

Laboratorio de arquitectura de
computadores y ensambladores 1, Sección N.



PRACTICA 2

MANUAL TECNICO

FECHA: 23/06/2023

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación de sistema de punto de ventas en ensamblador, diseñada a través del ensamblador masm611. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

Requisitos Mínimos del Sistema

Sistema operativo 64 bits

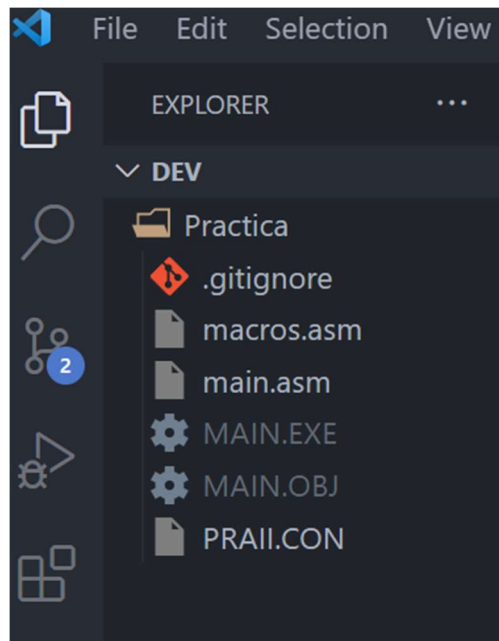
- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

Software (Indispensable tenerlo instalado)

- IDE: Visual Studio Code
Puede conseguirse en:
<https://code.visualstudio.com/>
- MS-DOS (DOS-BOX)
Puede conseguirse en:
<https://www.dosbox.com/download.php?main=1>
- MASM611
Puede conseguirse en:
<https://sourceforge.net/projects/masm611/>

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:



Directorio de la App: Raíz

El archivo main.asm:

Contiene el método principal de nuestra aplicación y controlara todos los métodos y la implementación de la interfaz.

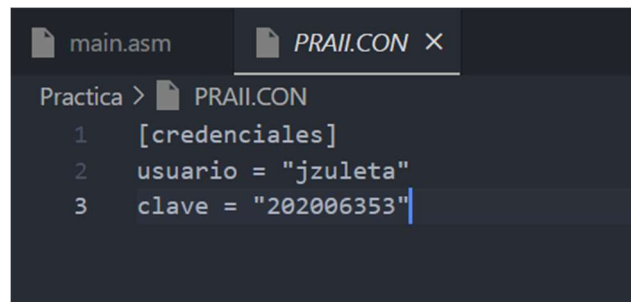
Los métodos y procedimientos utilizados son los siguientes:

Inicio:

Primero se limpia la pantalla y se despliega el mensaje inicial previamente declarado en la sección data:

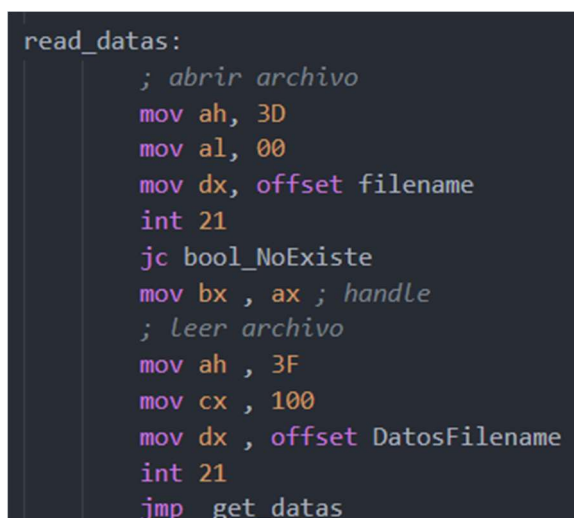
```
; Mensajes de bienvenida
minfo db 0a,"Universidad de San Carlos de Guatemala"
      db 0a,"Facultad de Ingenieria"
      db 0a,"Escuela de Vacaciones"
      db 0a,"Arquitectura de Computadoras y Ensambladores 1"
      db 0a,"Nombre: Juan Josue Zuleta Beb"
      db 0a,"Carne: 202006353"
```

Por consiguiente, se procede a leer el archivo PRAII.CON de credenciales el cual contiene la clave y el usuario:



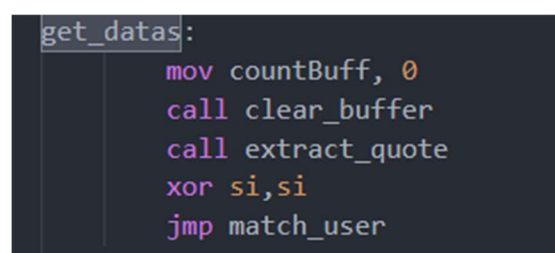
```
main.asm  PRAII.CON X
Practica > PRAII.CON
1  [credenciales]
2  usuario = "jzuleta"
3  clave = "202006353"
```

Una vez comparados con el método leer archivo se comprueba la existencia de contenido dentro del archivo



```
read_datos:
    ; abrir archivo
    mov ah, 3D
    mov al, 00
    mov dx, offset filename
    int 21
    jc bool_NoExiste
    mov bx, ax ; handle
    ; leer archivo
    mov ah, 3F
    mov cx, 100
    mov dx, offset DatosFilename
    int 21
    jmp get_datos
```

Luego se obtienen los datos por medio de la función obtener datos para lo cual se utiliza un buffer y se llama la función extraer comillas para encontrar los datos



```
get_datos:
    mov countBuff, 0
    call clear_buffer
    call extract_quote
    xor si, si
    jmp match_user
```

Luego se comprueba en base a variables previamente declaradas se realiza una comparación con los datos para saber si existe coincidencia, para ello se utiliza la función match user y match key

<pre> match_user: mov bh, outBuff[si] mov bl, identifier[si] cmp bh,bl jnz NoIgual cmp bh , "\$" jz key inc si jmp match_user </pre>	<pre> match_key: mov bh, outBuff[si] mov bl, password[si] cmp bh,bl jnz NoIgual cmp bh , "\$" jz Login inc si jmp match_key </pre>
--	--

Posterior a ello se manda a llamar el método login una vez que se ha comprobado la coincidencia de datos en ambos casos y esto genera un texto de bienvenida y espera la respuesta del usuario para luego mandar a llamar al método de menú principal

```

Login:
    mov ah, 3E
    mov bx, ax
    int 21
    PrintText empty_line_txt
    PrintText empty_line_txt
    PrintText logged_txt
    PrintText press_enter_txt
    EnterOption
    jmp Main_Menu

```

El menú principal contiene todas las opciones principales de la aplicación, las entradas se leen por medio de una entrada de texto para lo cual se espera un numero el cual corresponde al código ASCII del 1 al 4.

```

Main_Menu:
    ClearScreen
    PrintText mainmenu
    PrintText opt_txt
    ExtractOption
    cmp al , 31
    je Prod_Menu
    cmp al , 32
    je sold_Menu
    cmp al , 33
    je tool_menu
    cmp al , 34
    je Exit
    PrintText opt_error_txt
    jmp Main_Menu

```

Empezaremos con el menú de productos para lo cual tenemos distintas opciones que funcionan de igual forma que el menú principal con números correspondientes al código ASCII y con comparaciones y saltos básicos.

```
Prod_Menu:
    PrintText menuprod
    PrintText opt_txt
    ExtractOption
    cmp al , 31
    je mk_prod
    cmp al , 32
    je dl_prod
    cmp al , 33
    je sw_prod
    cmp al , 34
    je Main_Menu
    jmp Prod_Menu
```

La función make product se encarga de solicitar al usuario diferentes entradas las cuales pueden ser en orden según se lo requiera, primero el código del producto el cual debe cumplir con las limitaciones solicitadas de lo contrario se seguirá pidiendo hasta que el usuario logre ingresar uno que si cumpla, segundo se le pedirá la descripción del producto y por tercer paso se le pide al usuario que ingrese el precio del producto y por ultimo que ingrese las unidades del producto que se almacenaran.

```
make_prod proc
insert_product:
    PrintText code_prod
    mov dx , offset buffer_entrada
    mov ah , 0a
    int 21
    mov DI, offset buffer_entrada
    inc DI
    mov AL , [DI]
    cmp AL , 05
    jnb g_params
    jmp insert_product
g_params:
    mov SI, offset cod_prod
    mov DI, offset buffer_entrada
    inc DI
    mov CH, 00
    mov CL, [DI]
    inc DI
    jmp cp_name
cp_name:
    mov AL , [DI]
    mov [SI], AL
    inc SI
    inc DI
    loop cp_name
    jmp trns_code
```

La función show product se encarga de mostrar los productos actuales almacenados, para ello se verifican los contadores globales de productos y se consultan sus datos por medio de comparaciones en sus respectivas variables globales

```
shown_prod proc
show_prod:
    call rd_file
    mov [handle_prods], AX
    PrintText empty_line_txt
    mov si, 0
loop_show_p:
    inc si
    mov BX, [handle_prods]
    mov CX, 05
    mov DX, offset cod_prod
    mov AH, 3f
    int 21
    mov BX, [handle_prods]
    mov CX, 21
    mov DX, offset cod_desc
    mov AH, 3f
    int 21
    mov BX, [handle_prods]
    mov CX, 0004
    mov DX, offset num_price
    mov AH, 3f
    int 21
    cmp AX, 00
    je fin_mostrar
```

Por su lado la función delete product se encarga de borrar de la memoria un producto por medio de su código mientras no se encuentre se repetirá el ciclo y una vez encontrado lo borrara, y si no lo encuentra saldrá al menú de productos

```
delete_prod proc
dele_prod:
    mov DX, 0000
    mov [puntero_temp], DX
ask_for_code:
    PrintText code_prod
    mov DX, offset buffer_entrada
    mov AH, 0a
    int 21
    mov DI, offset buffer_entrada
    inc DI
    mov AL, [DI]
    cmp AL, 00
    je ask_for_code
    cmp AL, 05
    jb accept_code_to_delete
    jmp ask_for_code
accept_code_to_delete:
    mov SI, offset cod_prod_temp
    mov DI, offset buffer_entrada
    inc DI
    mov CH, 00
    mov CL, [DI]
    inc DI
```


Para el módulo de ventas se utiliza un contador global para el conteo del precio total de la venta, además se utiliza un método de comparaciones para buscar el producto que se desea comprar y posterior a ello se solicitan las unidades que se desean adquirir, una vez que se parsean los datos y se agregan al contador global del precio de las ventas este proceso se repite hasta que el contador de compras llega a 10 y esto conlleva a que el usuario ha terminado de comprar

```
sold_Menu:
    mov IncItem , 0
    loop_sold:
        cmp IncItem , 0A
        je main_menu
        get_date
        get_hour
        mov DX, 0000
        mov [puntero_temp], DX
        PrintText empty_line_txt
        PrintText solds
        mov AX, [TotalVentas]
        call int_to_stringTotal

        mov BX, 0001
        mov CX, 0005
        mov DX, offset numero
        mov AH, 40
        int 21
```

El archivo macros.asm:

Contiene funciones en forma de macros para ayudar a aligerar el archivo principal, dentro del mismo no se pudieron incluir procedimientos en forma de macros por lo cual se mantuvieron únicamente macros de tipo auxiliar.

Las macros utilizadas son los siguientes:

La macros ClearScreen sirve para limpiar la pantalla en cualquier momento de la ejecución y sirve para darle un mejor aspecto al momento del manejo de la aplicación

```
ClearScreen macro
    mov ah, 0
    mov al, 3
    int 10H
endm
```

La macros PrintText ayuda a imprimir una cadena previamente declarada como una variable de bits en este caso bajo la directiva de radix 16

```
PrintText macro buffer
    mov dx, offset buffer
    mov ah, 09
    int 21
endm
```

La macros EnterOption funciona como buffer de entrada y sirve para reconocer la entrada desde el teclado y al ser un código repetido funciona mejor siendo una macros para facilitar las entradas desde el teclado

```
EnterOption macro
    mov ah, 01
    int 21h
endm
```

La macros get date y get hour funcionan para obtener en tiempo real la fecha y la hora actual y ayuda en la creación de reportes y en la creación de compras para poder registrar la fecha y la hora en que se ha realizado

```
get_date macro
    mov ah, 2a
    int 21h

    xor ah, ah
    mov al, dl
    mov bl, 0a
    div bl
    add al, 30
    add ah, 30
    mov dia[0], al
    mov dia[1], ah
```

```
get_hour macro
    mov ah, 2ch
    int 21h

    xor ah, ah
    mov al, ch
    mov bl, 0a
    div bl
    add al, 30
    add ah, 30
    mov hora[0], al
    mov hora[1], ah
```