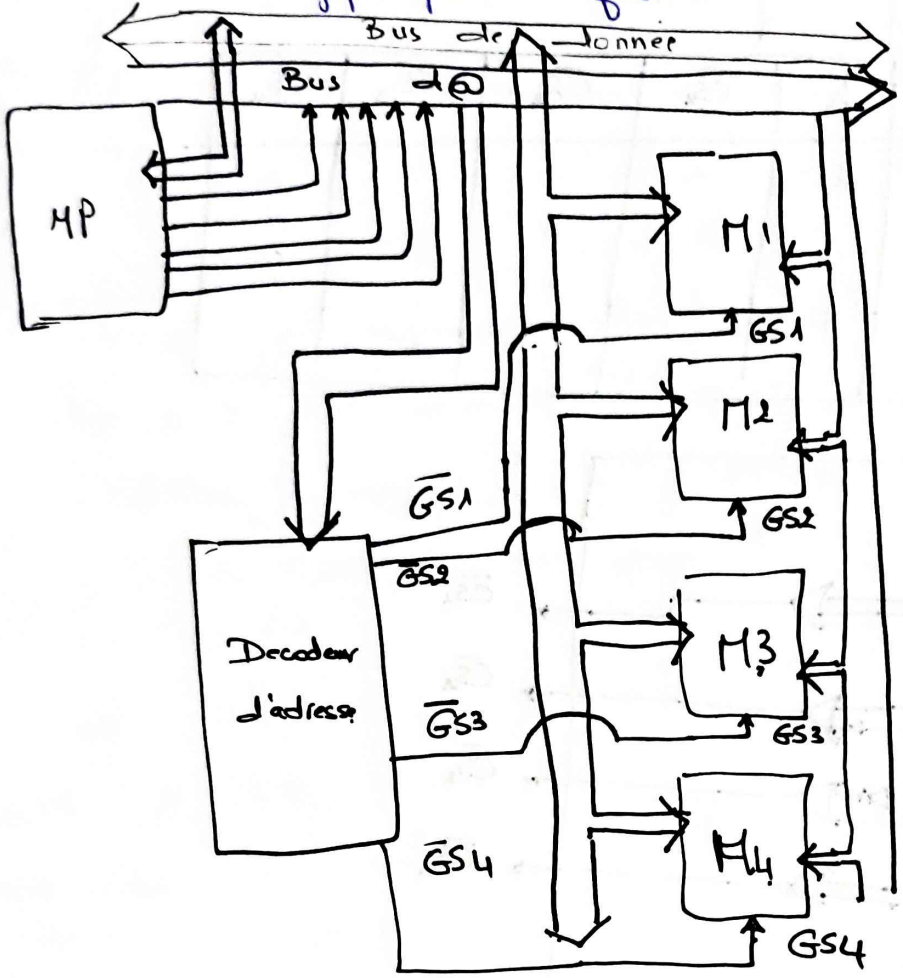
Exercice 2 :

f) Si On a un bus d@ de 16 lignes et un bus de données de 8 lignes, on associe 4 Circuits mémoires. Chaque mémoire a un bus d@ de 3 lignes. Déterminer le circuits logique pour interfacer le µP avec les 4 Circuit.



- Circuit logique de decodeur.

CS : chip select
$\overline{CS}$: validé à l'état

$\overline{CS} = 0$ → Circuit sélectionné
$\overline{CS} = 1$ → " non Selectionnée

Bus d@ : 5 bits

3 bits (A0, A1, A2)
↓
Adresse les cases mémoires dans chaque circuits

l'adresse envoyée par le µP → 3 bits
→ 2 bits → sélectionné le Circuits

A_3 et A_4

A_3	A_4	$\overline{CS}$
0	0	$\overline{CS}_1 = 0$; $\overline{CS}_2 = 1$; $\overline{CS}_3 = 1$ $\overline{CS}_4 = 1$
0	1	$CS_1 = 1$; $\overline{CS}_2 = 0$; $\overline{CS}_3 = 1$ $\overline{CS}_4 = 1$
1	0	$\overline{CS}_1 = 1$; $\overline{CS}_2 = 1$; $\overline{CS}_3 = 0$; $\overline{CS}_4 = 1$
1	1	$\overline{CS}_1 = 1$ $\overline{CS}_2 = 1$ $\overline{CS}_3 = 1$; $\overline{CS}_4 = 0$

A_3	A_4	$\overline{CS}_1$	$\overline{CS}_2$	$\overline{CS}_3$	$\overline{CS}_4$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

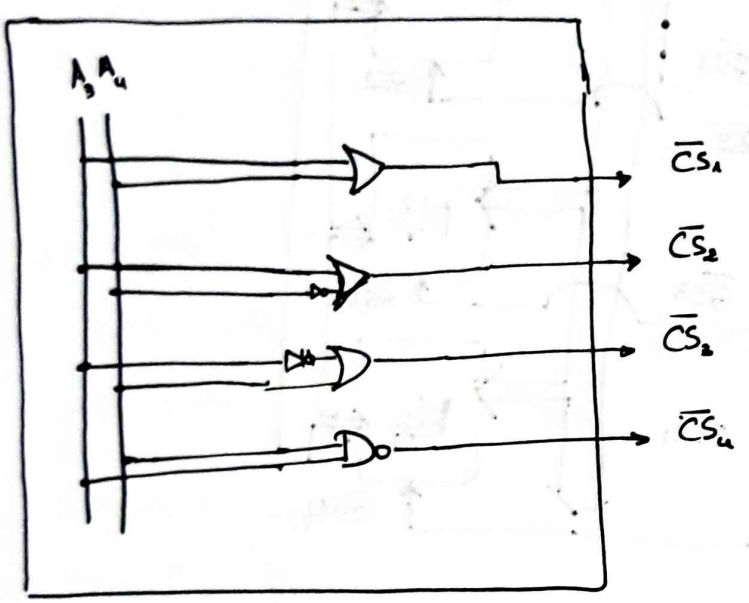
$\overline{CS}_1 = A_3 \cdot A_4$

$\overline{CS}_2 = A_3 \cdot \overline{A}_4$

$\overline{CS}_3 = \overline{A}_3 \cdot A_4$

$\overline{CS}_4 = \overline{A}_3 + \overline{A}_4$

$= \overline{A_3 \cdot A_4}$



- Decodeur d'adresse -

Pour accéder à la case n°1 dans M_4

A_4	A_3	A_2	A_1	A_0
0	0	1	0	1

0x05

Pour accéder à la case n°2 dans M_2

A_4	A_3	A_2	A_1	A_0
1	0	0	0	0

0x10

Pour accéder à la case n°6 dans M_3

A_4	A_3	A_2	A_1	A_0
0	1	1	0	1

0x0D

Pour accéder à la case n 6 dans M_4

A_4	A_3	A_2	A_1	A_0
1	1	1	1	1

0x1F

1) Ecrire un programme en assembleur ARM qui permet de :

- Transformer les données qui se trouve dans le registre de R_1 vers la case memoire n°6 dans M_4
- Récuperer la donner qui se trouve dans la case memoire n°2 dans M_2 et la mettre dans le registre R_2.
- Effectuer la somme $R_1 + R_2$ et sauvegarder dans R_1.
- Si $R_1 < 100$ Continuer sinon Arrêter

STR : Store

STR R_i, $[R_j]$

R_i : Registre qui contient les données ↪ $R_i \rightsquigarrow R_1$

$[R_j]$: Registre qui contient l'@ de la case memoire $R_j = ? \rightsquigarrow R_3$ par choix

Somme MOV R_3, # 0x04 ; $R_3 \leftarrow$ 0x04 (@ de la case memoire n°5 dans M_1)

STR R_1, $[R_3]$ ↪ Sauvegarder le Contenu de R_1 dans l'@ 0x04

MOV R_4, #0x11

LDR R_2, $[R_4]$

ADD $[R_1, R_1, R_2]$ ↪ Registre destination

CMP R_1, #. 100

Branch Less Sup : BLS Somme

END B END

Exercice :

Soit un message (Front texte) sauvegardé dans la mémoire RAM à partir de l'adresse 0x1FFF. Le texte est sauvegardé sous formes de chiffres hexadécimaux (qui correspond à le code ASCII de chaque lettre).

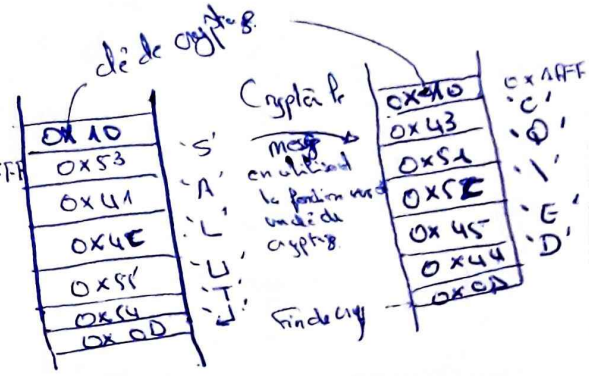
L'objectif est de crypter ce message, en utilisant la méthode 'XOR'

La méthode 'XOR' consiste à effectuer l'opération "ou Exclusif" entre le code ASCII de chaque lettre et un clé de cryptage sauvegardé dans l'adresse qui précède le message à crypter

Sachant que le message termine par un retour Chariot ('Enter') ecrire un programme en assembleur ARM qui permet de crypter le message et de le sauvegarder à partir de l'adresse 0x1FFF.

Réponse :

- Un message : "SALUT↵" → sauvegardé dans la RAM → 0x1FFF



0x41 : 0100 0001 / 0001 0000 → 0101 0001 → 5 1

0x53 : 0101 0011 / 0001 0000 → 0100 0011 → 4 3

0x4E : 0100 1110 / 0001 0000 → 0101 1110 → 5 E

0x55 : 0101 0101 / 0001 0000 → 0100 0101 → 4 5

0x54 : 0101 0100 / 0001 0000 → 0100 0100 → 4 4

"SALUT↵" → "CQIED↵"

décrypter

- Organigramme de Cryptage / décryptage :

Exercice:

1) Si un bus d'@ à 5 lignes, un bus de donnée à 8 lignes et un bus de contrôle à 2 lignes (lecture et écriture), Combien de bits peuvent être addressés et lus/écris pour un bus).

2) Quelle est la séquence d'opération effectuée par le CPU pour:

(a) Lire une donnée en mémoire ?

(b) Lire une instruction en mémoire ?

(c) écrire une donnée en mémoire ?

3) Qu'est ce qu'une UAL et quel est son rôle ?

4) Qu'est ce qu'un registre ?

5) Soit une instruction de langage machine qui peut permet de transformer (écrire) la valeur 0xE7 à l'adresse mémoire 0x1A8. au moment où cette instruction est exécuté indiquez la valeur binaire présente sur le bus d'adresse, sur le bus de données et indiquez le signal de contrôle actionné sur le bus de contrôle.

=> corrigées:

1) Bus d'adresse → 5 lignes → 5 bits.

Bus de données → 8 lignes → 8 bits.

Bus de contrôle → 2 ligne → 2 bits → Read / write.

→ combien de bits peuvent être adressé:

$N_a = 2^5 = 32$ adresse.

→ combien de bits peuvent être lus/ecris.

$N_D = 2^8 = 256 (0 \rightarrow 255)$.

2) a) lire une donnée en mémoire

i) Générer l'adresse de la donnée x en mémoire

ii) mettre l'adresse générée sur le bus d'adresse.

→ Activer la mémoire via chip select (cs).

iii) Envoyer via le bus de contrôle une requête "Read" vers la mémoire.

→ ACK: Acquitement et Reconnaissance de transfert de donnée mémoire → Bus de donnée.

iiii) ~~Réaction~~ Récupérer le donnée qui se trouve sur le bus de données.

b) Lire une instruction en mémoire:

i) Donnée → Instruction.

ii) —

iii) —

iiii) —

ix5) mettre l'instruction dans un registre spécial (Registre d'instruction).

ix6) Décoder l'instruction pour identifier son type.

ix7) Ordonner l'UAL d'excuter l'instruction + pointer vers l'instruction suivante.

c) écrire une donnée en mémoire.

i) Générer l'adresse.

ii) mettre l'adresse sur le bus d'adresse.

iii) Mettre la donnée sur le bus de données

ix4) Sélectionner la mémoire via une chip select (c

ix5) Envoyer une requête "write" vers la mémoire.

ix6) Recevoir un ACK quand l'operation d'écriture soit terminée avec succès.

5)

0xE7 → (0x118)

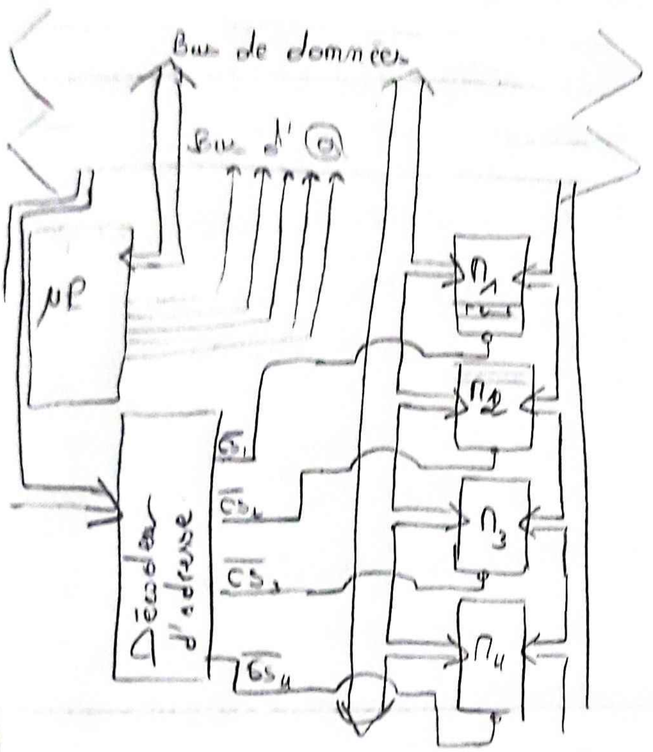
Le bus d'adresse 0x118 → 000100011000

0x118 | 0xE7

Sur le bus de donnée : 0xE7 → 1110 0111

Bus de contrôle : écrire/write

c) si on a un bus d'@ de 5 lignes et un bus de données de 8 lignes on associe 4 circuits mémoires. Chaque mémoire a un bus d'@ de 3 lignes. Déterminer le circuit logique pour interfacer le µP avec ces 4 circuits.



circuit logique d'un décodeur ???

CS : chip select

$\overline{CS}$: valide à l'état bas.

- $\overline{CS} = 0$ → circuit sélectionné
- $\overline{CS} = 1$ → circuit non sélectionné

Bus d'@ : 5 bits < 3 bits (A_0, A_1, A_2) ⇓ adresser les cases mémoires dans chaque circuit

L'adresse envoyée par le µP < 3 bits ; 2 bits → sélection le circuit

A_3	A_4	$\overline{CS_1}$	$\overline{CS_2}$	$\overline{CS_3}$	$\overline{CS_4}$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

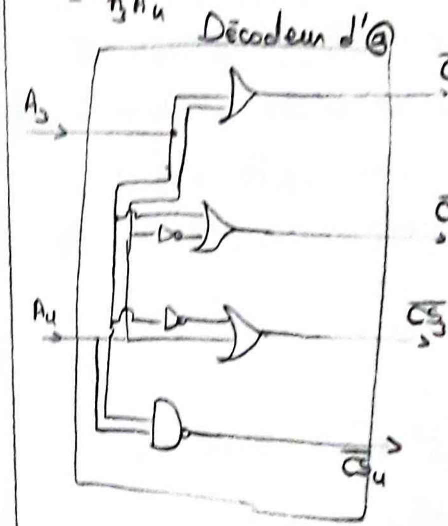
$\overline{CS_1} = A_3 + A_4$

$\overline{CS_2} = A_3 + \overline{A_4}$

$\overline{CS_3} = \overline{A_3} + A_4$

$\overline{CS_4} = \overline{A_3} + \overline{A_4}$

$= \overline{A_3 A_4}$



Pour accéder à la case n°5 dans M_1

A_4 A_3 : A_2 A_1 A_0

0 0 : 1 0 1

0x05

Pour accéder à la case n°2 dans M_2

A_4 A_3 : A_2 A_1 A_0

1 0 : 0 0 1

~~0x12~~ 0x11

PP BL crypt → @ de l'instruction est ox 1000

Fin B Fin → Après Exécution → LR = R14 = ox 1000 + 4 = ox 1004

```
PP    MOV R0, #0x1FFF
      MOV R1, #0x1FFE
      BL  crypt
Fin   B   Fin
crypt LDR R2, [R1]
loop  LDR R3, [R0]
      CMP R3, #0x0D
      BER END
      EOR R3, R3, R2
      STR R3, [R0] #4
      B   Loop
END   B   LR
```

inconnu dans d'autres microprocesseur. l'ARM par contre il passe les valeurs qui se trouve dans R0 et R1 vers la fonction

R1 → ARM → R1 = 0x1FFE

R0 = 0x1FFF

```
PP    MOV R0, #0x1FFF
      MOV R1, #0x1FFE
      BL  crypt
      MOV R0, #0x1FFF
      BL  Crypt
Fin   B   Fin
```

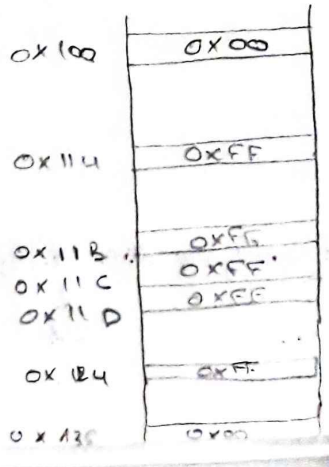
pour tester le prg. Si on retrouve le msg original alors le prg est ok.

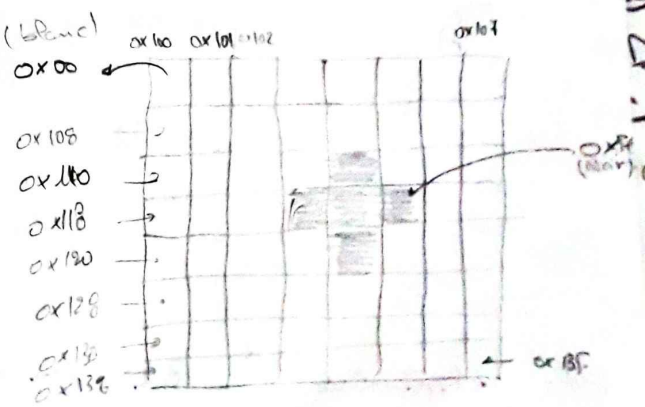
Exercice :

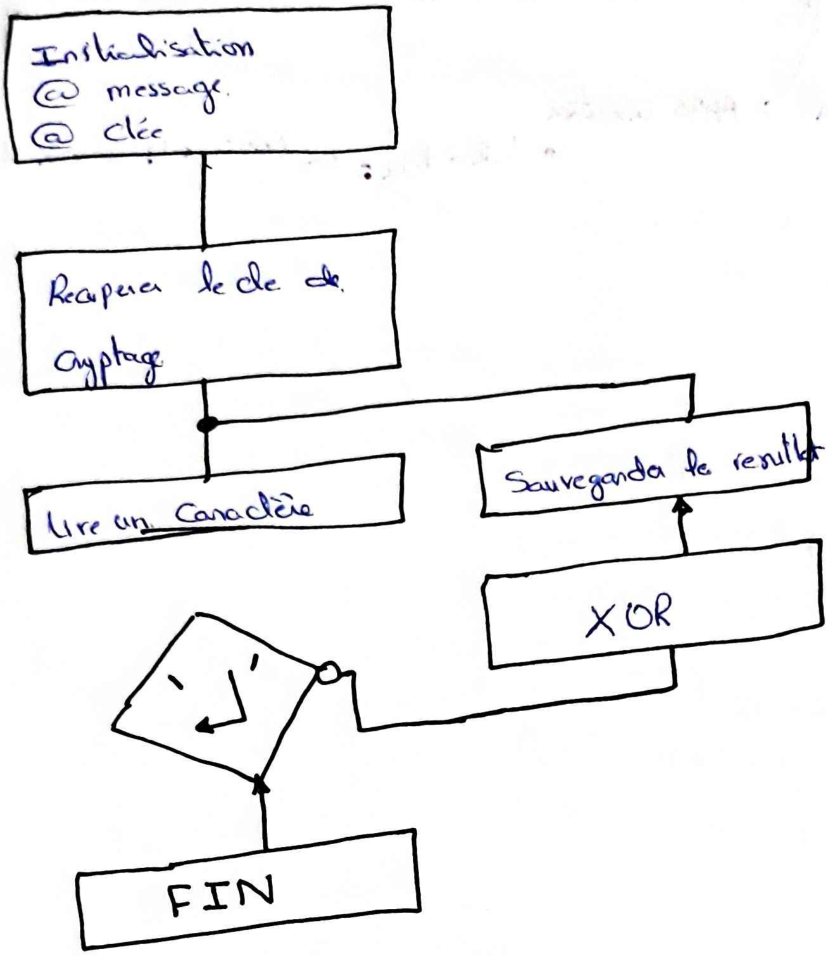
Soit une image de dimension [illegible] x [illegible] (pixels)

adresse du 1er pixel 0x100

On a 64 octets dans la mémoire







ok load le registre / apres une donnée de registre from R1 dans R2

Crypt

```
MOV R0 # 0X1FFF      (passage par valeur)
MOV R1 # 0X1FFE
LDR R2 , [R1]
```

R0 : contient l'adresse de msg

R1 " " du clé

R2 : " (0X1FFE) clé dans l'ext (0x0)

Loop

```
LDR R3 , [R0]     Chargé le donné qui se trouve dans R0
CMP R3 , # 0X0D   : comparer le caractère lu avec le retour chariot "CR"
BEQ END
EOR R3, R3, R2     R3 = R3 XOR R2
STR R3, [R0]
ADD R0, R0, # 1
B loop
```

STR R3 , [R0] , # 1 → Sauvegarder R3 dans (R0) qui se trouve dans R0 puis incrementer R0 de 1 → R0 = R0 + 1

STR R3 [R0 # 1] R0 + 1 incrementer puis sauvegarder R3 dans R0

pré-opération | incrementation before store

End.

NB :

- Appel d'une f° → BL : Nom de la f°
  ↓
  Branch and Link
  ↓
  @ de l'instruction d'appel est sauvegardée dans R14 ↓ LR
- Retourner de la fonction vers le prg principal.
  B LR

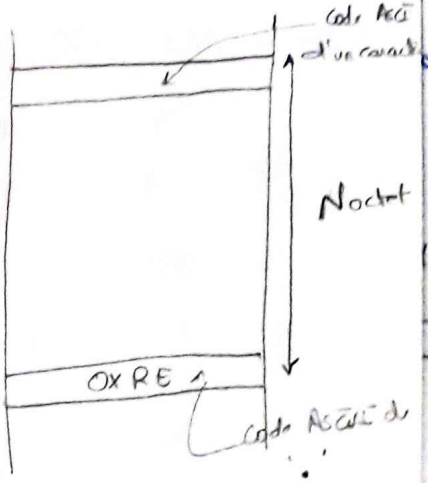
Exercice :

Soit un tableau de [illegible] octets sauvegardé dans la mémoire RAM à partir de l'adresse 0x1FFFE. Ce tableau contient [illegible] octets qui représent le code ASCI d'une chaine de caractères

1) Ecrire une fonction qui permet de déterminer la longueur de la chaine sachant que la phase se termine par un ('.')

2) Ecrire une fonction en assembleur ARM qui permet de convertir en Majuscule toutes les lettre qui se trouvent dans cette phrase

3) Ecrire un programme principal qui fait appel aux fonctions



Reponse :

1) Len Mov R_3 #0 % initialiser un compteur à 0

Boucle LDR R_2, [R_0], #1 % Load le registre from [R_0] to R_2, Post incrémenté de mémoire

CMPS R_2 R_1 % Comparer le contenu de R_2 par le code ASCII de pt '.' sauvée dans R_1

BEQ FIN % si $R_2 = R_1$ ~> sortir de boucle

ADD R_3, R_3 #1 % incrementer le compteur $R_3 = R_3 + 1$

B Boucle

FIN MOV R_0, R_3 % Mettre la longueur dans R_0 pour qu'il ne soit pas perdu

B LR % retour vers le PP (LR link register

L

Len Mov R_3, #0

Boucle LDR R_2, [R_0], #1

CMPS R_2, R_1

BEQ FIN

ADD R_3, R_3, #1

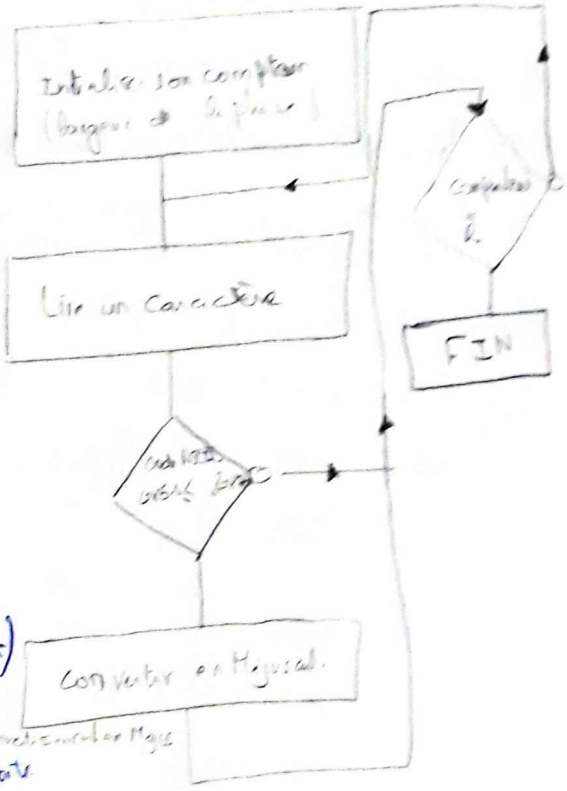
B Boucle

FIN PUSH R_3 % Sauvegarder dans l'APPEAL last in first out

B LR

2| A | 0x41 | a | 0x61

Z | 0x5A | 3 | 0x7A




```
% R0 contient @ de debut
% R1 contient la longueur de phrase.
2  MAJ   MOV  R2, #0
   Boucle LDR  R3, [R0], #1
          CMPS R3, #0x61
          ~~CMPS R3, #0x7A~~
          ~~BHS R3, #~~
          BLI  TEST          (inferieur strict)
          CMPS R3, #0x7A
          BHI  TEST          (superieur strict)
          SUB  R3, R3, #0x20  % R3 = R3 - 0x20
          STR  R3, [R0], #1   % incrementation de 1
   TEST   ADD  R2, R2, #1     % R2 = R2 + 1
          CMPS R2, R1
          BNE  Boucle
          B    LR
3| PP    MOV  R0, #0x1FFE
          MOV  R1, #0x2E
          BL   Len
          MOV  R1, R0          % sauvegarder la longueur dans R1
          MOV  R0, #0x1FFFE
          BL   MAJ
          B    END
   END
```