Introducció.

L'objectiu d'aquesta pràctica és exercitar l'estructuració del codi i aprofundir en el coneixement de l'arquitectura bàsica del IBM-PC.

En aquesta pràctica començarem a construir 'porcions' de codi en forma de subrutines 'ben definides' amb l'objectiu de re utilitzar-les en pràctiques posteriors. Aquestes subrutines, un cop verificat el seu bon funcionament en programes senzills, passaran a formar part de la nostra 'biblioteca personal de codi'.

Tots els exercicis d'aquesta pràctica es realitzaran entorn a 'zones de memòria', al principi sobre una zona 'genèrica' i acabarem escrivint sobre el 'buffer actiu de video'. D'aquesta forma veurem 'el reflex' (o conseqüència) d'accedir a un perifèric mapejat a memòria.

1- Subrutines.

Podem definir **Subrutina** com 'el conjunt d'instruccions en llenguatge màquina (o assemblador) que realitza una tasca parametrizable'. Aquestes subrutines serveixen per implementar els procediments, funcions i mètodes dels llenguatges d'alt nivell i estructurar el codi de manera 'jeràrquica'.

Gairebé tots els processadors disposen del recursos necessaris (instruccions i estructures) per implementar l'execució no-lineal d'un codi. Aquests recursos són bàsicament instruccions de modificació d'execució (crides i retorns) i estructura de pila (STACK). Això permet escriure un codi 'genèric' que produeixi resultat 'diferents' depenent de 'com sigui executat'. Aquesta 'diferència' respondrà a uns paràmetres ben definits i no a qüestions 'aleatòries' o 'd'atzar'. La primera necessitat és definir de manera clara i precisa els mecanismes que utilitzarem per gestionar les subrutines.

El primer element a considerar és el conjunt de paràmetres que 'es passaran' a la subrutina. Aquesta informació provindrà de les 'variables' del nostre programa (memòria o bé registres) i es transferirà a la subrutina a través de la pila. El pas de paràmetres es pot fer 'per valor' o bé 'per referència'. La diferència entre aquests dos mètodes radica, a part de 'la forma', en les seves conseqüències. Si passem un paràmetre 'per valor' efectuarem una còpia d'aquest valor i el transferirem, mentre que de l'altre manera el que estem transferint és 'una referència' al lloc on es troba emmagatzemat. En el cas que la subrutina modifiqués aquest valor, al retornar, ens trobarem amb situacions diferents depenent de com s'hagi passat. Si s'ha passat per valor, conservarem el valor 'original' (abans d'efectuar la crida) mentre que el pas per referència ens ofereix la possibilitat de mantenir el valor 'modificat'. En cada cas procedirem segons les nostres necessitats.

Un altre element important, que permet introduir el concepte de funció (a diferència del procediment), és el valor de retorn. Una subrutina pot retornar un 'resultat' de la seva execució. Aquest resultat sempre el retornarem a través del registre AX. En cas necessari (per mida) podrem retornar 'una referència' a una estructura de dades.

Les subrutines, per a la seva execució, poden necessitar 'variables locals' (un espai per emmagatzemar resultats intermitjos). Donada la característica 'temporal' d'aquestes dades (no és necessari mantenir ocupat un espai de dades permanentment) s'ubicaran en algun lloc adient.
Per acabar, l'element principal que intervé en la gestió de les subrutines és l'anomenat 'bloc d'activació'. Aquest bloc correspon a tota la informació que es manega durant l'execució d'una subrutina: paràmetres, adreça de retorn, estat del processador, variables locals i resultat. La major part del bloc d'activació es guarda a la pila.

2- Fases de la gestió de les subrutines.

Passar els paràmetres: Consisteix en col·locar els paràmetres en un lloc adequat perquè la subrutina pugui utilitzar-los quan sigui necessari. Considerarem que els paràmetres es passen sempre a través de la pila. L'ordre en que 'empilarem' aquests paràmetres serà el contrari a l'ordre en que estan escrits. Un paràmetre per valor es passa col·locant una còpia d'aquest al cim de la pila. Un paràmetre 'per referència' es passa col·locant l'adreça al cim de la pila.

Cridar a la subrutina: Consisteix en guardar l'adreça de retorn a la pila i passar a executar la primera instrucció de la subrutina. Tot això es fa quan s'executa la instrucció 'CALL'.

Establir el punter al bloc d'activació (o establir l'enllaç dinàmic): Consisteix en inicialitzar un registre (s'utilitza el 'Frame Pointer' o BP) perquè apunti a una posició fixa dins el bloc d'activació.

Crear les variables locals: Consisteix en reservar a la pila l'espai necessari per a les variables locals a la subrutina i, si cal, inicialitzar-les. L'ordre en que queden les variables locals a la pila és el mateix amb el que han estat declarades.

Salvar l'estat: Consisteix en guardar a la pila tots els registres que es modifiquen en el cos de la subrutina.

Executar la subrutina: Els aspectes específics a nivell de subrutines que es poden produir en el cos de la subrutina i que hem de saber realitzar correctament són: accés als paràmetres, accés a les variables locals i accés al resultat. A un paràmetre s'hi accedeix a través del registre BP amb el desplaçament positiu necessari segons el lloc que li correspon dins el bloc d'activació. A una variable local s'hi accedeix a través del registre BP amb el desplaçament negatiu adequat. El resultat d'una subrutina sempre es col·loca en el registre AX (o bé en AL si la mida del resultat fos de d'1 byte).

Restaurar l'estat: Consisteix en recuperar el valor que tenien els registres que s'hagin modificat al llarg de la subrutina.

Eliminar les variables locals: Consisteix en alliberar l'espai que ocupaven les variables locals dins la pila.

Restaurar el punter al bloc d'activació: Consisteix en recuperar el valor del registre BP que hi havia a l'inici de la subrutina.

Retornar de la subrutina: Consisteix en saltar a la instrucció següent a la de la crida a la subrutina. Això es fa quan s'executa la instrucció RET.

Eliminar els paràmetres: Consisteix en treure de la pila tots els paràmetres que s'hagin passat.

Recollir el resultat: Consisteix en agafar el resultat que ha deixat la subrutina en el registre AX (o bé en AL, segons la seva mida).

Un cop comentades totes les fases de gestió d'una subrutina us exposem un codi 'auto explicatiu' d'un programa senzill que il·lustra la implementació i la crida a una subrutina.

```
.model large                    ;definim el model de memòria emprat
```

```
.386                                    ;l'arquitectura sobre la que treballem
.stack          100h                    ;segment de pila, 'reservem' 256 bytes
.data                                   ;Inici del segment de dades
valors          DW      123,251 ; dues variables inicialitzades tipus Word
resultat        DD      ?               ; espai reservat per al resultat (Double Word)

.code           ;inici del codi

producte        PROC NEAR       ;punt d'entrada de la subrutina producte
                ;recull dos paràmetres tipus 'byte' i en retorna el producte en un 'word'
                ;declarat 'PROCediment proper' (NEAR), només s'empila l'offset en la crida
                ;s'enllaçarà en el mateix segment de codi que el prog. Pral.
;establim el punter al bloc d'activació
        PUSH            BP
        MOV             BP,SP
;no calen variables locals, en cas contrari: SUB  SP,mida
;salvem l'estat (els registres que farem servir)
        PUSH            DX      ;no caldrà però el deixo per claredat

;en aquests moments, la pila conté      SP ->   |       DL      |       (BP-2)
;i els registres BP i SP apunten...              |       DH      |
;                                       BP ->   |       BPanteriorL    |
;                                               |       BPanteriorH    |
;                                               |       @RetornL       |       (BP+2)
;                                               |       @RetornH       |
;                                               |       param1L        |       (BP+4)
;'l'ample' és d'un byte                         |       param1H        |
;                                               |       param2L        |       (BP+6)
;'l'ample' és d'un byte                         |       param2H        |


;cos de la subrutina, per executar MUL cal tenir un operand a AX/AL i l'altre en un registre
; o bé a memòria
        MOV     AX, 4[BP]       ;recullo el primer paràmetre de 'SS:BP+4'
        MUL     5[BP]           ;multiplico pel segon paràmetre
;el resultat queda  a DX:AX,  multipliquem 'words'
; per això no cal moure sobre AX el resultat
;restauro l'estat (salvat anteriorment)
        POP     DX
;no cal eliminar variables locals (no n'hem declarat).Si calés, 'MOV SP,BP'.
;Restaurar el punter anterior al bloc d'activació
        POP     BP
;retornem de la subrutina
        RET
producte ENDP   ;fi de la declaració del procediment

.startup  ;punt d'entrada del programa, inicialització dels registres de segment
;crida a 'FUNCIO producte(param1 TIPUS byte, param2 TIPUS byte) RETORNA 'word'
;passar paràmetres en ordre invers
        PUSH            valor+2         ;segon paràmetre
        PUSH            valor           ;primer paràmetre
;crida a la funció
        CALL    producte
;eliminem paràmetres (2 bytes)
        ADD     SP,4
;recollim el valor de retorn i el desem
        MOV     resultat, AX
.exit                                   ;inclusió del codi de retorn al sistema operatiu
END
```

Enunciat de la pràctica.

Implementeu les següents accions/funcions i verifiqueu-ne el seu correcte funcionament amb el depurador:

- **ACCIO escriu (paraula TIPUS word, adreça TIPUS word).**

Aquest procediment copia una paraula de 16 bits en una posició de memòria indicada per 'DS:adreça'. Per tal de verificar el correcte funcionament utilitzeu el següent programa:

```
PROGRAMA verifica_escriu
        VAR
                missatge:='HHoollaa  qquuee  ttaall',0,0
                destí TIPUS CADENA[30] bytes
                paraula TIPUS word
                índex TIPUS byte
        FVAR
        índex:=0
        paraula:=missatge[índex]
        REPETIR
                escriu(paraula,destí[índex])
                índex:=índex+1
                paraula:=missatge[índex]
        FINS    paraula=0
FPROGRAMA
```

- **FUNCIO posició (x TIPUS byte, y TIPUS byte) RETORNA word**

Aquesta funció retorna el valor corresponent a la operació matemàtica 2*(x+(80*y)). Verifiqueu el seu funcionament amb:

```
PROGRAMA verifica_posició
        ...
        x:=10
        y:=20
        posició(x,y)
        x:=7
        y:=3
        posició(x,y)
FPROGRAMA
```

FUNCIO base () RETORNA word.

Aquesta funció retorna 'B000h' si a la posició de memòria '0000:0449h' hi ha un '7' (byte) o bé 'B800h' si en aquesta posició hi ha un '2' o bé un '3' (byte). Per tal de fer l'adreçament al segment '0000' cal que us 'recolzeu' en un registre de segment (per exemple ES) i feu 'MOV AL, ES:0449' ;amb 0000 carregat a ES. Verifiqueu el seu funcionament depurant:

```
PROGRAMA verifica_base
        base()
FPROGRAMA
```

Un cop enteses, codificades i verificades aquestes subrutines us serà molt fàcil adaptar-les per tal de implementar funcions del tipus:

- **ACCIO copia_cadena(adreça origen, adreça destí)**, que es basa en la 'iteració' de **ACCIO copia_caràcter(caràcter, adreça destí)** 'fins que' caràcter=0 (incrementant l'adreça).

Feu un programa que 'copiï una cadena de text' (variables caràcter acabada amb 0) començant en l'adreça 'base()+posició(x,y)' i incrementant aquesta adreça de dos en dos (per cada caràcter) corresponent al següent codi:

```
PROGRAMA pantalla
        VAR
```

```
            text:='Hola que tal',0
            posició TIPUS word
            índex    TIPUS byte
        FVAR
        posició:=base()+posició(3,2)
        índex:=0
        REPETIR
            copia_caràcter(text[índex],posició)
            índex:=índex+1
            posició:=posició+2
        FINS text[índex]=0
FPROGRAMA
```

Amb aquest codi funcionant correctament us adonareu que la pantalla (de text de 80X25) no és res més que 'el reflex codificat en ASCII' d'una zona de memòria a la que hi podeu accedir lliurement. No oblideu que sou vosaltres (els programadors) els qui 'porteu el timó de la nau' si creieu convenient passar un paràmetre per referència no dubteu a fer-ho.

Per acabar només cal parar atenció al byte 'que anem saltant' (posicions senars) cada cop. Aquest byte indica 'l'atribut' amb el que serà mostrat aquest caràcter. Dins la memòria hi ha una parella de 'caràcter,atribut' per a cada posició de pantalla. Aquest atribut, fa referència al color de la lletra i el fons. Normalment (lletra blanca sobre fons negre) aquest atribut és '07h'. El significat de cadascun dels bits del atribut és molt senzill. Per al fons es fa servir el 'nibble' (4bits) alt i pel caràcter el 'nibble' baix. Els tres bits de menys pes de cadascun dels nibbles indiquen la combinació RGB (els colors són formats a partir de les components R de vermell, G de verd i B de blau). Experimenteu amb diferents atributs i podreu fer 'pantalles del vostre equip de futbol' o bé 'festes de marcianets'....

Recomanacions:

Escriure programes en assemblador no és una tasca 'gratificant' per si mateixa. Sovint els nostres 'vicis' (la pressa, el desordre, la manca de planificació, la pròpia ignorància, la documentació insuficient...) ens aboquen a cometre errors que impliquen una gran inversió en temps per a la seva localització i correcció. D'altrabanda, 'barallar-s'hi', ens dóna una visió en profunditat 'del que tenim a les mans' (valorem el cost real dels esquemes algorísmics, coneixem les possibilitats i dificultats de resoldre problemes amb un computador i comprenem el veritable funcionament d'aquest).

Sigueu meticulosos en la traducció dels algorismes a codi. Comenteu àmpliament el codi (incloent-hi capçaleres en la implementació de les subrutines). Aneu 'pas a pas', sense voler córrer, avançant etapes consolidades.

Porteu les pràctiques “preparades” (llegides, enteses i potser “picades”) i proveïu-vos de la documentació necessària (és difícil recordar tot el repertori d’instruccions).