MEXICO
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

Carrera: Sistemas Computacionales

Unidad 4. Programación de dispositivos.

Alumno:

Reyes Villar Luis Ricardo | 21070343

Profesora: Guadalupe Martínez Jauregui

Materia: Lenguajes de Interfaz

Hora: 09:00 – 10:00 hrs

Grupo: 6502-C

Semestre: Agosto 2023 – Diciembre 2023

4.1 El buffer de video en modo texto.

Un buffer (o búfer en español) es un espacio de memoria dedicado al almacenamiento temporal de información, en el cual se guardan datos durante el tiempo de espera antes de ser procesados. El ejemplo más común de un buffer es el streaming, que es la ejecución de audio o video sin descargarlo a la computadora o dispositivo en el que se quiere utilizar, ya que de no utilizar un búfer de video la posibilidad de que se corte la reproducción por un problema en el ancho de banda es mayor.

El modo texto es un modo de video en el cual el contenido de la pantalla se representa por medio de caracteres en vez de pixeles individuales. La pantalla se muestra como un plano el cual contiene celdas de caracteres, cada una de las cuales almacena un carácter para desplegarlo. Generalmente los caracteres de los que se dispone al utilizar el modo texto son los caracteres ASCII.

El uso del modo texto se extendió mayormente durante los 70's cuando las terminales de texto orientados a video empezaron a volverse el estándar.

El uso del modo texto respecto al grafico permite un menor consumo de memoria, una manipulación de pantalla más rápida y requisitos de ancho de banda menores en uso remoto, aunque tiene la desventaja de que solo puede desplegar los caracteres predefinidos que posee, lo que es la mayor limitante para su uso.

Ensamblador utiliza el modo texto para desplegar en pantalla, por lo que, aunque puede ser muy eficiente para el manejo de recursos, no puede tener interfaces amigables con el usuario.

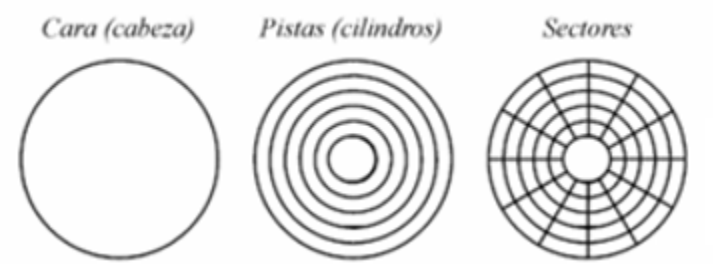
4.2 Acceso a discos en lenguaje ensamblador.

El acceso a discos desde ensamblador es complejo y requiere que se entiendan varios conceptos para poder utilizarlo.

El primero es que para poder escribir o leer desde un disco, se tiene que realizar en bloques de bytes y no byte por byte ya que el acceso implica movimientos mecánicos del disco y que la información en él se almacena por sectores. El proceso de acceso a disco no es llevado a cabo por la UCP, ésta solo envía comandos a la interfaz que maneja la unidad de disco, la cual se encarga de leer o escribir información del área de acceso directo a memoria o DMA (Direct Memory Access), la cual es un espacio de memoria especial para estos procesos.

El segundo concepto importante es cómo está constituido un disco de almacenamiento. Los discos están conformados físicamente por discos a los que se

les denomina caras, los cuales estan divididos en anillos concéntricos a los cuales se les denomina pistas (track) y está dividido en un número determinado de sectores, los cuales son divisiones de las pistas.



A continuación, se presentan tres formas de utilizar el acceso a discos, siendo la primera lectura, la segunda escritura y la tercera calcular espacio libre en un disco.

AbsoluteRead:

Transfiere el contenido de uno o más sectores del disco al buffer especificado, accesando directamente a los sectores lógicos. En caso de error, se enciende el bit de acarreo y AX contiene el código de error.

```
_AbsoluteRead PROC NEAR
ARG Buffer:DWORD,
Start:WORD,NumSect:WORD,
Drive:BYTE= ArgLen          →    Inicializar búfer.
push bp                     →    Almacenar en pila.
mov bp,sp                   →    Almacenar valor bp.
push bx                     →    Permitir acceso a  los argumentos.
push cx                     →    Salvar registros.
push dx                     →    Salvar registros.
push ds                     →    Salvar registros.
mov al,Drive                →    Mover valor de Drive a al.
```

mov cx,NumSect → Mover valor de NumSect a cx.

mov dx,Start → Mover valor de Start a dx.

lds bx,Buffer → Preparar para lectura.

int 25h → Ejecutar Interrupción 25h para lectura absoluta.

pop bx → Quitar de la pila valor de bx.

pop ds → Quitar de la pila valor de ds.

pop dx → Recuperar registros.

pop cx → Recuperar registros.

pop bx → Recuperar registros

pop bp → Recuperar registros.

ret ArgLen → Directiva de retorno.

_AbsoluteRead ENDP → Fin del procedimiento _AbsoluteRead

AbsoluteWrite:

Transfiere el contenido del búfer especificado a uno o más sectores de disco, accesando directamente a los sectores lógicos. En caso de error, se enciende el bit de acarreo y AX contiene el código de error.

_AbsoluteWrite PROC NEAR

ARG Buffer:DWORD,Start:WORD,NumSect:WORD,Drive:BYTE= ArgLen à Inicializar búfer.

push bp → Almacenar en pila.

mov bp,sp → Almacenar valor bp.

push bx → Permitir acceso a los argumentos.

push cx → Salvar registros.

push dx → Salvar registros.

push ds → Salvar registros.

mov al,Drive → Mover valor de Drive a al.

mov cx,NumSect → Mover valor de NumSect a cx.

mov dx,Start → Mover valor de Start a dx.

lds bx,Buffer → Preparar para escritura

int 26h → Ejecutar Interrupción 26h para escritura absoluta.

pop bx → Quitar de la pila valor de bx.

pop ds → Quitar de la pila valor de ds.

pop dx → Recuperar registros.

pop cx → Recuperar registros.

pop bx → Recuperar registros

pop bp → Recuperar registros.

ret ArgLen → Directiva de retorno.

_AbsoluteWrite ENDP → Fin del procedimiento _AbsoluteWrite

FreeDiskSpace:

Devuelve en DX:AX el espacio libre en disco, expresado en Kilobytes. En caso de error, se enciende el bit de acarreo.

_FreeDiskSpace PROC NEAR

ARG Drive:BYTE= ArgLen → Inicializar.

push bp → Mover bp a pila.

mov bp,sp → Salvar bp.

push bx → Permitir acceso a los argumentos.

push cx → Salvar registros.

mov ah,36h → Cargar servicio 36h al registro ah.

mov dl,Drive → Preparar para ejecución.

int 21h → Ejecutar servicio 36h de la interrupción 21h.

mul cx → Multiplicar por el valor en cx.

mov cx,1024 → Asignar 1024 a cx.

div cx → Dividir entre cx para obtener el resultado en Kb.

mul bx → Obtener espacio libre.

pop cx → Quitar cx de la pila.

pop bx → Recuperar registros.

pop bp → Recuperar registros.

ret ArgLen → Directiva de retorno.

_FreeDiskSpace → Fin del procedimiento.

4.3 Programación del puerto serial.

En lenguaje ensamblador se puede utilizar el puerto serial para el intercambio de datos, para lograrlo, se puede acudir a la interrupción 14H de la ROM-BIOS para configurar, leer, escribir o simplemente para conocer el estado del puerto; cada una de estas cuatro opciones es un servicio de la interrupción, y se seleccionan a través del registro AH.

Servicio	Descripción
00	Inicializar puerto serie
01	Enviar un dato
02	Recibir un dato
03	Obtener el estado del puerto

En todos los casos, el registro DX debe contener el número del puerto serie; el primero de ellos, COM1 se especifica como 00h.

Para configurar o inicializar el puerto serie, bastará con utilizar el servicio 00 de la interrupción, colocando en el registro a los valores equivalentes a los parámetros. Con este método es posible obtener frecuencias de transmisión que van desde los 110 hasta los 9600 baudios.

Bit 7 6 5 4 3 2 1 0	Utilización
X X X	Relación de baudios
. . . X X . . .	Paridad
. . . . . X . .	Bits de parada
. . . . . X X	Tamaño del caracter

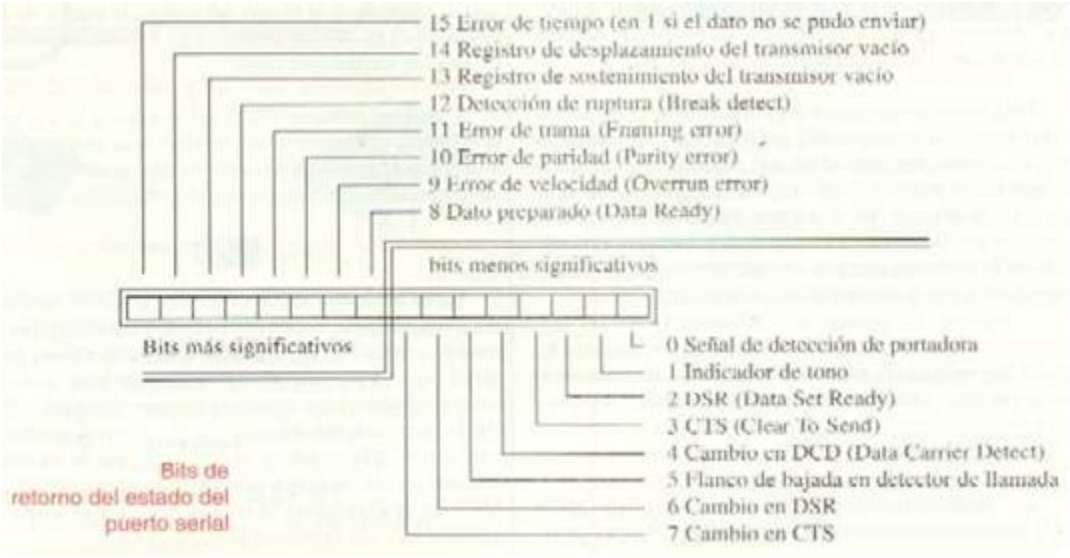
Paridad	
Bits	Significado
00	Ninguna
01	Paridad impar
10	Ninguna
11	Paridad par

Relación de baudios	
Bits	Velocidad
000	110
001	150
010	300
011	600
100	1200
101	2400
110	4800
111	9600

Tamaño del caracter	
Bits	Significado
00	No se utiliza
01	No se utiliza
10	7 bits
11	8 bits

Bits de parada	
Bits	Significado
0	Uno
1	Dos

Para utilizar correctamente el puerto serial, es necesario conocer el significado de cada uno de sus bits.



Ejemplo:

En este ejemplo se utiliza el lenguaje ensamblador para configurar el puerto serial para 1200 baudios, sin bit de paridad, sin bit de parada y 8 bits, realizando el siguiente programa:

mov ah,0 → Servicio 00 de INT 14h: inicializa el puerto

mov a1,83 → configuración: 1200 baudios, no paridad, sin bit de parada, 8 bits de datos

mov dx,00 → Selecciona el puerto COM1

int 14 → Interrupción del BIOS

mov ah,4c → Servicio 4ch de INT 21h: terminar

int 21 → Interrupción servicio del DOS

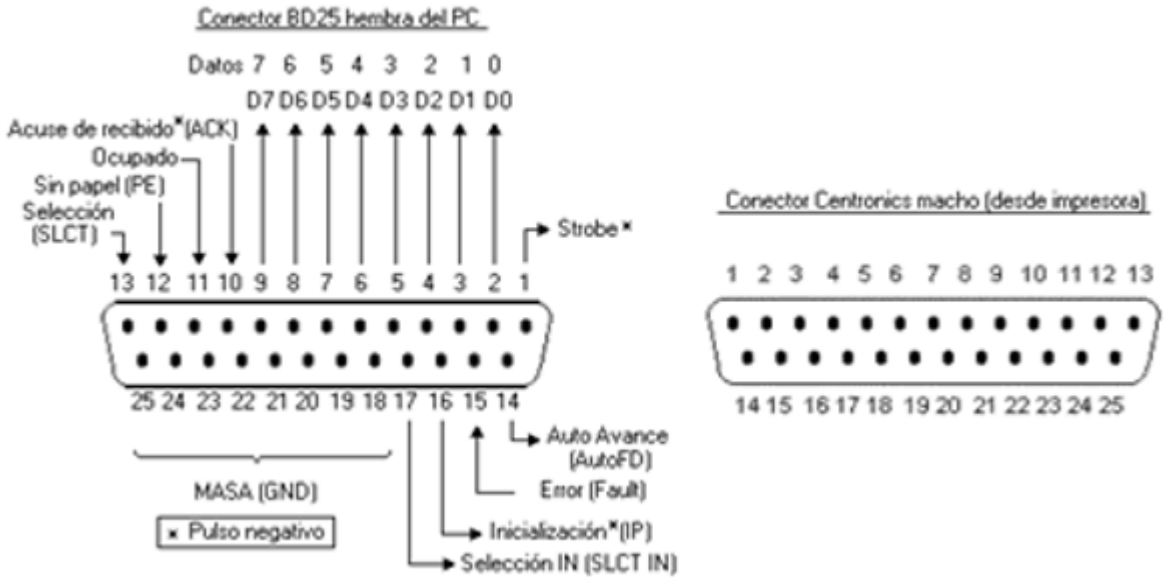
4.4 Programación del puerto paralelo.

Es necesario saber que antes de utilizarlo, que el puerto paralelo se compone de los siguientes tipos de pines:

De estado: Dan información al sistema al igual que el de control.

De datos: Que son los más importantes, ya que por estos sale información crucial, la cual es la que se utiliza para mostrar las salidas de datos.

Cada uno de los pines del puerto paralelo cuenta con un identificador y una función.



En lenguaje ensamblador, se puede leer un dato del puerto mediante la instrucción IN ó escribir un dato en el puerto con la instrucción OUT , en ambos casos el registro AL debe participar activamente en la instrucción, bien sea como fuente (en operaciones de escritura) o destino (en operaciones de lectura) del dato, como en los siguientes casos:

out DX, AL → Lleva al puerto DX el contenido del registro AL

in AL,DX → Lleva al registro AL, el contenido del puerto DX

Otra, exigencia, es que el número del puerto sobre el que se va a realizar la transferencia de datos debe estar señalado por el registro DX, a excepción de los casos en los cuales el número del puerto es inferior a 255 (FFh), en cuyo caso la instrucción que lee o escribe puede señalar directamente el puerto.

La utilización del puerto paralelo en lenguaje ensamblador en comparación con otras opciones de entrada y salida de datos es la más sencilla.

4.5 Programación híbrida.

La programación hibrida consiste en mezclar lenguaje ensamblador con lenguajes de alto nivel, creando programas que aprovechen las ventajas de cada uno de estos, con el fin de que sean más rápidos y eficientes.

Para el siguiente ejemplo se utilizará lenguaje ensamblador y Turbo Pascal, el cual tiene compatibilidad directa con Turbo Ensamblador.

Turbo Pascal permite escribir procedimientos y funciones en código ensamblador e incluirlas como parte de programas escritos en Pascal, para lo cual utiliza dos palabras reservadas: Assembler y Asm.

Assembler permite indicarle a Turbo Pascal que la rutina o procedimiento que se está escribiendo está escrita en código ensamblador.

Ejemplo:

Procedure Limpia_Pantalla; → Declaracion del procedimiento en Pascal.

Assembler; → Instrucción que indica que el código estará

escrito en ensamblador.

AsmMov AX,0600h	→	Mover 0600h al registro AX.
Int 10h	→	Iniciar servicio 06h de la interrupción 10h, es decir, limpiar pantalla.
End	→	Fin del procedimiento.

El código del ejemplo anterior está escrito para Turbo Pascal, a pesar de que parece ser un programa de ensamblador.

4.6 Programación de puerto USB.

El Bus Universal en Serie o USB por sus siglas en inglés, es un bus estándar industrial que define los cables, conectores y protocolos utilizados en bus para conectar, comunicar y proveer de alimentación eléctrica a dispositivos y periféricos.

Fue creado por las empresas que buscaban unificar la forma de conectar periféricos a los equipos, y aunque su versión 1.0 se publicó en 1996, no fue sino hasta 1998 con la especificación 1.1 que se comenzó a utilizar de forma masiva.

La utilización del puerto USB en ensamblador es con los objetivos principales de reorientar la utilización de los periféricos para que tengan mejor rendimiento y utilizar al máximo sus capacidades, así como poder crear nuevos periféricos.

Referencias.

https://brandon22esquivel.wixsite.com/misitio/post/unidad-4-programaci%C3%B3n-de-dispositivos