TP2 – Manipulation de la mémoire

## Objectif du TP :

Le but de ce TP est de s’approprier les concepts d’accès aux zones mémoire que ça soit via l’émulateur mais également par programme pour l’accès aux données liés à l’architecture d’un ordinateur parmi lesquels :

Exercice n° 1 : Exploitation de la mémoire : exemple de programme

PILE SEGMENT STACK

DW 256 DUP(?)

Base:

PILE ENDS

DATA SEGMENT

chaine DB 10,13,'Hello',13,10,'$' ; la chaine a afficher doit se terminer par '$'.

;les caracteres 10,13 et 13,10 affichent des retour a la ligne

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA,SS:PILE

main:

; Initialisation du registre DS avec valeur de segment de données

MOV AX,DATA ;Positionnement de DS

MOV DS,AX ; dans le segment de données

; Initialisation du registre SS avec valeur de segment de Pile

MOV AX,PILE

MOV SS,AX

; Appel à l’interruption 21h , 09h pour réaliser l’affichage

MOV AX,0

MOV DX, OFFSET Chaine ; Met dans DX l'adresse de la chaine a afficher

MOV AH,09H ; affichage de la chaine de caracteres

INT 21H ; dont l'adresse est dans DX

; Appel à l’interruption 21h 4Ch pour pour retour au système

MOV AH,4CH

INT 21H

CODE ENDS

END main

1. Repérez les différents segments de ce code.
2. Avez-vous une idée de la taille des différents segments ?
3. Récupérez le fichier *SampleAffichage.asm* qui contient le code ci-dessus Compilez le sous emu68086 et allez voir le fichier généré par le compilateur qui contient la table des symboles. Que comprenez-vous de cette table ?
4. Vous allez maintenant charger le code compilé sous l’émulateur.

* Quelle est la valeur du segment de CODE ?
* Quelle est la valeur du segment de DATA ?
* Quelle est la valeur du segment de PILE ?
* Qu’en déduisez-vous par rapport au relogement de code fait par le MMU et la valeur de son registre de translation ?
* Vérifiez dans l’émulateur la présence des données et du code dans les bonnes zones mémoires.
* Une adresse physique est exprimée sous la forme aaaa:bbbb en hexadécimal où aaaa est le segment et bbbb est l’offset à l’intérieur du segment. La valeur de l’adresse physique sur 20 bits est aaaa0h+bbbb ou encore aaaa\*16+bbbb.

Vérifiez la cohérence des adresses de l’implantation du code telle avez celles trouvées dans l’émulateur

* Expliquez d’où vient ce calcul

1. Vous allez maintenant exécuter le code.

*Vous constaterez que ce code affiche la chaîne Hello grâce à l’utilisation de l’interruption 21h en ayant préalablement choisit la fonction 09h qui permet d’afficher une chaîne dont l’adresse est dans DX.*

*Nous utiliserons par la suite cette interruption qui nous permettra d’afficher des éléments d’un tableau. Mais dans ce cas nous utiliserons la fonction 02h qui permet l’affichage caractère.*

Exercice 2 : Manipulation des différents types d’adressage

Le but de l’exercice est d’utiliser les différents types d’adressage : basé, indexé pour manipuler cette chaîne et afficher sa valeur.

En annexe, vous sont donnés quelques éléments concernant les déclarations de variables ainsi que les différents modes d’adressage.

Vous allez maintenant écrire le code qui affiche un tableau de caractères à l’écran.

1. Vous partirez de *squelette.asm* disponible dans le répertoire du TP2 et qui contient l’essentiel du code nécessaire. Que vous exécuter pour comprendre le fonctionnement de l’INT 21 concernant l’affichage.
2. Vous déclarerez une chaîne de caractère que vous initialiserez selon votre choix.

Vous écrirez le code, qui permet, grâce à une boucle d’afficher le tableau caractère par caractère.

Exercice n° 3 : boucle simple en utilisant des instructions de saut

Regardez ces deux programmes simples. Que font-ils ?

|  |  |
| --- | --- |
| MOV AX,0  MOV CX,10  iter:  INC AX  LOOP iter | MOV AX, 0  MOV CX, 10  iter:  INC AX  DEC CX  CMP CX, 0  JNE iter |

Ecrivez ces deux programmes, et lancez emu8086. Regardez pour chaque programme l’évolution des registres AX et CX. Faites une exécution Pas à Pas pour cela.

* Vous irez voir sur internet la signification de l’instruction JNE. Et vous expliquerez quels registres sont utilisés par cette instruction.
* Quelles sont les valeurs des registres AX et CX à la fin de l’exécution des deux

programmes ?

* Que pouvez-vous en conclure ?
* Réaliser un programme qui affiche 10 fois la chaîne de caractères ‘Hello’. Vous

utiliserez pour cela une boucle en initialisant au début du programme le registre CX à 10.

3- Annexes sur la syntaxe assembleur

**Déclaration de variables**

Une variable en assembleur dans le segment de données se définit de la manière suivante :

<Nom> <Type> <Valeur>

<Type> correspond à l’espace mémoire à allouer à la variable <Nom> :

DB 1 octet

DW 2 octets = 1 mot

<Valeur> correspond à la valeur initiale de la variable <Nom>.

Exemple de déclaration de variables initialisées :

UnOCTET DB 1 ; Variable=1

MOT DW ? ; Variable non initialisée

Exemple de déclaration de **Vecteurs et Tableaux**

VO DB 50 DUP(?) ; *50 octets non initialises*

VM DW 10 DUP (0); *10 mots a zéro*

Exemple de déclaration de **Message ou chaîne de caractères**

MESS DB 'Bonjour' ;*Chaîne de caractères*

CONST DW 7,6,12h,11B ; *Tableau de 4 mots constants*

Exemple de déclaration de **constante**

Constante EQU 10 ; Constante =10

**Manipulation de boucles**

Exemple d’itération d’une séquence d’instructions.

MOV CX, 10 ;*Initialisation du compteur de boucle*

;*obligatoirement dans CX*

repeter :

! *Séquence d’instructions*

Loop repeter

**Modes d’adressage**

**Adressage Immédiat** : on manipule la valeur directement

CMP AX, 3412H

**Adressage direct** : la donnée est spécifiée par son identificateur, on fait un accès mémoire

MOV AX, BX ; on dit aussi adressage par registre dans ce cas-là

**Adressage indirect** : le registre BX contient l’adresse de l’emplacement mémoire à accéder. On

fait deux accès mémoire

MOV BX,OFFSET VAR

MOV AX, [BX]

**Adressage basé** : On ajoute un déplacement à la valeur contenue dans le registre

MOV AX, [BP+2]

**Adressage indexé** : Utilise les registres d’index SI ou DI. Utile pour les tableaux.

ADD AX, TAB[SI]

**Adressage basé indexé** : Combinaison des modes précédents

ADD AX, TAB[BX + SI]