Projeto

Deteção de Imagens Geradas por IA através de Esteganografia em *Assembly*

1 Introdução

1.1 Motivação

O atual momento da Inteligência Artificial (IA) trouxe consigo avanços significativos na criação de imagens por meio de modelos generativos como o *Midjourney*, *o Stable Diffusion* e o *DALL·E 2*. Estas IAs têm a capacidade de criar imagens realistas e surpreendentes (*e.g.*, Figura 1), o que tem vindo a levantar questões éticas sobre a autenticidade e origem destas imagens.

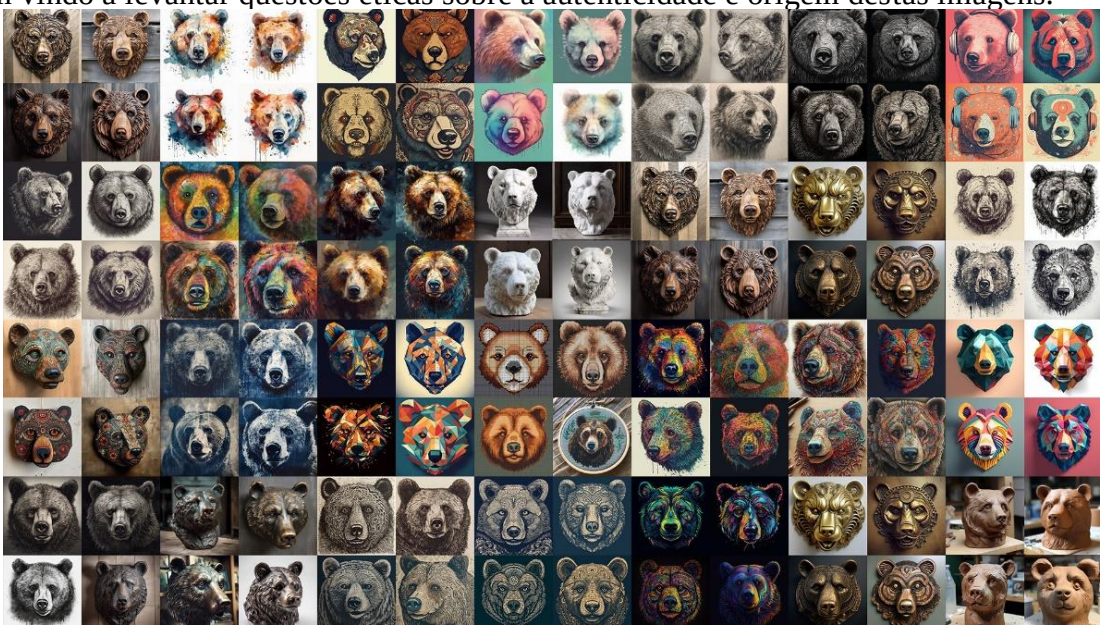

Figura 1: Exemplos de imagens em diversos estilos de ilustrações criadas por uma IA baseada em modelos generativos. Fonte: https://www.groovejones.com/explore-prompting-for-midjourney-generative-ai-artificial-intelligence-art/

A busca por métodos não-intrusivos que permitam verificar se uma imagem foi gerada por uma IA é um desafio atual crucial em diversas áreas, como a forense digital, a verificação da veracidade dos fatos (*e.g.*, *fake news*) e a segurança digital (*e.g.*, *deep fakes*). Neste projeto, vamos assumir que uma empresa fictícia de IA vos contratou para resolver este desafio através de esteganografia em *Assembly*.

1.2 Esteganografia

A esteganografia é uma área que estuda diversas técnicas para ocultar a **existência** de uma mensagem dentro de outra. Ela difere-se da criptografia, pois esta última foca-se no estudo de técnicas para ocultar o **significado** de uma mensagem. As duas áreas complementam-se, pois, um indivíduo pode querer esconder uma mensagem em outra (*i.e.*, esteganografia) ao mesmo tempo que esta mensagem oculta será gravada cifrada (*i.e.*, criptografia). Neste caso, mesmo que alguém desconfie da existência de uma mensagem oculta em outra e consiga intercetá-la, não conseguirá decifrá-la para ler o seu conteúdo.

1.3 Exemplos Práticos de Esteganografia

Em meios digitais, o conteúdo de uma mensagem pode ser de qualquer tipo de dados que possa ser representado através de dígitos binários. Texto, documentos, imagens, áudios e vídeos são exemplos comuns de tipos de dados usados em comunicações por troca de mensagens. Relativamente à esteganografia em meios digitais, um dos exemplos mais comuns consiste em **esconder uma mensagem de texto em uma imagem**, o qual será o tema deste projeto. Outros exemplos, ortogonais a este projeto, incluem esconder textos e imagens em músicas por diversão, imagens com logótipos em imagens e vídeos para a proteção de direitos de autor e de distribuição, textos subliminares em notícias para contornar meios de comunicação sob censura, *etc*.

Para o caso específico deste projeto, vamos utilizar ficheiros de imagem bitmap, os quais normalmente possuem um cabeçalho e uma secção vetorial com a informação relativa aos píxeis da imagem. Para esconder uma mensagem de texto em uma imagem, coloca-se cada bit da mensagem de texto no bit menos significativo (LSB, do inglês *Least Significant Bit*) de cada byte das cores dos píxeis da imagem. Como apenas o bit menos significativo de cada byte de cor é alterado, o resultado desta transformação será uma imagem visualmente idêntica à imagem original, porém com uma mensagem de texto escondida nela como uma marca d'água. Na Secção 2.3 detalhamos o formato de imagem *bitmap* (BMP), o qual será o formato de imagem utilizado neste projeto.

2 Contexto

O objetivo deste projeto será **criar um programa em linguagem de programação *Assembly* x86 de 64 bits**, no formato Intel para o `nasm`, **que esconda uma mensagem de texto (i.e., uma marca d'água) em uma imagem *bitmap* no formato BMP**. Esta marca d'água (em modo texto) será útil para que outros programas possam utilizá-la para detetar se uma imagem foi gerada por IA.

Nas próximas secções serão detalhados um exemplo de fluxo de execução para este projeto (Secção 2.1), o formato de texto (TXT, Secção 2.2), o formato de imagem *bitmap* (BMP, Secção 2.3), o processo sobre como esconder uma mensagem de texto num ficheiro de imagem (Secção 2.4), a apresentação dos ficheiros de apoio ao projeto fornecidos juntamente com este enunciado (Secção 2.5) e exemplos de como utilizar o `hexdump` para visualizar os bytes de um ficheiro de imagem (Secção 2.6).

2.1 Exemplo de Fluxo de Trabalho

Para contextualizar o objetivo de esconder uma mensagem de texto (*i.e.*, uma marca d'água) numa imagem e detetá-la posteriormente, é apresentado o fluxo de execução da Figura 2. Neste exemplo, um *Disseminador* (*e.g.*, de *fake news*) faz um *Pedido* a uma *Empresa de IA* para gerar uma imagem que será usada para disseminar desinformação. A *Empresa de IA* gera uma *Imagem Original*, um ficheiro de texto para a *Marca d'Água* e escolhe um *Modo m* de escrita. A *Empresa de IA* executa então o vosso programa, chamado *Aplicar*, o qual:

A1. Verifica se recebeu a quantidade correta de argumentos e imprime uma mensagem de erro em caso negativo.

A2. Aplica a esteganografia para esconder o texto contido no ficheiro da *Marca d'Água* na *Imagem Original*, seguindo o *Modo m* indicado pela *Empresa de IA*.

A3. Gera a *Imagem Modificada*.

O programa *Aplicar* recebe o seu próprio caminho (`argv[0]`) e outros quatro argumentos, nesta ordem: o *Modo m* de escrita (`argv[1]`), o caminho para o ficheiro de texto da *Marca d'Água* (`argv[2]`), o caminho para o ficheiro da *Imagem Original* em BMP (`argv[3]`) e o caminho onde deve ser escrito o ficheiro da *Imagem Modificada* em BMP (`argv[4]`).

A *Empresa de IA* pode então enviar a *Imagem Modificada* como *Resposta* ao *Pedido* do *Disseminador*. Em seguida, o *Disseminador* faz uma *Publicação* (*e.g.*, em alguma rede social na Internet) com a *Imagem Modificada*. O *Disseminador* está alheio à existência e ao uso de esteganografia por parte da *Empresa de IA* e por isso publica inadvertidamente (sem fazer novas modificações) a *Imagem Modificada* recebida como *Resposta* da *Empresa de IA*.

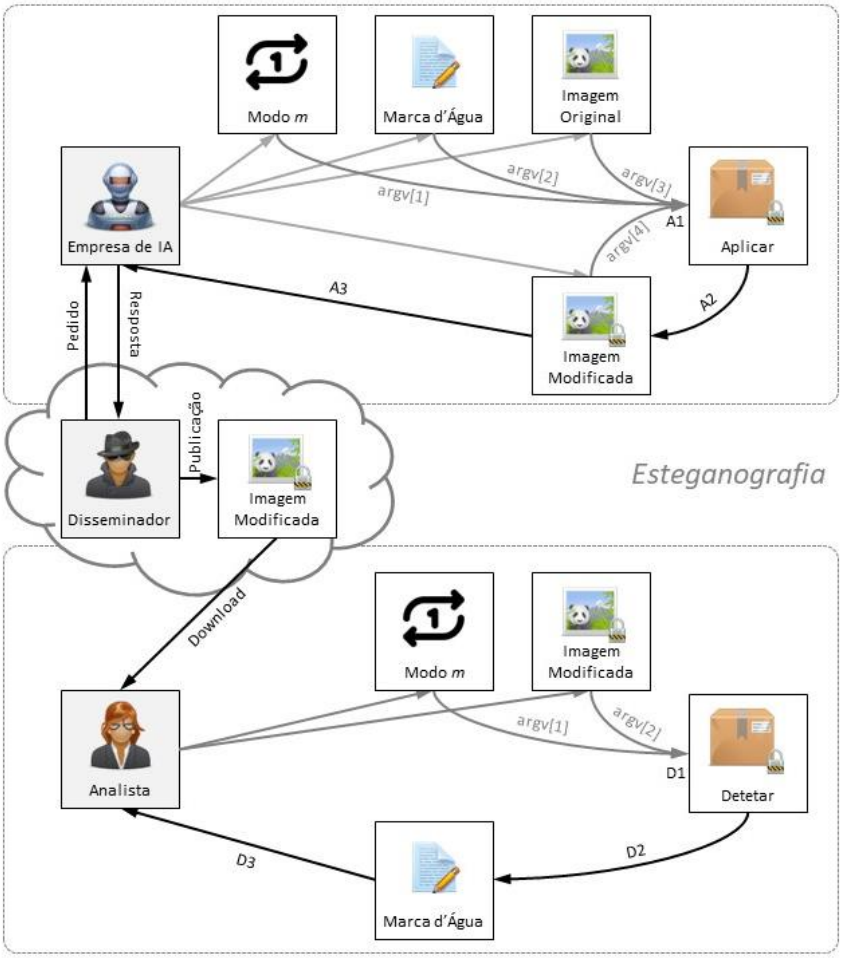

Figura 2: Exemplo de fluxo de execução do uso da esteganografia.

A seguir, uma *Analista* encontra a *Publicação* do *Disseminador* com a *Imagem Modificada* e resolve descarregá-la para avaliar a veracidade dos factos e das imagens nela contidas. Ela então executa outro programa vosso, chamado *Detetar*, o qual extrai desta *Imagem Modificada* a *Marca d'Água* que estava oculta nela e a imprime na saída padrão (*stdout*). Se o programa *Detetar* receber uma *Imagem Original* (que não possua uma *Marca d'Água* oculta), então ele naturalmente imprime na saída padrão uma mensagem aparentemente aleatória. O programa *Detetar* recebe o seu caminho (`argv[0]`) e outros dois argumentos, nesta ordem: o parâmetro do *Modo m* de leitura (`argv[1]`) e o caminho para o ficheiro da *Imagem Modificada* em BMP (`argv[2]`), e realiza os seguintes passos:

D1. Verifica se recebeu a quantidade correta de argumentos e imprime uma mensagem de erro em caso negativo.

D2. Aplica o processo inverso da esteganografia na *Imagem Modificada* para obter o texto da *Marca d'Água*;

D3. Imprime na saída do terminal (*stdout*) o texto da *Marca d'Água* recuperado.

2.2 O formato de texto TXT

O formato de texto TXT armazena os caracteres de uma mensagem. Cada caractere da mensagem é codificado em um byte (8 bits) conforme os valores definidos pela tabela da convenção ASCII. Como os valores em decimal da tabela ASCII vão do 0 ao 127, estes poderiam ser armazenados com apenas 7 bits cada. Porém para facilitar o trabalho, vamos assumir 1 byte por caractere, onde o bit mais significativo é sempre o 0 (zero) e os sete bits menos significativos são os bits normais do valor em ASCII.

Este formato não possui cabeçalho, pelo que o primeiro byte do ficheiro corresponde já ao byte do primeiro caractere. O término do ficheiro é marcado pelo caractere ASCII de valor 0 (`0x00`, *zero*), pois este corresponde ao *null*. Note que é possível também usar caracteres especiais que representam uma nova linha (`0x0A`, *new line*), uma tabulação (`0x09`, *horizontal tab*), *etc*.

Há diversas formas de se criar um ficheiro de texto TXT, seja através da linha de comandos da consola, de um editor de texto em modo texto na consola (*e.g.*, o *nano* no Linux), ou um editor de texto em modo gráfico (*e.g.*, o *gedit* no Linux ou o *Notepad* no Windows). Por exemplo, para escrever e ler um ficheiro de texto a partir da linha de comandos da consola em Linux, pode-se utilizar os seguintes comandos:

```
$ echo "Esta imagem foi gerada pela Empresa de IA." > mensagem.txt
$ cat mensagem.txt
Esta imagem foi gerada pela Empresa de IA.
```

2.3 O formato de imagem *bitmap* (BMP)

Os ficheiros *bitmap* (BMP) são ficheiros de imagens onde o valor exato de cada píxel (*px*) da imagem é apresentado explicitamente. Os ficheiros no formato BMP são divididos em duas partes: o **cabeçalho** e a secção dos **píxeis**. Estas duas partes são apresentadas na Figura 3.

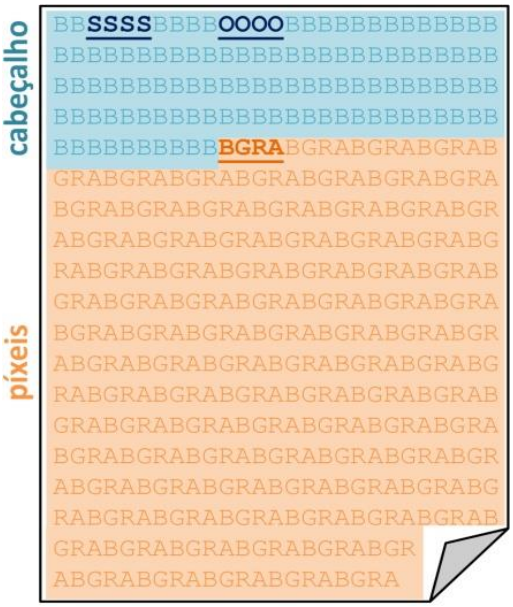

Figura 3: Estrutura de um ficheiro BMP

O cabeçalho pode seguir uma de diversas especificações existentes e pode ter um tamanho variável. Porém, para este trabalho, será utilizada a especificação ARGB32 (que será explicada a seguir) e as únicas informações que serão importantes de ler do cabeçalho serão o tamanho do ficheiro (*size*), o qual está indicado como **SSSS** na Figura 3, e o deslocamento (*offset*), o qual está indicado como **OOOO** na mesma figura. O *size* é um número do tipo *double word* (de 4 bytes de tamanho) que começa no 3º byte do ficheiro e indica o tamanho total do ficheiro BMP em bytes. O *offset* também é um número do tipo *double word* (de 4 bytes de tamanho), porém começa no 11º byte do ficheiro e indica exatamente em qual byte a secção dos píxeis inicia.

A secção dos píxeis começa a partir desta posição de deslocamento (*offset*) e é composta por conjuntos também de 4 bytes (*i.e.*, 32 bits) por píxel, representados por **BGRA** na Figura 3. Cada conjunto **BGRA** contém 1 byte para a intensidade (um número entre 0 e 255) da cor azul (**B**), 1 byte para o verde (**G**), 1 byte para o vermelho (**R**) e 1 byte para a transparência (**A**, que representa o chamado canal *alfa* e vai de 0 para totalmente transparente a 255 para totalmente opaco).

Esta especificação é chamada ARGB32 e em arquiteturas *little-endian* utilizam esta sequência **BGRA** para cada píxel. Portanto, um píxel composto pelos bytes **0x0000FFFF** é um píxel vermelho puro sem transparência, um **0x00FF00FF** é um verde puro, um **0xFF0000FF** é um azul puro, um **0x00FFFFFF** é um píxel amarelo e assim por diante. Qualquer píxel do tipo **0x??????00** é um píxel transparente. Neste trabalho, vamos sempre utilizar o valor **0xFF** (i.e., 255) para a transparência (o byte **A** de cada píxel). O fim da secção dos píxeis (e do ficheiro BMP) corresponde ao byte na posição indicada pelo número **SSSS** da Figura 3.

2.4 Como esconder uma mensagem de texto numa imagem?

Para o caso específico deste projeto, vamos esconder uma mensagem de texto (*i.e.*, uma marca d'água que originalmente está presente num ficheiro TXT) numa imagem no formato BMP. Primeiramente, obtém-se o conteúdo binário da mensagem de texto, onde cada caractere encontra-se codificado em 1 byte, conforme a tabela ASCII. É também preciso ler o ficheiro da imagem original onde vai se esconder a mensagem de texto e detetar o início da secção dos píxeis nesta imagem (ver a Secção 2.3).

Após estes passos, para esconder uma mensagem de texto em uma imagem, coloca-se cada bit da mensagem de texto no bit menos significativo (LSB, do inglês *Least Significant Bit*) de cada byte das cores dos píxeis da imagem. Os bytes da transparência (**A**) permanecem inalterados, o que evita variações de transparência na *Imagem Modificada*, as quais poderiam chamar a atenção do *Disseminador*.

Como apenas o bit menos significativo de cada byte de cor é alterado, o resultado desta transformação será uma imagem visualmente idêntica à imagem original, porém com uma mensagem de texto (uma marca d'água) escondida nela. Devido ao facto de utilizarmos em cada píxel apenas os 3 bytes de cor (o **B**, o **G** e o **R**), será possível guardar 3 bits da mensagem por cada píxel da imagem. A título de curiosidade, este modelo implica que uma imagem com `N×N`*px* possa guardar até `(N²×3)/8` caracteres da mensagem.

A ordem com que os bits da mensagem de texto são inseridos nos bytes do píxeis de cor da imagem é muito importante. Esta ordem deve ser seguida tanto para esconder a mensagem na imagem, como para recuperá-la. Mais especificamente, deve-se respeitar esta ordem:

- O **1º** bit do **1º** caracter da mensagem será guardado no LSB do 1º byte (índice 0, **B**) do 1º píxel da imagem.
- O **2º** bit do **1º** caracter da mensagem será guardado no LSB do 2º byte (índice 1, **G**) do 1º píxel da imagem.
- O **3º** bit do **1º** caracter da mensagem será guardado no LSB do 3º byte (índice 2, **R**) do 1º píxel da imagem.
- Nada será guardado no 4º byte (índice 3, **A**) do 1º píxel, pois este é o byte do canal *alfa*.
- O **4º** bit do **1º** caracter da mensagem será guardado no LSB do 1º byte (índice 0, **B**) do 2º píxel da imagem.
- ...
- O **8º** bit do **1º** caracter da mensagem será guardado no LSB do 2º byte (índice 2, **G**) do 3º píxel da imagem.
- O **1º** bit do **2º** caracter da mensagem será guardado no LSB do 3º byte (índice 2, **R**) do 3º píxel da imagem.
- Nada será guardado no 4º byte (índice 3, **A**) do 3º píxel, pois este é o byte do canal *alfa*.
- O **2º** bit do **2º** caracter da mensagem será guardado no LSB do 1º byte (índice 0, **B**) do 4º píxel da imagem.
- O **3º** bit do **2º** caracter da mensagem será guardado no LSB do 2º byte (índice 2, **G**) do 4º píxel da imagem.
- O **4º** bit do **2º** caracter da mensagem será guardado no LSB do 3º byte (índice 2, **R**) do 4º píxel da imagem.
- Novamente, será guardado no 4º byte (índice 3, **A**) do 4º píxel, pois este é o byte do canal *alfa*.
- O **5º** bit do **2º** caracter da mensagem será guardado no LSB do 1º byte (índice 0, **B**) do 5º píxel da imagem.
- E assim por diante conforme a Figura 4.

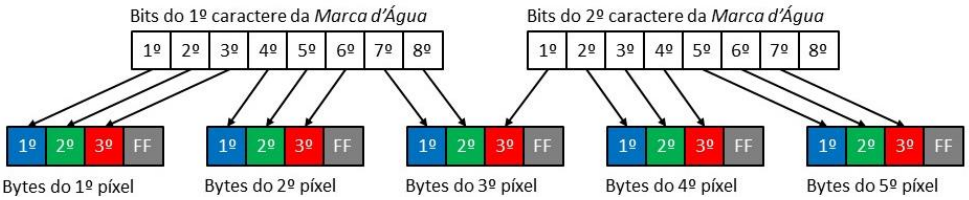

Figura 4: Diagrama a exemplificar a distribuição dos bits da Marca d'Água pelos bytes dos píxeis da imagem.

2.5 Ficheiros de apoio ao projeto

Junto ao enunciado, é disponibilizado um ficheiro (***apoio_projeto.tar.gz***) com diversos ficheiros BMP e uma biblioteca de funções para auxiliar a implementação do projeto:

- **Ficheiro básico:** O ficheiro ***básico.bmp*** serve para auxiliar a compreensão da estrutura dos ficheiros BMP e auxiliar nos estágios iniciais da implementação, pois possui uma complexidade baixa ao nível de cores. Este ficheiro possui um tamanho de `10×10`*px*, onde todos os píxeis possuem a cor *Lapis Lazuli* sem transparência (i.e., `0xFF336699`), pois esta facilita a visualização dos componentes das cores dos seus píxeis.

- **Ficheiros de teste:** Estes ficheiros servem para auxiliar a implementação dos programas e realizar testes para as fases da implementação (as quais são descritas na Secção 3.1). São disponibilizados três ficheiros, cada um com um tamanho de `150×150`*px*, os quais possuem uma complexidade elevada de cores. O ficheiro ***fcul.bmp*** auxiliará a implementação do programa *Aplicar*, pois ele é uma *Imagem Original* que não possui nenhuma marca d'água aplicada nela. Os ficheiros ***fcul_mod_S.bmp*** e ***fcul_mod_R.bmp*** auxiliarão a implementação do programa *Detetar* pois eles são ficheiros que já possuem marcas d'água ocultas neles. O primeiro possui uma marca d'água simples (*i.e.*, sem repetição) oculta através do uso de esteganografia, enquanto o segundo possui uma marca d'água repetida até o último byte de cor do último píxel da imagem.
- **Biblioteca de funções:** Esta biblioteca contém algumas funções úteis para o desenvolvimento do projeto. As principais funções implementadas nela incluem a leitura de um ficheiro de texto TXT para a memória (`readTextFile`), a leitura de um ficheiro de imagem BMP para a memória (`readImageFile`) e a escrita de um ficheiro de imagem BMP a partir de um buffer (*i.e.*, espaço) na memória (`writeImageFile`). Algumas funções secundárias também são disponibilizadas, nomeadamente uma para finalizar a execução do programa (`terminate`) e outra para escrever uma *string* no terminal de saída (`printStrLn`).

Para além destes ficheiros, os alunos podem futuramente criar ou utilizar quaisquer outras imagens BMP com o código implementado, desde que a especificação utilizada nos ficheiros BMP seja a ARGB32 (conforme mencionado na Secção 2.3). O código pode ainda ser modificado para aceitar outras especificações, mas isto é um trabalho fora do escopo deste projeto.

2.6 Visualização de uma imagem BMP com o comando `hexdump`

Pode-se utilizar o comando `hexdump` para visualizar o conteúdo binário (em hexadecimal, octal, decimal ou ASCII) de um ficheiro de imagem BMP (*e.g.*, o ficheiro ***basico.bmp*** descrito na Secção 2.5). A seguir são apresentados alguns exemplos de comandos `hexdump` que serão úteis para o desenvolvimento deste projeto:

1. Imprimir todo o conteúdo de um ficheiro, byte a byte, onde cada byte é apresentado com 2 dígitos hexadecimais na ordem em que os bytes são armazenados em arquiteturas *little-endian*:

```
$ hexdump -v -e '1/1 "%02X "' basico.bmp
42 4D 1A 02 00 00 00 00 00 00 8A 00 00 00 7C 00 00 00 0A 00 00 00 0A 00 00 00 01
00 20 00 03 00 00 00 90 01 00 00 23 2E 00 00 23 2E 00 00 00 00 00 00 00 00 00 00
00 00 FF 00 00 FF 00 00 FF 00 00 00 00 00 00 FF 42 47 52 73 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 99 66 33 FF 99 66 33 FF 99 66 33 FF …
```

2. Imprimir o tamanho (em bytes) do ficheiro BMP, o qual está guardado no seu cabeçalho. Esta informação está armazenada nos 4 bytes a partir do 3º byte do ficheiro. Para visualizá-la, basta saltar 2 bytes (com o argumento `-s`, de *skip*) e definir o tamanho da leitura para 4 bytes (com o argumento `-n`):

```
$ hexdump -v -s 2 -n 4 -e '1/4 "Tamanho (em hexadecimal): %08X\n"' basico.bmp
Tamanho (em hexadecimal): 0000021A
$ hexdump -v -s 2 -n 4 -e '1/4 "Tamanho (em decimal): %d\n"' basico.bmp
Tamanho (em decimal): 538
```

3. Pode-se validar o tamanho obtido no comando anterior através do comando `du`:

```
$ du -b basico.bmp
538
```

4. Imprimir o deslocamento (*offset*) para o início da secção dos píxeis da imagem (i.e., o fim do cabeçalho) no ficheiro BMP, o qual também está presente no cabeçalho do mesmo. Esta

informação está armazenada nos 4 bytes a partir do 11º byte do ficheiro. Para visualizá-la, basta saltar 10 bytes (i.e., `-s 10`) e definir o tamanho da leitura para 4 bytes (i.e., `-n 4`):

```
$ hexdump -v -s 10 -n 4 -e '1/4 "Offset (em hexadecimal): %08X\n"' basico.bmp
Offset (em hexadecimal): 0000008A

$ hexdump -v -s 10 -n 4 -e '1/4 "Offset (em decimal): %d\n"' basico.bmp
Offset (em decimal): 138
```

5. Imprimir somente os píxeis do ficheiro de imagem BMP, a cada 4 bytes (i.e., 32 bits) no formato **ARGB**. Note que agora o salto (`-s`) deve ser igual ao tamanho do *offset* obtido com o comando anterior (este número pode ser diferente noutros ficheiros BMP) e que a ordem dos dados apresentados difere da ordem em que eles estão armazenados (ver o primeiro exemplo do comando `hexdump`):

```
$ hexdump -v -s 138 -e '1/4 "%02X "' basico.bmp
FF336699 FF336699 FF336699 …
```

3 Implementação

3.1 Fases da implementação

A entrega do projeto consiste numa única data (ver a Secção 4.1). Porém, recomenda-se que a sua implementação seja feita em três fases separadas:

1) **Tratamento dos argumentos:** A primeira fase consiste em começar por implementar a validação do número de argumentos nos programas ***fcXXXXX_Aplicar.asm*** e ***fcXXXXX_Detetar.asm***, onde XXXXX é o número do aluno, para que estes recebam os argumentos passados pela linha de comandos (ver Secção 3.3) e os interprete apropriadamente.
2) **Detetar:** A segunda fase consiste em implementar o programa ***fcXXXXX_Detetar.asm***, onde XXXXX é o número do aluno. Esta é a parte mais simples e permitirá aos alunos contextualizarem-se melhor com o formato BMP e com algumas funcionalidades que serão úteis também na fase seguinte. Como descrito na Secção 2.5, é vos dado dois ficheiros BMP (***fcul_mod_S.bmp e fcul_mod_R.bmp***) com marcas d'água ocultas, as quais o vosso programa deverá conseguir recuperar.
3) **Aplicar:** A terceira fase consiste em implementar o programa ***fcXXXXX_Aplicar.asm***, pois este é mais complexo e o seu resultado deverá funcionar corretamente com o programa *Recuperar* criado na fase anterior. Neste caso, os alunos devem utilizar a outra imagem BMP de teste (***fcul.bmp***), pois esta é considerada uma imagem original, sem nenhuma marca d'água oculta nela.

3.2 Detalhes sobre os *Modos m* de Escrita ou Leitura

O projeto deve suportar dois *Modos m* de escrita (no caso do programa *Aplicar*) da *Marca d'Água* na *Imagem Original* para gerar a *Imagem Modificada* ou de leitura (para o programa *Detetar*) da *Marca d'Água* oculta na *Imagem Modificada*.

O primeiro modo será o SIMPLES (onde *m* = *S*) e implicará na escrita do conteúdo da *Marca d'Água* apenas uma vez nos LSBs dos bytes de cor dos píxeis da *Imagem Original*. Quando todos os bits da mensagem de texto forem escritos (resp. lidos), o programa *Aplicar* (resp. *Detetar*) termina.

O segundo modo será o REPETIDO (onde m = R) e implicará na escrita do conteúdo da *Marca d'Água* repetidas vezes até esgotar todos os LSBs dos bytes de cor dos píxeis da *Imagem Original*. Note que o conteúdo da *Marca d'Água* pode ser composto por um único caracter ASCII ou até múltiplas frases. Quando todos os bits da mensagem de texto forem escritos (resp. lidos), se ainda houver bytes de cor de píxeis disponíveis, o programa *Aplicar* (resp. *Detetar*) volta a escrever (resp. ler) todos os bits da mensagem de texto novamente. O programa executa até que não haja mais bytes de cor disponíveis na imagem. A vantagem deste segundo modo é que a marca d'água é mais resiliente a edições posteriores da *Imagem Modificada* por parte do *Disseminador*.

3.3 Detalhes de compilação e execução

- A biblioteca de funções auxiliares ***Biblioteca.asm*** fornecida (ver Secção 2.5) passará pelo *Assembler* `nasm` com o seguinte comando:
  ```
  $ nasm -F dwarf -f elf64 Biblioteca.asm
  ```
- O programa ***fcXXXXX_Aplicar.asm*** passará pelo *Assembler* `nasm` e será ligado à biblioteca de funções auxiliares fornecida através dos seguintes comandos:
  ```
  $ nasm -F dwarf -f elf64 fcXXXXX_Aplicar.asm
  $ ld fcXXXXX_Aplicar.o Biblioteca.o -o Aplicar
  ```
- O objeto *Aplicar,* resultante dos comandos anteriores, será executado através de um dos dois comandos a seguir, onde o primeiro argumento indica qual o modo de escrita a usar (S para SIMPLES ou R para REPETIDO) pelo programa. Os argumentos seguintes indicam os caminhos para os ficheiros a usar pelo programa. Os primeiros dois caminhos de ficheiros são usados como entradas do programa que devem conter, respetivamente, a mensagem (*Marca d'Água*) no formato TXT e a *Imagem Original* no formato BMP, enquanto o terceiro caminho será usado como a saída que conterá a *Imagem Modificada* resultante. Os dois comandos possíveis são:
  ```
  $ ./Aplicar S mensagem.txt img.bmp img_mod.bmp
  ```
  ou
  ```
  $ ./Aplicar R mensagem.txt img.bmp img_mod.bmp
  ```
- De modo semelhante, o programa ***fcXXXXX_Detetar.asm*** passará pelo *Assembler* `nasm` e será ligado à biblioteca de funções auxiliares fornecida através dos seguintes comandos:
  ```
  $ nasm -F dwarf -f elf64 fcXXXXX_Detetar.asm
  $ ld fcXXXXX_Detetar.o Biblioteca.o -o Detetar
  ```
- O objeto *Detetar,* resultante dos comandos anteriores, será executado através de um dos dois comandos a seguir, onde o primeiro argumento indica qual o modo de leitura a usar (S para SIMPLES ou R para REPETIDO) pelo programa. O argumento seguinte indica o caminho para a *Imagem Modificada*. Os dois comandos possíveis são:
  ```
  $ ./Detetar S img_mod.bmp
  ```
  ou
  ```
  $ ./Detetar R img_mod.bmp
  ```

Note que os comandos apresentados nesta secção assumem que todos os ficheiros de código *Assembly*, os ficheiros de texto TXT, as imagens BMP e a biblioteca fornecida devem estar todos na mesma diretoria para que estes funcionem corretamente. Pode haver casos (*e.g.*, *debug* com o SASM) em que é necessário passar o caminho absoluto de cada ficheiro (*i.e.*, os caminhos começados com o caractere "/" como em "`/home/aluno-di/ASC/projeto/01_esq.bmp`") como argumentos. **O não cumprimento da ordem e número dos argumentos descritos nesta secção implica no não funcionamento do programa e invalidará o projeto**.

3.4 Detalhes e sugestões para a implementação

Muitas das funcionalidades a serem implementadas já foram tema das aulas de ASC ou são abordadas nas referências bibliográficas da disciplina. Seguem alguns exemplos:

- A validação e obtenção dos argumentos passados a um programa: Permitirá receber, através da linha de comandos, o algoritmo de escrita (S ou R) e o caminho para os ficheiros a serem utilizados pelos programas. Se os parâmetros não estiverem de acordo com o esperado então o programa deve escrever uma mensagem de informação para a consola e terminar. Este tema foi abordado nos Exercícios 2 do [ASC14] e 1.2 do [ASC15a];
- A leitura de ficheiros: Permitirá ler a mensagem de texto em TXT e as imagens BMP. Este tema é abordado pela Secção 13.8.2 do livro [Jor20]. Note que na parte final da secção "*Read from file*" do código deste exemplo, o registo *rax* guarda na verdade a quantidade de bytes que foram lidos do ficheiro.

- A escrita de ficheiros: Permitirá escrever a imagem BMP modificada pelo programa *Aplicar*. Este tema é abordado pela Secção 13.8.1 do livro [Jor20];
- Encontrar o tamanho (*size*) e o deslocamento (*offset*) nos ficheiros BMP: Permitirá saber onde começa e onde termina a secção dos píxeis nestes ficheiros. Ver a Secção 2.3.
- Percorrer a memória byte-a-byte: Permitirá obter separadamente cada byte da mensagem ou cada byte dos píxeis da imagem BMP. Este tema foi abordado em diversos exercícios e programas com exemplos de ciclos com endereçamento indexado.
- A depuração de código com o GDB auxilia a compreensão do código durante o desenvolvimento, cujo tema foi abordado no [ASC15b].

4 Entrega do trabalho

4.1 Data da entrega

O trabalho é **individual** e deverá ser entregue até às **23:59** do dia **17/12/2023**.

4.2 Formato da entrega

A entrega final será feita através da página da disciplina no Moodle em local assinalado para tal (são permitidas ilimitadas submissões enquanto o período de disponibilidade do recurso de entrega estiver válido). É boa prática submeter o projeto ao longo do desenvolvimento cada vez que uma etapa é concluída. Caso os alunos não sigam exatamente as regras especificadas a seguir, serão penalizados na nota:

- Os ficheiros de código *Assembly* devem ser gravados exatamente com os nomes ***fcXXXXX_Aplicar.asm*** e ***fcXXXXX_Detetar.asm***, onde XXXXX é o número do aluno.
- Os alunos devem incluir, no início do ficheiro *Assembly* entregue, uma linha de comentário com o seu número de aluno (exatamente "`; fcXXXXX`").
- O ficheiro *Assembly* deve ser colocado em um ficheiro comprimido ZIP ou TAR.GZ com o número de aluno como o nome do ficheiro (exatamente ***fcXXXXX.zip*** ou ***fcXXXXX.tar.gz***).
- O ficheiro comprimido (***fcXXXXX.zip*** ou ***fcXXXXX.tar.gz***) deve conter **apenas** os ficheiros *Assembly* implementados pelo aluno e **não** deve incluir o ficheiro ***Biblioteca.asm*** fornecido.

4.3 Critérios de avaliação

1. Compilação do código e ligação das bibliotecas a funcionar sem erros. Serão utilizados os comandos apresentados na Secção 3.3 e outros semelhantes.
2. O código realizar a validação do número de argumentos passados ao programa e utilizar corretamente cada argumento na ordem especificada na Secção 3.3.
3. Funcionamento correto do programa *Aplicar* conforme especificado neste enunciado. Serão executados os comandos apresentados na Secção 3.3 com os ficheiros de teste, para além de outros ficheiros BMP.
4. Funcionamento correto do programa *Detetar* conforme especificado neste enunciado. Serão executados os comandos apresentados na Secção 3.3 com os ficheiros de teste, para além de outros ficheiros BMP.
5. Conformidade estrita com o formato de entrega descrito na Secção 4.2.
6. Organização dos códigos em *Assembly*. Aspetos que serão valorizados incluem respeitar as convenções de chamada de funções *System V ABI*, aplicar boas práticas de uso da memória, evitar referências à memória desnecessárias, *etc*.
7. Documentação dos códigos em *Assembly*. Aspetos que serão valorizados incluem comentar o funcionamento de trechos de códigos mais complexos e não-triviais.

As cotações de cada critério apresentado não serão facultadas, porém naturalmente os Critérios 3 e 4 terão o maior peso. Trabalhos vazios para cumprir apenas com o Critério 1 terão nota zero.

4.4 Plágio

Não é permitido aos alunos partilharem códigos com soluções, ainda que parciais, de nenhuma parte do projeto com outros alunos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso será reportado aos órgãos responsáveis em Ciências@ULisboa.

Chamamos a atenção para o facto das plataformas generativas de texto baseadas em Inteligência Artificial (*e.g.*, *ChatGPT*, *BARD* e *GitHub Co-Pilot*) gerarem um número limitado de soluções diferentes para o mesmo problema, as quais podem ainda incluir padrões característicos deste tipo de ferramenta e que podem ser detetáveis pelos verificadores de plágio. Desta forma, recomendamos fortemente que os alunos não submetam trechos de código gerados por este tipo de ferramenta a fim de evitar riscos desnecessários.

Por fim, é responsabilidade de cada aluno garantir que a sua *home*, os seus diretórios e os seus ficheiros de código estão protegidos contra a leitura de outras pessoas (que não o utilizador dono dos mesmos). Por exemplo, se os ficheiros estiverem gravados na sua área de aluno nos servidores de Ciências@ULisboa, então todos os itens mencionados anteriormente devem ter as permissões de acesso `700` (ver o último item da Secção 1.3 do Guião [ASC1]). Se os ficheiros estiverem no *GitHub*, garantam que o conteúdo do vosso repositório não esteja visível publicamente.

5 Bibliografia

[ASC1] M. Calha *et al*. (2023). Guião de Laboratório – Linux. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=224965.

[ASC14] M. Calha *et al*. (2023). Ficha de exercícios – x86: Pilha. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=225036.

[ASC15a] M. Calha *et al*. (2023). Guião de laboratório – x86: Pilha. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=225037.

[ASC15b] M. Calha *et al*. (2023). Guião de GDB. Disponível online em https://moodle.ciencias.ulisboa.pt/mod/resource/view.php?id=239188.

[Jor20] Ed Jorgensen. (2020). x86-64 Assembly Language Programming with Ubuntu. Disponível online em http://www.egr.unlv.edu/~ed/x86.html.