# ESIR-SYS1 TP 6-7-8 Assembleur: Implémenter un code de Beaufort (TP Noté)

François Taïani (`ftaiani.ouvaton.org`)*

## 1 Préliminaires

### 1.1 Préambule

- Créez tout d'abord un répertoire `tp6_7_8` dans votre répertoire `tpasm` (créé lors de la séance précédente).

### 1.2 But du TP

L'objectif de ce TP est d'implémenter un chiffre de Beaufort (`https://en.wikipedia.org/wiki/Beaufort_cipher`). Le chiffre de Beaufort est une méthode de chiffrement par substitution dérivé du code de Vigenère, mais ou la procédure de déchiffrement est identique à la procédure de chiffrement (parce que les permutations utilisées sur l'alphabet du message sont des symétries.)

## 2 Travail demandé

### 2.1 Partie I : Chiffrement pas symétrie simple (`step_1_symmetric_cipher.asm`)

La première étape consiste à réaliser un chiffrement par substitution réalisant une simple symétrie sur les lettres de l'alphabet. Un chiffrement par substitution remplace chacune des lettres d'un message par une autre lettre selon un dictionnaire de substitution qui reste fixe tout au long du message. Nous allons ici utiliser une *symétrie* sur l'alphabet romain

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ZYXWVUTSRQPONMLKJIHGFEDCBA
```

Dans cette substitution, l'alphabet est simplement inversé : un `A` est remplacé par un `Z`, un `B` par un `Y`, etc., jusqu'à `Z` qui est remplacé par un `A`. Le message `HELLO WORLD` devient ainsi `SVOOL DLIOW`. La procédure de déchiffrement est identique à celle de chiffrement (la symétrie utilisée étant son propre inverse).

Plus formellement, si l'on numérote les lettres par les nombres 0 (pour `A`) à 25 (pour `Z`) l'encodage s'écrit :

$$\text{numéro\_lettre\_codée} = 25 - \text{numéro\_lettre\_initiale}$$

#### 2.1.1 Jalon 1 : Version initiale

Réaliser un programme qui réalise un chiffrement par symétrie simple, comme décrit ci-dessus, et imprime le résultat sur la console.

Dans cette première version on supposera que la chaîne à chiffrer ne contient que des lettres majuscules, sans caractères non alphabétiques (espaces, ponctuation). On supposera que la chaîne à chiffrer est déjà stockée en mémoire dans le segment de donnée, de la manière suivante :

*francois.taiani@irisa.fr

```
SECTION   .data
message: db  "HELLO", 10                ; input message with newline
len_msg: equ $-message                  ; length of input message
```

Pour plus de simplicité, la chaîne sera chiffrée “en place” (ce qui évite d'avoir à prévoir un espace mémoire pour le résultat).

Exemple d'exécution (lorsque `message` contient `HELLO`):

```
$ ./step_1_symmetric_cipher
SVOOL
```

**Quelques conseils:**

- Pour progresser par étape, vous pouvez commencer par n'encoder qu'un seul caractère. Cela vous permettra de reporter à un deuxième temps la boucle dont vous aurez besoin pour les chaînes de plusieurs caractères.
- Attention dans votre boucle à ne pas traiter le caractère de retour chariot (code ASCII 10).
- Vérifiez que `HELLO` donne effectivement `SVOOL`, et inversement.
- Testez d'autres messages, en vérifiant que vous retombez bien sur le message initial lorsque vous ré–appliquer le chiffrement.

### 2.1.2 Jalon 2 : Prise en compte des caractères non alphabétiques

Étendez maintenant votre solution du jalon 1 pour pouvoir encoder des messages contenant des caractères non alphabétiques, en plus de lettres majuscules (nous ignorerons les lettres minuscules par simplicité). Les caractères non alphabétiques sont simplement à ignorer. Par exemple la chaîne `CA BOUM ICI!` doit maintenant donner `XZ YLFN RXR!`.

## 2.2 Partie II : Chiffrement de "César"[1] (`step_2_caesar_cipher.asm`)

Ajoutez maintenant un décalage au chiffrement de la partie I. Le décalage est représenté par la valeur que prend la lettre `A` (la "clé" du code). Par exemple si la clé est `D`, la permutation devient

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
DCBAZYXWVUTSRQPONMLKJIHGFE
```

Le message `HELLO WORLD` devient maintenant `WZSSP HPMSA`.

Formellement, le chiffrement s'écrit maintenant :

$$\text{numéro\_lettre\_codée} = (\text{numéro\_clé} - \text{numéro\_lettre\_initiale}) \mod 26$$

où numéro_clé est le numéro de la clé (par exemple 3 pour `D`).

La clé utilisée doit être stockée en mémoire dans le segment de données, comme ceci :

```
SECTION   .data
message: db  "HELLO WORLD",10           ; input message with newline
len_msg: equ $-message                  ; length of input message
key:     db  'D'                        ; key encoding the shift of the symmetry
```

**Quelques conseils**

- Une difficulté consiste à implanter l'opération modulo 26. Vous pourriez utiliser le reste de la vision Euclidienne retourné par l'instruction assembleur `div`, mais l'utilisation de `div` est un peu complexe, et le reste d'une division euclidienne n'est pas nécessairement identique à une opération

[1] Ce chiffrement est en fait légèrement plus complexe qu'un chiffre de César, qui consiste juste à opérer un décalage sur les lettres (par exemple, `A` devient `B`, `B` devient `C`, etc.)

modulo pour les nombres négatifs (ce qui est le cas ici). Dans la formule ci-dessus, la valeur de (numéro_clé − numéro_lettre_initiale) va être comprise entre -25 (quand la clé est `A` et la lettre à coder `Z`) et 25 (inversement). Vous pouvez donc implémenter le modulo avec un simple test, en ajoutant 26 lorsque cette différence est négative pour retomber sur une valeur entre 0 (correspondant à `A`) et 25 (à `Z`).

### 2.3 Partie III : Chiffrement de Beaufort (`step_3_beaufort_cipher.asm`)

Modifiez maintenant le code de votre partie II pour réaliser un chiffre de Beaufort.

Le chiffre de Beaufort fonctionne comme la méthode de chiffrement de la partie II, mais change de décalage pour chaque lettre du message à coder en utilisant une clé composée de plusieurs lettres (par exemple `DIANA`) sur lequel il boucle : Avec la clé `DIANA`, la première lettre du message est codée en utilisant `D` comme clé de décalage, la deuxième en utilisant `I`, etc., et l'on re–boucle au début de `DIANA` pour la 6ème (avec `D` de nouveau).

Avec ce nouveau chiffre, la chaîne `HELLO WORLD!` est maintenant chiffrée en `WEPCM HUJCX!` avec la clé `DIANA`. On remarque en particulier que les 2 lettres `O` du message sont chiffrées avec des caractères différents selon leur position (`M` et `U`). Comme pour les deux chiffres précédents, la procédure de déchiffrement est identique à celle de chiffrement.

La clé utilisée doit être stockée en mémoire, dans une chaîne, comme ceci :

```
SECTION   .data
message:            db  "HELLO WORLD!",10          ; input message with newline
len_msg:            equ $-message                  ; length of input message
key_string:         db  "DIANA"                    ; key of successive shifts
key_string_length:  equ $-key_string               ; length of key string
```

**Quelques conseils**

- En plus d'un index sur les caractères de la chaîne à coder, vous aurez besoin d'un second index qui parcourt la clé (`DIANA` dans l'exemple ci-dessus).
- L'index de la clé n'est incrémenté que lorsqu'une lettre doit être chiffrée. Les caractères non alphabétiques (espace, ponctuation) n'ont donc pas d'effet sur cet index.
- L'index de la clé doit revenir à zéro lorsqu'il atteint la fin de la clé (5 dans le cas de `DIANA`).

## 3 Soumission et barème

### 3.1 Soumission

- Emballez l'ensemble de vos codes source assembleur (`step_?_*.asm`) dans une archive `.zip` (par exemple avec la commande `zip`), et soumettez votre archive sur le site Moodle du cours. Attention à respecter la date de soumission indiquée sur le site Moodle du cours. Les rendus en retard seront pénalisés.

### 3.2 Barème

Votre code sera noté en fonction de (i) sa correction, et (ii) de sa qualité (intelligibilité et commentaires).

- Partie 1 : 8 points
- Partie 2 : 7 points
- Partie 3 : 5 points

## 4 Collusion, plagiat et fraude

Le code soumis doit être le résultat du travail personnel de votre binôme. L'entraide entre binômes lors d'un travail noté n'est pas interdite, mais elle doit se limiter à des conseils ponctuels et/ou de haut niveau.

Il est en revanche **strictement interdit** de soumettre tout ou partie de la solution d'un autre binôme (plagiat/copie), ou de soumettre une solution commune à plusieurs binômes, que les parties communes soient partielles ou totales (collusion). Le fait d'avoir compris le code que vous soumettez sans en être l'auteur n'est pas acceptable, car c'est ici votre capacité à produire une solution par vous-même qui est évaluée.

Des contrôles de plagiat et collusion seront réalisés sur les codes soumis. Toute étudiant ou étudiante suspecté(e) de fraude s'expose à être convoqué(e) devant la **commission disciplinaire** de l'université, et encourt des peines qui peuvent entre autres aller jusqu'à **l'exclusion temporaire ou définitive** de l'établissement.

— Fin du sujet de TP —