Zachary Benayyad
Rachana Or
2B

But Informatique

Université Paris Saclay
2023-2024

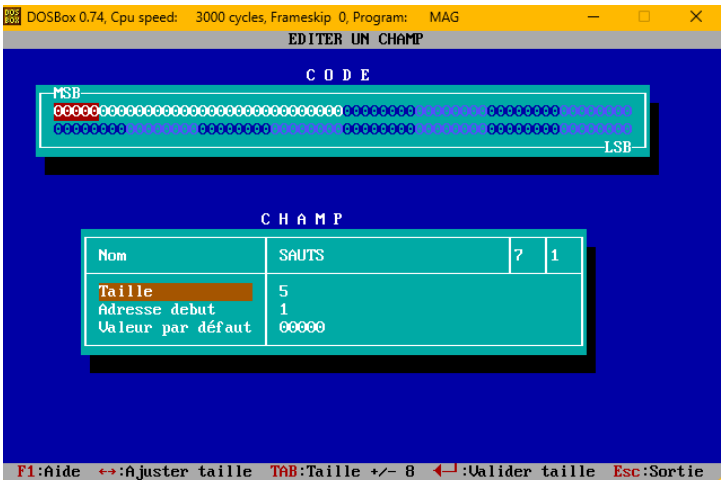
R204

Développement d'une application

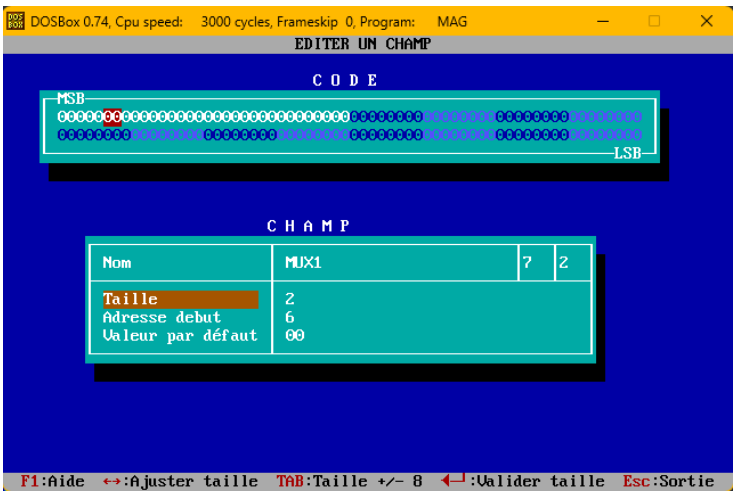
Rachana OR, Zachary Benayad

RENDU

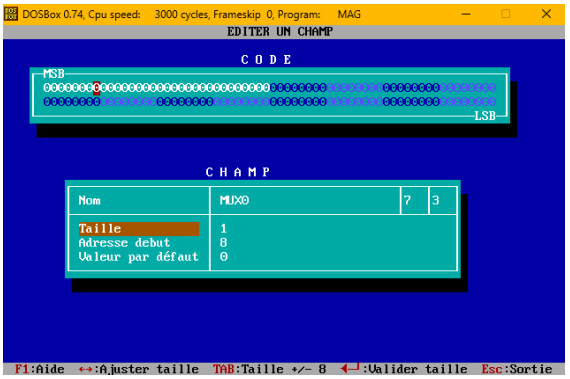
1) description des champs que nous avons créés



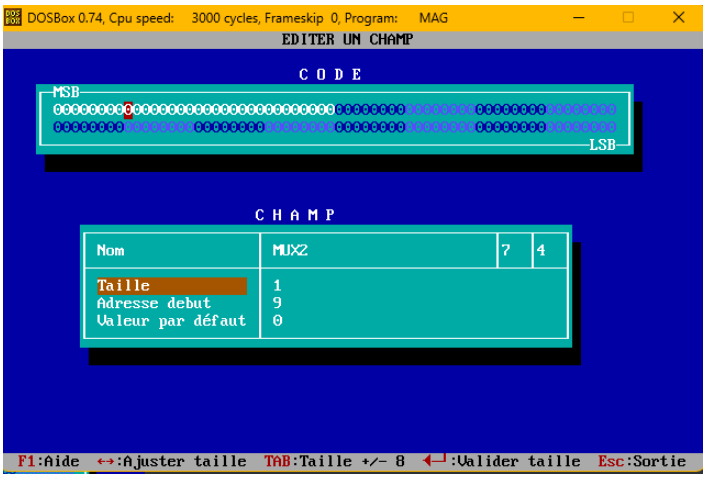
Le champ “saut” que nous avons édité nous permet de d’effectuer un saut dans le circuit, il a une taille de 5 bits , nous le savons grâce au circuit, chaque bit peut activer un saut selon sa condition (JMP, JMPZ, JMPNZ, JMPN, JMPPZ), ce saut en question est influencé par le résulat de l’ALU.



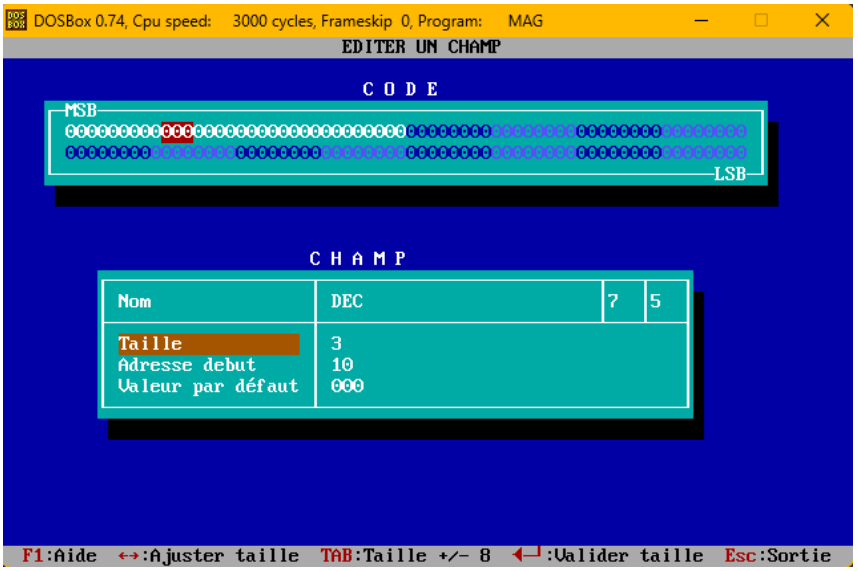
Ce MUX est le MUX1 que l'on retrouve dans le circuit,il ne s'active pas comme les autres, il permet de rentrer dans I1 la donnée des registres B, DI, SI et de la RAM de donnée en fonction de la valeur des 2 bits de contrôle du MUX1.



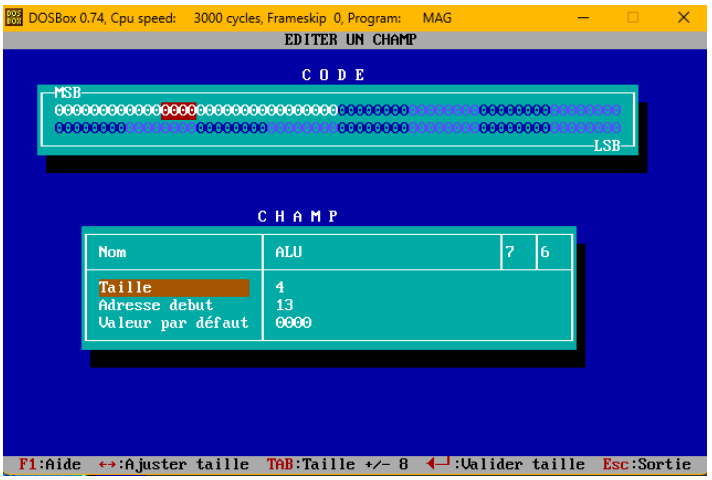
Le MUX0 est le mux qui permet d'avoir le code d'adresses sur 16 bits si on veut rentrer des valeurs précises (pour faire un mot aussi), les données qui passent par le MUX0 rentre dans I0 sur l'UAL. Il a une taille de 1 bit car nous voulons juste savoir si on l'active ou pas.



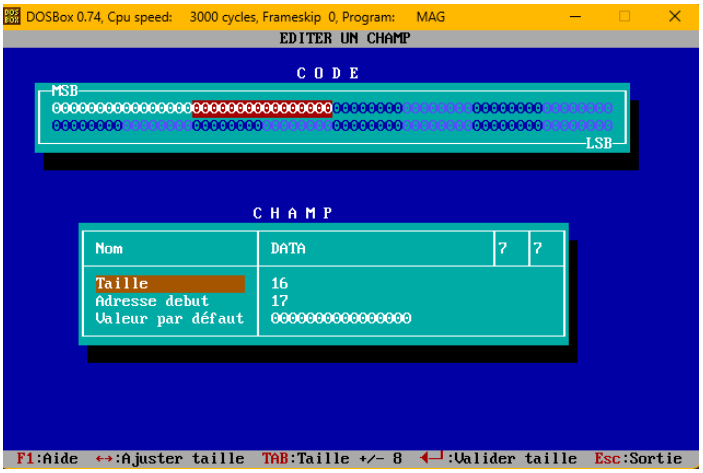
Le MUX2 est le mux qui permettra de rentrer les données dans RAM pour les registres DI et SI selon la valeur du MUX2, il a aussi une taille de 1 bit, si nous voulons l'adresse de DI nous passons le bit à 1, mais si on veut l'adresse de SI on le passe à 0. Grâce à ce mux, nous pourrons choisir qu'elle adresse on utilisera dans notre code.



Le décodeur est un champ qui à une taille de 3 bits, le choix de chacun de ses bits permettra de choisir le registre où sera écrit le résultat de l'UAL, donc A, B, SI, Di, la RAM de donnée ou aucun.



L'ALU est un champ avec une taille de 4 bits pour faire diverses opérations, si il a une taille de 4 bits nous le savons grâce aux tableaux de l'ALU qui nous montre toutes les opérations possibles entre I0 et I1 qui sont à 4 bits.



La data est le code adresses à 16 bits, si le code opératoire est à 16 bits alors l'adresse du début sera 17, c'est pourquoi 17 est son adresse de début, ce champ permet de rentrer n'importe quel valeur sur 16 bits, c'est ces valeurs que nous pourrons utiliser si on le souhaite grâce au MUX0.

2) La description détaillée des instructions que vous avez créées (avec les valeurs des champs).

LOADA #valeur :

Champ Data : valeur (16 bits)

MUX0 : activé pour rentrer la valeur immédiate dans I0 de l'UAL

Décodeur : troisième bit activé pour écrire dans le registre A

LOADSI #valeur :

Champ Data : valeur (16 bits)

MUX0 : activé pour rentrer la valeur immédiate dans I0 de l'UAL

Décodeur : quatrième bit activé pour écrire dans le registre SI

LOADADI :

Aucun champ Data requis

MUX1 : choisi la donnée du registre DI

Décodeur : troisième bit activé pour écrire dans le registre A

LOADAADRSI :

Aucun champ Data requis

MUX1 : choisi la donnée de la RAM des données à l'adresse spécifiée par SI

Décodeur : troisième bit activé pour écrire dans le registre A

LOADBADRDI :

Aucun champ Data requis

MUX1 : choisi la donnée de la RAM des données à l'adresse spécifiée par DI

Décodeur : quatrième bit activé pour écrire dans le registre B

LOADDIADRSI :

Aucun champ Data requis

MUX1 : choisi la donnée de la RAM des données à l'adresse spécifiée par SI

Décodeur : cinquième bit activé pour écrire dans le registre DI

INCSI :

Aucun champ Data requis

MUX1 : incrément de la valeur du registre SI

Aucun registre spécifié pour l'écriture

DECDI :

Aucun champ Data requis

MUX1 : décrémentation de la valeur du registre DI

Aucun registre spécifié pour l'écriture

CMPSIA :

Aucun champ Data requis

MUX1 : soustraction de A de SI

L'UAL mettra à jour les indicateurs en conséquence

CMPBA :

Aucun champ Data requis

MUX1 : soustraction de A de B

L'UAL mettra à jour les indicateurs en conséquence

JMP <label> :

Champ Data : adresse de l'instruction étiquetée par <label>

JMPNZ <label> :

Champ Data : adresse de l'instruction étiquetée par <label>

JMPPZ <label> :

Champ Data : adresse de l'instruction étiquetée par <label>

LOADASI :

Aucun champ Data requis

MUX1 : choisi la donnée du registre SI

Décodeur : troisième bit activé pour écrire dans le registre A

LOADBADRSI :

Aucun champ Data requis

MUX1 : choisi la donnée de la RAM des données à l'adresse spécifiée par SI

Décodeur : quatrième bit activé pour écrire dans le registre B

LOADADRSIB :

Aucun champ Data requis

MUX1 : choisi la donnée du registre B

Décodeur : troisième bit activé pour écrire dans la RAM des données à l'adresse spécifiée par SI

CMPDIA :

Aucun champ Data requis

MUX1 : soustraction de A de DI

L'UAL mettra à jour les indicateurs en conséquence

CMPB #valeur :

Champ Data : valeur (16 bits)

MUX0 : activé pour rentrer la valeur immédiate dans I0 de l'UAL

MUX1 : soustraction de B de la valeur immédiate

L'UAL mettra à jour les indicateurs en conséquence

SUBB #valeur :

Champ Data : valeur (16 bits)

MUX0 : activé pour rentrer la valeur immédiate dans I0 de l'UAL

MUX1 : soustraction de B de la valeur immédiate

Décodeur : quatrième bit activé pour écrire dans le registre B

JMPN <label> :

Champ Data : adresse de l'instruction étiquetée par <label>

3) c. Une explication sur les programmes réalisés.

Ex2)

```
        LOADDIADRSI
boucle  INCSI
        LOADAADRSI
        LOADBADRDI
        CMPBA
        JMPNZ FALSE
        DECDI
        LOADADI
        CMPSIA
        JMPPZ TRUE
        JMP boucle
FALSE   LOADA #-1
        JMP fin
TRUE    LOADA #1
fin     JMP fin
```

Initialisation :

LOADDIADRSI : Charge dans le registre DI la donnée de la RAM des données dont l'adresse se trouve dans le registre SI. Cela place DI à l'adresse de début de la chaîne.

Boucle de vérification de palindrome :

boucle : Incrémente l'indice de la chaîne dans SI pour parcourir la chaîne.

INCSI : Incrémente l'indice de la chaîne dans SI.

LOADAADRSI : Charge dans le registre A le premier caractère de la chaîne.

LOADBADRDI : Charge dans le registre B le dernier caractère de la chaîne.

CMPBA : Compare le premier caractère avec le dernier.

JMPNZ FALSE : Saute à l'étiquette FALSE si les caractères ne sont pas égaux.

DECDI : Décrémente l'indice de la chaîne dans DI pour avancer vers le centre de la chaîne.

LOADADI : Charge dans le registre A le caractère suivant dans la chaîne.

CMPSIA : Compare le caractère actuel avec celui à l'autre extrémité de la chaîne.

JMPPZ TRUE : Saute à l'étiquette TRUE si les caractères sont égaux.

JMP boucle : Répète la boucle jusqu'à ce que tous les caractères aient été comparés.

Fin de la vérification :

FALSE : Charge dans le registre A la valeur -1 pour indiquer que la chaîne n'est pas un palindrome.

JMP fin : Saute à l'étiquette fin pour terminer le programme.

TRUE : Charge dans le registre A la valeur 1 pour indiquer que la chaîne est un palindrome.

fin : Saute à l'étiquette fin pour terminer le programme.

Terminaison :

JMP fin : Saute à l'étiquette fin pour terminer le programme.

Pour comparer les caractéristiques de la chaîne, ce programme adopte une méthode itérative en partant des extrémités vers le centre. En cas de non-conformité des caractères à chaque itération, le programme affiche l'étiquette FALSE et signale que la chaîne n'est pas un palindrome. Autrement, si tous les éléments correspondent, le programme signale que la chaîne est un palindrome en sélectionnant l'information TRUE. Finalement, le programme se conclut en atteignant la fin.

EX4)

```
        LOADSI #-1
boucle  INCSI
        LOADDIADRSI
        LOADASI
        CMPDIA
        JMPN fin
        LOADBADRSI
        CMPB #123
        JMPPZ boucle
        CMPB #97
        JMPN boucle
        SUBB #32
        LOADADRSIB
        JMP boucle
fin     JMP fin
```

L’Initialisation :

La première instruction LOADDIADRSI charge l'adresse de début de la chaîne dans le registre DI.

LOADSI #-1 initialise le registre SI avec la valeur -1, qui sera utilisée comme indicateur pour parcourir la chaîne.

Boucle de modification pour les majuscules :

La boucle est entamée avec l'étiquette "boucle". Elle itère à travers la chaîne en incrémentant l'indice SI pour parcourir la chaîne.

À chaque itération, les instructions suivantes sont exécutées :

INCSI incrémente l'indice de la chaîne dans SI.

LOADAADRSI charge le caractère de la chaîne dans le registre A.

LOADASI charge la valeur ASCII de 'A' dans le registre A pour comparer les caractères.

CMPDIA compare le caractère actuel avec 'A'.

Si le caractère est déjà une majuscule (CMPDIA résultat négatif), le programme saute à l'étiquette "fin".

LOADBADRSI charge le caractère de la chaîne dans le registre B pour effectuer une autre comparaison.

CMPB #123 compare le caractère avec 'z' (code ASCII 122).

Si le caractère est une minuscule, le programme saute à l'étiquette "boucle" pour continuer la conversion.

CMPB #97 compare le caractère avec 'a' (code ASCII 97).

Si le caractère est une minuscule, le programme effectue la conversion en soustrayant 32 à la valeur ASCII (ce qui équivaut à la conversion en majuscule).

LOADADRSIB charge le caractère converti dans la mémoire des données.

Le programme saute ensuite à l'étiquette "boucle" pour continuer la conversion pour les caractères restants.

Fin :

L'étiquette "fin" marque la fin du programme.

Ce programme effectue une conversion des lettres minuscules en majuscules en ajustant leurs valeurs ASCII dans la mémoire des données. Le processus de conversion consiste à comparer chaque caractère avec les codes ASCII correspondant aux lettres minuscules ('a' et 'z') et en enlevant ensuite 32 de la valeur ASCII afin de procéder à la conversion. Après avoir parcouru et converti toute la chaîne, le programme prend fin.