Rapport de Projet en Assembleur MIPS
STEGANOGRAPHIE

Marko MARTIC
Philippe WEILER

Avril 2019

Table des matières

Chapitre 1

Introduction

1.1 Get started

Instructions pour utiliser le programme :

- Modifier le chemin vers le répertoire contenant l'image en début du code source
- Suivre les instructions lors des différentes demandes d'entrées utilisateur

1.2 Présentation

Dans le cadre de l'unité d'enseignement Architecture des Ordinateurs en L2 d'Informatique à l'Université de Strasbourg, nous devons réaliser en binôme un projet en assembleur MIPS. Le but est d'implanter le système de la dissimulation d'informations par stéganographie[1].

Ce projet consistera en la réalisation de deux programmes en assembleur MIPS fonctionnant avec le simulateur MARS :

- Le premier programme demandera à l'utilisateur de choisir un fichier image (au format BMP 24 bits) et d'entrer une chaîne de caractères à cacher. Une fois la chaîne de caractères cachée dans l'image, le programme enregistrera le fichier résultat sous un nom fournit par l'utilisateur.
- Le deuxième programme demandera à l'utilisateur de choisir un fichier image et affichera la chaîne de caractère qui s'y trouve cachée. Le logiciel n'affichera ni l'image d'entrée ni l'image de sortie. Pour visualiser une image, on utilisera par exemple l'utilitaire gqview ou eog (ou tout autre utilitaire capable d'afficher une image BMP).

[1] La stéganographie est l'art de la dissimulation : son objet est de faire passer inaperçu un message dans un autre message. Elle se distingue de la cryptographie, « art du secret », qui cherche à rendre un message inintelligible à autre que qui-de-droit (Wikipedia)

Toute fonction ajoutée peut valoir un bonus dans la note finale, par exemple :

- Supporter plusieurs formats d'images BMP (autre que 24 bits).
- Cacher un fichier entier dans l'image au lieu d'une chaîne de caractères, et le restituer au décodage.
- Permettre de choisir combien de bits cacher dans chaque octet de l'image, ou à l'inverse, choisir l'espace entre les octets de l'image qui contiendront un bit ou plusieurs bits dissimulés.

1.3 Recherches

1.3.1 Images BMP

Windows bitmap est un format d'image matricielle ouvert développé par Microsoft et IBM. C'est un des formats d'images les plus simples à développer et à utiliser pour programmer. Il est lisible par quasiment tous les visualiseurs et éditeurs d'images.
L'organisation d'un fichier BMP est la suivante :

1. l'en-tête du fichier (occupe 14 octets)

Offset#	Taille	Valeur
0x0000	2	Utilisation faite du fichier(icône, image normale)
0x0002	4	La taille du fichier BMP en octets
0x0006	4	Réservé à l'application créatrice
0x000A	4	L'offset du début de l'image (début du codage des pixels)

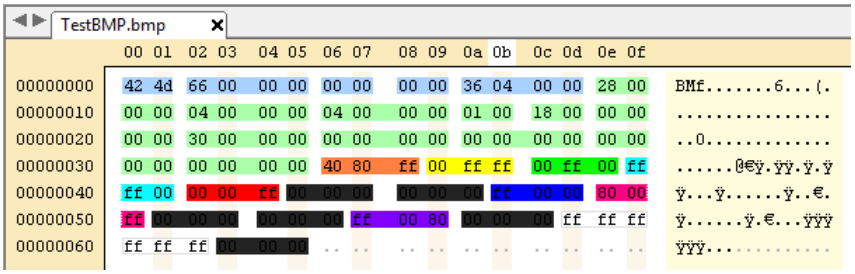
Table 1.1: en-tête du fichier

2. la palette de couleurs (non obligatoire, et inutile au programme)
3. l'en-tête de l'image
   Sont stockées ici des données telles que la taille de l'en-tête de l'image, la largeur de l'image en pixels, la longueur de l'image en pixels...
4. les données relatives à l'image
   Ces données sont situées à l'adresse indiquée à l'offset 0x000A.
   A partir de cette adresse, trois octets[2] représentent le premier pixel de l'image finale, les 3 suivants le deuxième pixel de l'image etc.
   Pour chaque groupement de 3 octets :
   - Le premier représente la proportion du rouge dans le pixel.
   - Le second la proportion du vert.
   - Le troisième la proportion du bleu.

[2]les trois octets correpondent aux trois couleures primaires de la synthèse additive

Les premiers octets de l'image correspondent aux pixels se trouvant en bas à gauche de l'image.

Pour résumer schématiquement :
D'après les binaires suivantes d'un fichier BMP :



On peut déduire que le premier pixel est orange, le deuxième jaune... Ce qui nous donne l'image suivante :

1.4 De la théorie à la pratique

Le programme encodage simplifié marche selon le schéma suivant : (les bonus sont expliqués dans le chapitre dédié)

1. Demande du nom de fichier à l'utilisateur
2. Concaténation entre le nom de l'image entré par l'utilisateur et l'adresse absolue renseignée en dur dans le code source, afin de faire un chemin pseudo-relatif[3].
3. Ouverture de l'image dans laquelle sera dissimulée la chaîne, avec un poids maximal de 12 millions d'octets : cela correspond à une image

[3] Dans la vie tout est relatif... seule la Vodka est Absolut ! :D

2000px*2000px en 24bits (2000*2000*3o = 12Mo).
En sortie on a un buffer contenant les binaires de l'image.

4. Lecture de l'en-tête du buffer (1.1) :
   (a) pour connaître la taille de l'image BMP
   (b) pour connaître l'adresse du début de l'image
5. Modification des octets à partir de l'adresse indiquée à l'adresse 0x0A du buffer.
6. Demande du nom de fichier sortant à l'utilisateur, avec les informations dissimulées.
7. Concaténation du nom de fichier sortant et du réportoire.
8. Création d'une nouvelle image avec
   (a) le buffer modifié à l'étape 6.
   (b) la taille calculée à l'étape 4a.
   (c) le nom issu de l'étape 7.

Chapitre 2

Différentes procédures

Ici seront détaillés tous les blocs et procédures annexes qui méritent que l'on s'y attarde !

2.1 Lecture/Ecriture d'un fichier

Le bloc de lecture de l'image met les binaires du fichier dans un buffer.

L'écriture de la nouvelle image est faite à partir du buffer modifié.

Ceux deux blocs ont été directement récupérés sur le site officiel de MARS.

2.2 Lecture4o : Lecture de l'en-tête du fichier

Il nous faut extraire deux éléments de l'en-tête de l'image:

- la taille du fichier BMP : pour pouvoir la renseigner au moment de l'écriture du nouveau fichier
- l'offset du début de l'image : pour connaître l'octet à partir duquel commencer à coder la chaîne

Les deux informations occupent 4 octets dans le buffer, le procédé de lecture étant le même, go pour faire une procédure :

Les octets constituant les valeurs sont rangés de gauche à droite en commençant par l'octet de poids faible.

Exemple :

1. 7000 fait 1B58 en hexa et sera écrit : 58 1B 00 00 sur 4 octets.
   Afin de lire 1B58, on load l'octet 58, puis 1B. (un load word aurait été

plus simple, mais souvent impossible car la commande .align n'a pas l'air de marcher...)
Parfois il arrivait que l'on se retrouve avec 0xFFFFFF58 au lieu de 0x58 lors de la lecture.
Ce désagrément est résolvable avec un nettoyage par un AND avec une valeur de 0x000000FF.

2. Chaque octet possédant son poids dans le nombre final, on multiplie 1B par 16^2 et 58 par 16^0.

3. On somme les deux : 0x00001B00 + 0x00000058 = 0x00001B58 = 7000.

C'est selon ce principe que fonctionne le calcul de la taille, et le calcul de l'adresse du début de l'image.

2.3 ConcatWNom : Concaténation de chaînes

Cette fonction est utilisée pour concaténer le chemin absolu défini par l'utilisateur dans le code source, et le nom du fichier entré par l'utilisateur durant l'exécution du programme.

Pour concaténer deux chaînes, on les met dans une troisième qui a l'espace alloué suffisant.

Pour ce faire, on y met d'abord les caractères un par un de la première chaîne, jusqu'à ce que le caractère traîté vaille '\0' (donc il faut faire attention à ce qu'il n'y ait pas de "\n" avant non souhaité)

Ensuite on continue à y mettre la deuxième chaîne, qui s'arrête au moment où l'on arrive au caractère "\n" (== 0x0000000A) qui est non désiré (et qui est forcément présent car, dans l'utilisation qui en est faite, il est toujours inséré par l'utilisateur à l'entrée d'une chaîne de caractères)

Chapitre 3

Encodage

3.1 Principe général

Doit être défini, dans le code source, le chemin absolu vers le répertoire contenant l'image dans laquelle effectuer la dissimulation.

Pour cacher un caractère composé de 8 bits dans l'image, on modifiera légèrement l'image.

En effet, le bit de poids faible de chaque octet de l'image sera modifié, et il correspondera à un bit du caractère, de cette façon, un caractère sera dissimulé dans 8 octets de l'image.

Cela crée des variations de couleurs imperceptibles à l'oeil nu.

Schématiquement, cela correspondra à ceci :

x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Pour y coder par exemple la lettre A, qui vaut 0x61 en hexa, soit 0b01100001 en binaire, on fera[1] :

x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	1	x	x	x	x	x	x	x	1
x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	0
x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	1	x	x	x	x	x	x	x	x

[1] A lire de gauche à droite, d'en haut vers le bas. Même si dans le fichier final ce sera bien comme indiqué dans 1.2.1

3.2 En pratique

La fonction d'encodage consiste en la modification du buffer fourni par le bloc de lecture de l'image.

Pour cette fonction, il nous faut 2 boucles imbriquées :

1. Pour parcourir toutes les lettres de la chaîne.
   - Condition d'arrêt : lettre traitée == '\0' (juste avant figure le '\n', qui sera utile au décodeur pour savoir que c'est la fin de la chaîne).
   - Incrément de boucle : Incrémente l'adresse du caractère traîté, comme un caractère occupe 1 octet, le caractère qui suit se trouve à l'adresse qui suit aussi.
2. Pour parcourir les 8 bits de la lettre en cours et avancer d'octet en octet de l'image.
   - Condition d'arrêt : les 8 bits de la lettre en cours ont été traités (compteur de 0 à 7)
   - Incrément de boucle :
     - Incrémente l'adresse de l'octet traîté de l'image.
     - Passé en paramètre, permet de modifier l'octet traité :
       * Exemple :
         Soit l'incrément de boucle 2.
         Cela veut dire qu'on traite le troisième bit du caractère, soit : 0bXX1XXXXX.
         Afin de ramener le 1 tout à droite, on doit effectuer un shift[2] vers la droite d'une valeur de $7 - 2 = 5$ pour obtenir 0bXXXXXXX1.
         Le 7 est une valeur fixe, le 2 est l'incrément de boucle.
         Si le bit à dissimuler est :
         · 1 alors on fera un OU logique entre l'octet et un 1 (=0x1), qui set le LSB[3] de l'octet à 1.
         · 0 alors on fera un ET logique entre l'octet et 0, qui set le LSB de l'octet à 0.

[2] décalage
[3] Least Significant Bit : bit de poids faible

Chapitre 4

Décodage

4.1 Principe général

Pour effectuer le décodage, on effectue la procédure de l'encodage, mais à l'envers.

- On applique tout d'abord un masque[1] permettant de récupérer le LSB.
- Une fois le LSB récupéré, on le shift à gauche pour qu'il ait le bon poids dans le caractère, et on l'additionne au caractère en cours.
- Une fois les 8 bits traités, le caractère en cours est entièrement constitué, et on peut le rajouter à la chaîne cible.
- Ainsi de suite jusqu'à ce que le caractère traîté vaille "\n" (=0x0000000a), qui est marqueur de fin de chaîne.

4.2 En pratique

Pour la partie décodage, il nous faut également deux boucles imbriquées.

1. Une qui parcourt les octets
   (a) Condition d'arrêt : caractère traité vaut '\n'
   (b) Incrément de boucle : avance d'octet en octet
2. Une qui parcourt les bits de l'octet
   (a) Condition d'arrêt : tous les bits de l'octet traités (compteur de boucle vaut 8)
   (b) Incrément de boucle i :
      i. Permet de calculer le bon shift à effectuer [2]

[1] c'est un ANDI avec 0x1 (garde la dernière valeur seulement)
[2] pour un exemple précis, voir 3.2

Chapitre 5

Bonus

5.1 Multi-bit

5.1.1 Codage

Le programme demande à l'utilisateur de choisir un nombre n de bits à cacher par octet :
avec n qui est une puissance de 2 : 1, 2, 4 ou 8.

Meme principe que dans le 3.2 sauf la partie de la modification qui est realisée à l'aide d'un masque qui met à zéro les n derniers bits de l'octet (par ex n=2 0xFF 0xFC) .
Ensuite on réalise un OU logique avec les n premiers bits de la chaîne à cacher.

1. Soit l'increment de boucle 2 et n = 2. On a l'octet qui vaut 0b10110010.
2. On applique le masque afin de mettre à zero les deux derniers bits 0b10110000.
3. Si on traite un caractère qui vaut 0b00110000.
   (a) Afin de ramener les bits tout à droite, on doit effectuer un shift logique vers la droite d'une valeur de $7 - 5 = 2$ pour obtenir 0b00000011. Le 7 est une valeur fixe le 5 est 2^2+1.
   (b) On fait donc un ou logique avec 0b10110000 et 0b00000011 et on obtient donc un octet de valeur 0b10110011.

5.1.2 Décodage

Le décodage et réaliser de la même manière que l'encodage sauf qu'on se sert du masque dans le sens inverse.

1. Soit l'incrément de boucle 2 et n = 2. On a l'octet qui vaut 0b10110010
2. On applique le masque afin de mettre à zéro les six premiers bits 0b00000011.

3. Afin de ramener les bits à la bonne position, on doit effectuer un shift logique vers la gauche d'une valeur de $7 - 5 = 2$ pour obtenir 0b00110000. Le 7 est une valeur fixe le 5 est 2^2+1

4. La stocker et l'additionner avec l'incrément suivant et ainsi de suite.

5.2 Cacher un fichier entier

5.2.1 Codage

Le programme demande à l'utilisateur de rentrer un entier qui corespond au type à cacher un fichier ou un texte comme le programme de base. Ensuite il applique le même principe que le programme de base. Hélas il n'est pas fini il faudrait y rajouter un caractère au début du codage afin d'indiquer le type du message. Mais également y rajouter les tests de taille afin de savoir si l'on peut coder tous le fichier dans l'image.

Le programme est similaire que le programme de base mais il demande soit d'entrer un nom de fichier, soit une chaîne de caractères en fonction de ce qui est désiré.

5.2.2 Décodage

Le programme de décodage est le décodage de base avec comme sortie un fichier et une chaîne de caractère du fait qu'il manque un caractère spécial afin de savoir si c'est un fichier ou une chaîne de caractères.