INTRODUCTION

I – Sujet

Le projet réalisé cette année doit pouvoir lire un fichier incluant des instructions d'ordre graphique, interpréter ces instructions, puis dessiner la figure correspondante.

II – Outils

Le projet est réalisé entièrement en assembleur grâce aux programmes emu8086 et MASM.
Emu8086 permet d'écrire le programme, MASM sert à créer les .obj nécessaires à la réalisation d'un exécutable (.exe).

III – Composition du projet

Le projet est découpé en 4 étapes.

- Chaque étape possède une grammaire bien spécifique.
- Chaque étape doit être découpée en 3 fichiers assembleurs.

IV – Aide

Nous avons à disposition une bibliothèque fileio.obj, cependant dans notre projet nous avons créé notre propre bibliothèque.

V – Condition de rendu

Le projet doit être réalisé en binôme ou en trinôme appartenant au même groupe de TP.
Le projet doit être rendu par mail avant la 10^{e} séance de TP.

Le mail doit être un .rar ou tar.gz et son extraction doit se faire dans un répertoire portant les noms des auteurs.
Le répertoire sera constitué d'autres répertoires nommé « EtapeX » ou X correspond à chaque étape du projet.
Chaque étape sera constituée de la manière suivante :

- Un fichier « bin » : on trouvera dans se fichier l'exécutable du projet.
- Un fichier « objet » : qui comportera tous les .obj du projet.
- Un fichier « source » : comportant les fichiers assembleur.
- Un fichier « test » : comportant les fichiers textes de test.

ETAPE 1

L'étape 1 doit pouvoir lire un fichier comportant la grammaire suivante :

Instruction = *Type Espace Ligne Espace Colonne Espace Couleur Espace Longueur*
Type = *'H' | 'B'| 'G' | 'D'*
Ligne = *Valeur*
Colonne = *Valeur*
Longueur = *Valeur*
Couleur = *DigitHex*
Valeur = *Digit Digit Digit*
Espace = *' '*
Digit = *'1' | '2' | '3' | '4' | '5'| '6' | '7' | '8' | '9'*
DigitHex = *'1' | '2' | '3' | '4' | '5'| '6' | '7' | '8' | '9' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'*

Notre programme est découpé en 3 fichiers assembleurs : projet.asm, libdessin.asm et fileio.asm.

I – Projet.asm « programme principal »

Voici comment nous avons découpé notre programme :

- Passage en mode vidéo « InitGraphe »
- Ouverture du fichier « openFile »
- Lecture du fichier « readFile »
- Analyse et stockage des valeurs du fichier
- Analyse du « type » dans l'instruction
- Faire la fonction correspondante au « type » « DR », « BA », « GA », « HA »
- Fermeture du fichier « closeFile »
- Fin programme

Légende :
En bleu : procédure externe dans fileio.asm
En orange : procédure externe dans libdessin.asm

Analyse et stockage des valeurs du fichier :
Pour pouvoir stocker nos valeurs contenu dans le fichier texte nous disposons de plusieurs variables.
Comme nous connaissons exactement la grammaire de l'instruction il est facile de stocker les valeurs. En effet grâce à la procédure de lecture nous disposons d'un tableau (buffer) qui contient à chacune de ses cases le contenu du fichier.
Pour stocker les valeurs dans nos variables il suffit donc de se déplacer dans le tableau et de stocker la valeur.

Analyse du « type » dans l'instruction :
Après avoir stocké les valeurs du fichier dans nos variables, nous comparons la variable contenant le « type » de l'instruction au choix des « type » attendu.

II – fileio.asm

Ce fichier assembleur comprend les procédures dont nous avons besoin dans notre programme principal, hormis les procédures relatives au dessin.

Voici les procédures externe que contient se fichier assembleur :

```
PUBLIC openFile     ; ouverture du fichier
PUBLIC readFile     ; lecture du fichier
PUBLIC closeFile    ; fermeture du fichier
```

III – libdessin.asm

Ce fichier assembleur comprend les procédures dont nous avons besoin dans notre programme principal gérant le dessin.

Voici les procédures externe que contient se fichier assembleur :

```
PUBLIC  InitGraphe      ; passage en mode vidéo
PUBLIC  DR              ; dessine un trait vers la droite
PUBLIC  BA              ; dessine un trait vers le bas
PUBLIC  HA              ; dessine un trait vers le haut
PUBLIC  GA              ; dessine un trait vers la gauche
```

VI – Compilation

Pour pouvoir lancer le programme, il suffit d'écrire une instruction dans le fichier texte nommé « text.txt » et lancer l'exécutable nommé « projet.exe ».

ATTENTION : pour que l'exécutable puisse ouvrir le fichier texte il faut que celui-ci ce trouve dans le même répertoire.

Vous avez à disposition plusieurs instructions déjà écrite que vous pouvez tester.
Ces instructions se trouvent dans le répertoire « test ».

ETAPE 2

L'étape 2 doit pouvoir lire un fichier comportant la grammaire suivante :

Programme = Instruction | Instruction NL Programme ***NL*** = nouvelle ligne

La grammaire de l'étape 2 est donc identique à la grammaire de l'étape précédente sauf que cette fois ci nous pouvons avoir plusieurs instructions, chaque nouvelle instruction étant sur une nouvelle ligne.

Comme nous avons vu dans l'étape précédente, notre programme est découpé en 3 fichiers assembleur.
Cependant, comme les 2 bibliothèques « fileio.asm » et « libdessin.asm » ne sont pas modifiées, nous n'allons détailler que le programme principal.

I – Projet.asm « programme principal »

Voici comment nous avons découpé notre programme :

- Passage en mode vidéo « InitGraphe »
- Ouverture du fichier « openFile »
- Etiquette de boucle
- Lecture du fichier « readFile »
- Analyse et stockage des valeurs du fichier
- Réinitialisation du buffer
- Analyse du « type » dans l'instruction
- Faire la fonction correspondante au « type » « DR », « BA », « GA », « HA »
- Saut à l'étiquette de boucle
- Fermeture du fichier « closeFile »
- Fin programme

Légende :
En bleu : procédure externe dans fileio.asm
En orange : procédure externe dans libdessin.asm
En rouge : modification apportée par rapport a l'étape 1

Les modifications apportées permettent de pouvoir lire ligne par ligne le contenu du fichier puis de traiter l'instruction.

II – Compilation

Même procédure que l'étape 1, sauf que nous pouvons mettre plusieurs instructions dans le fichier texte.

Etape 3

L'étape 3 doit pouvoir lire un fichier comportant la grammaire suivante :

Programme_Logo *= Init NL List_Instructions*
Init *= 'I' Ligne Espace Colonne Espace Couleur*
List_Instructions *= Instruction_Logo | Instruction_Logo NL List_Instructions*
Instruction_Logo *= Instruction_Couleur_Logo | Instruction_Deplacement_Logo | Instruction_Rotation_Logo*
Instruction_Deplacement_Logo *= Type_Deplacement Espace Longueur*
Instruction_Rotation_Logo *= Type_Rotation Espace Digit*
Instruction_Couleur_Logo *= 'BC' DigitHex | 'LC'*
NL *= nouvelle ligne*
Type_Deplacement *= 'AV' | 'RE'*
Type_Rotation *= 'TD' | 'TG'*
Ligne *= Valeur*
Colonne *= Valeur*
Longueur *= Valeur*
Couleur *= DigitHex*
Valeur *= Digit Digit Digit*
Espace *= ' '*
Digit *= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'*
DigitHex *= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'*

I – Projet.asm « programme principal »

1 – Conception

L'approche de la conception de cette étape n'est en aucun point similaire aux précédentes.

La logique de conception est la suivante :

En analysant la grammaire du programme on s'aperçoit que l'instruction principale est le type de déplacement. Nous nous sommes basé sur cette instruction.

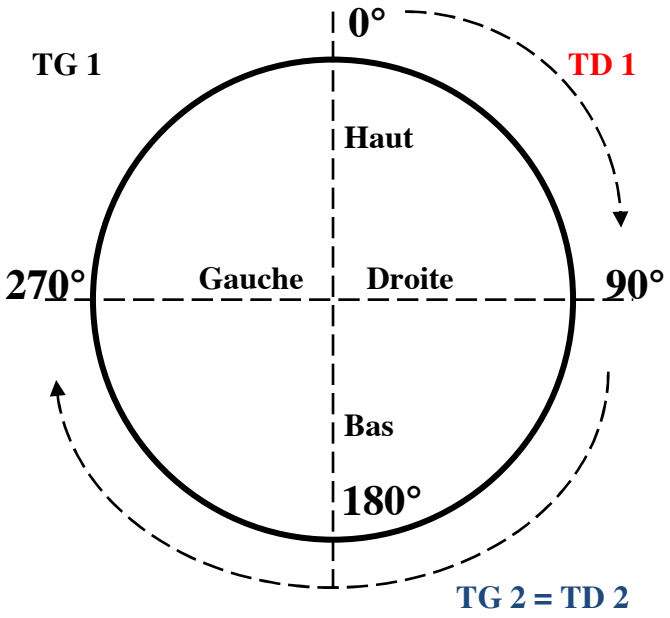
Après avoir lu la première ligne du fichier pour initialiser les coordonnées ligne et colonne, nous allons lire ligne par ligne les instructions du fichier (boucle d'analyse du fichier) jusqu'à tomber sur une instruction de type déplacement (« AV » ou « RE », comprendre « avancer » ou « reculer »).
En traitant cette instruction nous allons pouvoir exécuter celle-ci en sortant de la boucle d'analyse du fichier (grâce à une variable mise à 1).
La fonction étant traitée, nous allons retourner analyser le fichier jusqu'à tomber sur une nouvelle instruction de déplacement, ainsi de suite.

Le programme est structuré autour de 3 étiquettes principales (voir l'algorigramme page 8).

2 – Rotation LOGO

TD 1
TG 2
…

Comment traiter ces instructions ?



Le principe est le suivant :

Degres =	0°	=	haut
Degres =	90°	=	droite
Degres =	180°	=	bas
Degres =	270°	=	gauche

Au départ « degres » vaut 0.
TD = 1 :
Degres = degres + 90° = 90° = droite
TG = 2 :
Degres = degres - 180° = 90 + 180° = 270° = gauche

3 – Déplacement LOGO

Cette instruction doit gérer le déplacement du dessin.

Dans cette partie nous allons juste récupérer la distance à parcourir, puis si le déplacement est « RE » (reculer) nous allons ajouter à la variable « degres » 180° pour pouvoir faire le déplacement adéquat.
On veillera à bien rétablir cet angle de 180° après avoir effectué le déplacement pour ne pas se tromper entre la gauche et la droite dans la suite.

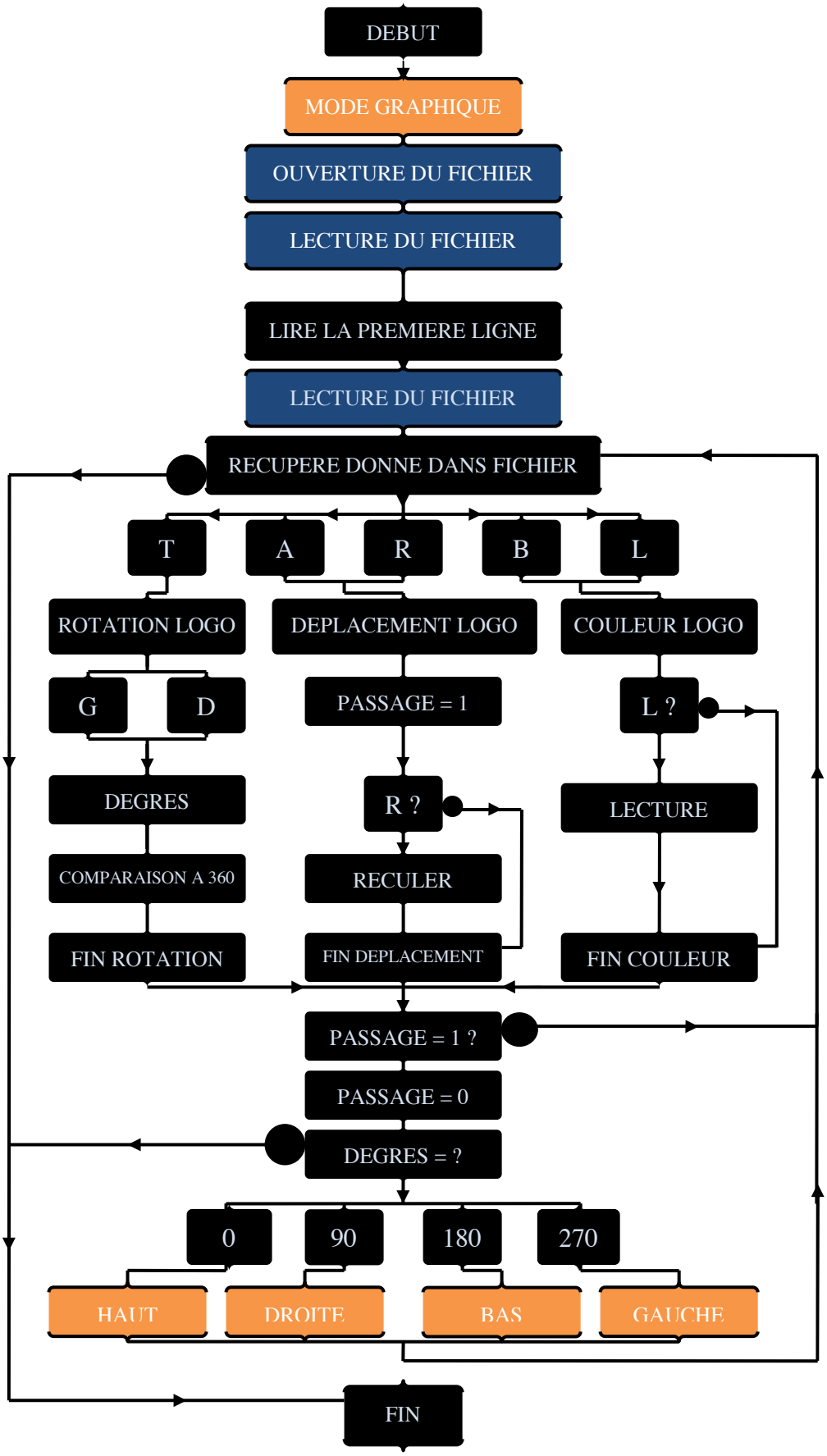
4 – Couleur LOGO

Cette instruction permet de lever le crayon (« LC ») ou de le baisser (« BC »).

Si l'instruction est « BC » nous allons juste récupérer la couleur associée.
Si l'instruction est « LC » nous allons mettre une variable à 1 pour pouvoir effectuer un déplacement sans dessin dans les procédures de dessin.

5 – Compilation

Nous disposons toujours d'un fichier nommé « text.txt » où il vous faudra placer vos instructions.


DEBUT
MODE GRAPHIQUE
OUVERTURE DU FICHIER
LECTURE DU FICHIER
LIRE LA PREMIERE LIGNE
LECTURE DU FICHIER
RECUPERE DONNE DANS FICHIER
T
A
R
B
L
ROTATION LOGO
DEPLACEMENT LOGO
COULEUR LOGO
G
D
PASSAGE = 1
L ?
DEGRES
R ?
LECTURE
COMPARAISON A 360
RECULER
FIN ROTATION
FIN DEPLACEMENT
FIN COULEUR
PASSAGE = 1 ?
PASSAGE = 0
DEGRES = ?
0
90
180
270
HAUT
DROITE
BAS
GAUCHE
FIN

ETAPE 4

I – Rotations avancées

Cette amélioration n'a pas été très dure à mettre en place, il nous a suffit de rajouter les procédures de dessin en diagonale, puis de modifier la partie rotation pour pouvoir associé les huitièmes de tour.

II – Mode interactif avancées

Pour le mode interactif, nous avons opté pour l'utilisation d'un buffer permettant de manipuler les chaînes de caractères. En effet, grâce à celui-ci l'utilisateur peut revenir en arrière en cas de faute de frappe et validera ses commandes en appuyant sur entrer comme pour toute application (cela évite de devoir finaliser la chaîne de caractères par '$').

Voici la syntaxe de ce buffer_interactif :

• buffer_interactif db 20, ?, 20 DUP(?)

- Le premier paramètre désigne le nombre maximal de caractères que peut entrer l'utilisateur, en l'occurrence ici 20. Si il essaie d'entrer plus de 20 caractères, le buffer le bloquera.
- Le second paramètre s'actualisera avec le nombre de caractères entré par l'utilisateur.
- Le troisième paramètre désigne le tableau de 20 emplacements où seront stockés les caractères entrés par l'utilisateur.

Ce buffer est manipulé grâce à l'interruption 21H 0AH–>AH qui permet d'y inscrire des données.

Nous récupérerons les informations du buffer dans le troisième paramètre ainsi :

• MOV DX, OFFSET buffer_interactif+2

Afin de coller au programme nous copierons son contenu dans le buffer manipulé par toutes les étapes, ainsi nous pourrons rendre le programme interactif.

III – Mode pilotage

Pour ce nouveau mode l'utilisateur peut dessiner directement en utilisant les touches du clavier.
Voici les différentes touches que nous avons mises à disposition :

- Les flèches pour la direction
- Le pavé numérique pour le choix de la couleur

- Les lettres ‘A’ et ‘D’ pour augmenter ou diminuer la longueur
- La lettre ‘L’ pour lever le crayon
- La touche « Échap » pour quitter

Chaque pression d’une touche du clavier appelle une étiquette qui traitera l’instruction correspondante.

IV – Interface globale

Pour cette interface nous proposons à l’utilisateur de choisir entre l’étapes 1, 2 ou 3 ou de choisir le mode pilotage.

```
        PROJET ASR2 LOGO annee 2008/2009

Bonjour, voici les etapes que vous pouvez choisir
Etape 1 : tappez 1
Etape 2 : tappez 2
Etape 3 : tappez 3
Mode pilotage : tappez 4

Veuillez entrer votre choix :
```

Si l’utilisateur à choisie une des 3 étapes nous lui proposons alors de passer en mode interactif avancées ou d’ouvrir d’un fichier déjà créé.

```
Mode Interactif : tappez 1
Mode Fichier : tappez 2

Veuillez entrer votre choix :
```

Si l’utilisateur à choisie de passer en mode pilotage nous passons directement en mode vidéo pour pouvoir dessiner à l’aide des touches mises à dispositions par le mode pilotage.