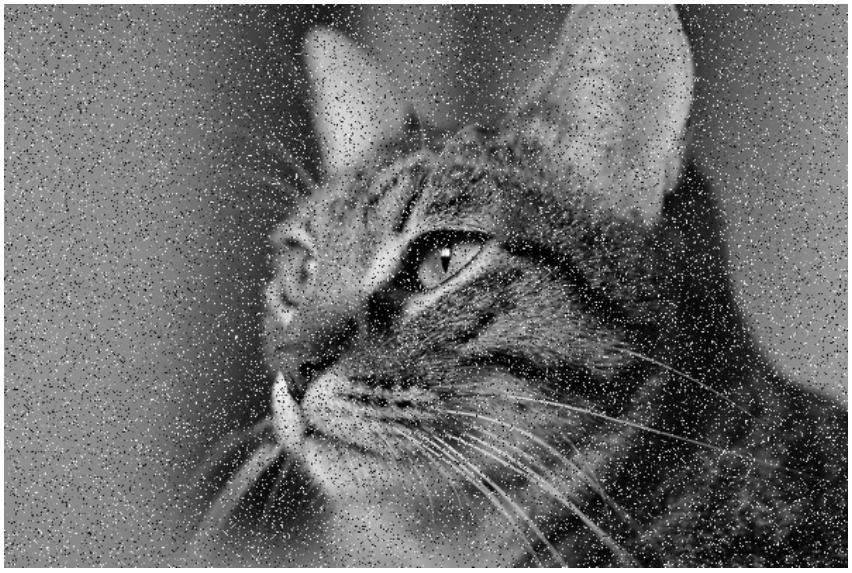
UNIVERSIDADE DE ÉVORA

Arquitetura de Computadores I

Licenciatura em Engenharia Informática

Relatório do Trabalho Prático

Remoção de Ruído de uma Imagem



Miguel Pombeiro, 57829 | Miguel Rocha, 58501

Departamento de Informática
Universidade de Évora
Maio 2024

Índice

1 Introdução **2**

2 O Programa **3**

2.1 Macro 3

2.2 *Data Segment* 4

3 Funções Implementadas **5**

3.1 FUNC_read_gray_image 5

3.2 FUNC_median_filter 5

3.3 FUNC_mean_filter 7

3.4 FUNC_write_gray_image 8

3.5 FUNC_median_of_nine 9

3.6 FUNC_mean_of_nine 9

3.7 FUNC_menu 10

3.8 main 10

4 Resultados **11**

5 Discussão/Conclusão **12**

1 Introdução

Este relatório descreve a implementação de um programa que reduz o ruído de uma imagem corrompida (*denoise*) em formato `.gray`, com recurso a *Assembly* para a arquitetura RISC-V.

Um ficheiro em formato `.gray`, é um ficheiro que contém, apenas, os valores das intensidades dos vários píxeis de uma imagem, em sequência, sem qualquer tipo de formatação extra. Este formato não contém mais nenhum tipo de informação relativa à imagem, tal como as suas dimensões ou *pixel depth*, entre outros, que são característicos de outros formatos que codificam ficheiros de imagem (como o formato `.png`, usado para visualizar a imagem após aplicação dos filtros). Desta forma, é possível idealizar o formato `.gray` como uma matriz de duas dimensões com intensidades de píxeis, ou mesmo como um vetor com organização *row-major*.

Para o *denoising* da imagem, foram propostas duas técnicas diferentes, a aplicação de um filtro de média e a aplicação de um filtro de mediana, sendo expectável que ambos produzam resultados diferentes. Para a implementação dos filtros em *Assembly* RISC-V foram seguidas as metodologias disponibilizadas no enunciado deste Trabalho.

O simulador "RARS 1.6" para *Assembly* RISC-V, usado no desenvolvimento deste trabalho possui algumas limitações, não só ao nível de desempenho bem como a falta de suporte ao nível de alocação dinâmica de memória (que teria de ser feita através do serviço alocador `sbrk` do sistema operativo). Este fator acoplado com a restrição imposta de não utilizar instruções que não façam parte da arquitetura base de RISC-V torna bastante complexa a implementação de um programa mais dinâmico que consiga lidar com algumas variações como imagens de vários tamanhos, entre outros. Desta forma, e tal como foi sugerido, o grupo decidiu fazer a implementação do programa apenas para a imagem fornecida com o enunciado deste trabalho.

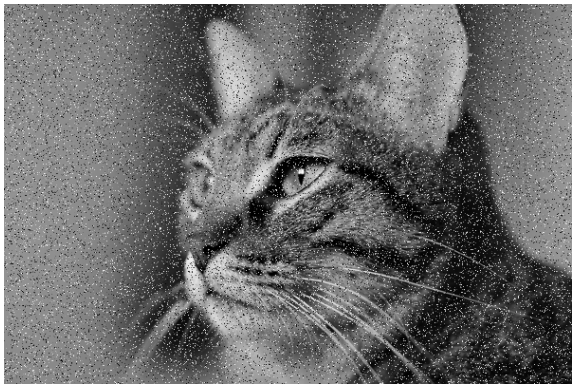


Figura 1: Imagem original, com ruído, fornecida no enunciado do trabalho.

2 O Programa

Tal como em qualquer programa de *assembly*, também este foi subdividido em dois segmentos que delimitam duas áreas de um programa: o *text segment*, que contém todas as instruções (que foram subdivididas em troços - funções) necessárias à execução do programa, e o *data segment*, que contém todo o espaço de memória (dito global), onde estão armazenadas as variáveis necessárias à execução do programa.
No inicio do desenvolvimento e de forma a reduzir a repetição de código e a melhor compartimentalizar este programa, foram definidas várias funções, com fins específicos e que serão descritas mais adiante neste relatório.
Um dos problemas que se tornou, desde logo, evidente, após uma cuidada análise do enunciado do trabalho foi a aplicação do filtro nas margens da imagem. Uma vez que, ambos os filtros dependem das intensidades dos píxeis vizinhos e que todas as margens têm uma vizinhança reduzida, quando comparadas com a maioria dos píxeis da imagem, a aplicação dos filtros nestas zonas teriam de ser feitas de forma diferente. Para resolver esta situação, foram surgindo várias ideias, tais como:

- Não aplicar qualquer filtro, copiando, apenas as margens da imagem original;
- Ignorar as margens (que ficariam a negro) aplicando, apenas, o filtro no interior;
- Aplicar o filtro individualmente a cada margem e canto da imagem;
- Fazer um *padding* da imagem original, acrescentando mais uma linha/coluna com os mesmos valores de intensidade das margens.

Devido às limitações do simulador "RARS 1.6", optou-se por não fazer o *padding* da imagem, uma vez que este teria de ser implementado aquando da leitura da imagem para memória, o que iria forçar a leitura da imagem linha a linha, tornando o programa bastante lento. Pela mesma razão, foi também descartada a ideia de tratar individualmente cada um dos cantos da imagem e os restantes píxeis das margens, uma vez que isto implicaria a verificação de uma quantidade muito grande de condições (instruções tipo B) a cada iteração do ciclo que percorre os píxeis da imagem original. Assim sendo, restavam duas soluções, que foram ambas testadas e comparadas, chegando-se à conclusão que em termos visuais da imagem final, esta ficaria mais apelativa se os píxeis das margens fossem "saltados", ficando a negro na imagem final.

2.1 Macro

De modo a evitar a repetição de código e para facilitar a sua compreensão, foi implementada a `macro` *Ending*. Esta macro é chamada em situações especificas para terminar a execução do programa e mostrar na consola uma mensagem personalizada para cada situação. Para tal, utilizaram-se quatro códigos diferentes:

- **`0`** → Este é o código que indica que o programa correu bem e que o filtro escolhido foi aplicado, cuja mensagem é "Filtro aplicado com sucesso!".
- **`-1`** → Este erro ocorre se não for encontrada a imagem com ruído, em formato `.gray`, na diretoria informada. É, exibida a mensagem "Não foi possível ler o ficheiro atual".
- **`-2`** → Este erro poderá acontecer se não for informado o nome do novo ficheiro, e está associado à seguinte mensagem "Não foi possível guardar o novo ficheiro.".
- **`-3`** → Este é o erro que aparece quando o utilizador cancela a escolha do novo filtro e confirma a saída do programa, cuja mensagem é "Programa terminado pelo utilizador.".

2.2 *Data Segment*

Devido às limitações do "RARS 1.6", já descritas, e, também, de forma a facilitar a leitura do código em *assembly*, o grupo preteriu a utilização de variáveis e espaço ditos globais para guardar vários valores e vetores que são necessários à execução do programa, em detrimento da utilização de memória alocada dinamicamente ou mesmo a utilização da *stack*. Desta forma, foram feitas várias marcações no segmento de dados por forma a facilitar o acesso às várias localizações de memória onde se encontram os vários valores. As *labels* utilizadas marcam os seguintes dados:

- **`input_name_gray`** → Esta *label* marca uma *string* com a diretoria e nome do ficheiro a ler.
- **`output_name_gray`** → Esta *label* marca uma *string* com a diretoria e nome do ficheiro a escrever, como o resultado final.
- **`prompt_text`** → Esta *label* marca uma *string* com o texto que aparece ao executar a função menu.
- **`confirm_text`** → Esta *label* marca uma *string* com o texto que aparece ao executar a função menu.
- **`original_image`** → Esta *label* marca o espaço alocado na memória que vai conter os valores da imagem original. O tamanho deste espaço é calculado pelo produto das dimensões da imagem.
- **`final_image`** → Esta *label* marca o espaço em memória onde são escritas as alterações feitas por um dos filtros à imagem original corrompida.
- **`median_space`** → Esta *label* marca um espaço em memória para um *array* utilizado nos cálculos da mediana.

3 Funções Implementadas

Nesta secção estão descritas as várias funções implementadas, bem como quais os registos utilizados e as respetivas funcionalidades. De modo a facilitar a leitura do código do programa, o grupo tentou não reutilizar os mesmos registos para mais do que uma funcionalidade dentro de uma mesma função.

3.1 FUNC_read_gray_image

Esta função utiliza a instrução `ecall` para fazer chamadas ao sistema operativo, de forma a ler um ficheiro `.gray` que contém uma imagem com ruído. Desta forma, são pedidos três serviços que permitem, respetivamente, abrir o ficheiro, ler para memória o seu conteúdo e voltar a fechar o ficheiro. Caso ocorra algum erro ao abrir o ficheiro, será executada a `macro` *Ending*, que terminará o programa com uma mensagem de erro e o código de erro `-1`.
Esta função recebe como argumento no registo `a0` o endereço de memória que contém a string com o caminho para o ficheiro `.gray` e, faz uso dos seguintes registos:

- **`a0`** → Registo auxiliar usado para dar *input* à instrução `ecall`.
- **`a1`** → Registo auxiliar usado para dar *input* à instrução `ecall`.
- **`a2`** → Registo auxiliar usado para dar *input* à instrução `ecall`.
- **`a7`** → Especifica qual o serviço a requisitar ao Sistema Operativo.
- **`s1`** → Contém uma cópia do *file descriptor* após a abertura do ficheiro.
- **`s2`** → Contém uma cópia do endereço de memória onde está guardada a imagem.
- **`s3`** → Contém o valor `-1` e é utilizado para verificar erros na abertura do ficheiro.

Por fim, esta função retorna, no registo `a0`, o endereço de memória onde está guardada a imagem `.gray` em memória.

3.2 FUNC_median_filter

Esta função contém toda a dinâmica necessária para aplicar um filtro de mediana a uma imagem no formato `.gray` em memória, de forma a reduzir o seu ruído. Para isso, é aplicada a lógica necessária à leitura da matriz da imagem, com recurso a dois ciclos, que controlam a coluna e linha da matriz que se está a percorrer.
De forma a calcular a mediana de cada píxel, todos os oito píxeis que constituem a sua vizinhança, bem como o píxel que se está a visitar, são copiados para um *array* de nove elementos que é dado como argumento à função `FUNC_median_of_nine`, que retorna a mediana dos valores contidos nesse *array*. De forma a encontrar estes píxeis da vizinhança,

foram implementados mais dois ciclos que irão, novamente, coordenar a coluna e linha de interesse da matriz imagem a que se vai aceder. No final, este valor da mediana, é armazenado em memória num outro espaço reservado para guardar a imagem já filtrada. É de notar, ainda, que não é aplicado nenhum filtro aos píxeis que se encontram nas margens da imagem, sendo que esses píxeis são "saltados" durante a iteração dos ciclos e não são guardados na imagem final.
Esta função recebe como argumentos no registo `a0` o endereço de memória onde está guardada a imagem original, no registo `a1` a largura da imagem, no registo `a2` o endereço da memória alocada para guardar a imagem filtrada e no registo `a3` o tamanho total da imagem. Durante a sua execução, a função faz ainda, uso dos seguintes registos:

- **`s0`** → Contém uma cópia do endereço onde se encontra a imagem original.
- **`s1`** → Contém uma cópia do valor da largura da imagem.
- **`s2`** → Contém uma cópia do tamanho da imagem.
- **`s3`** → Índice da coluna da imagem original que se está a visitar.
- **`s4`** → Índice da linha da imagem original que se está a visitar.
- **`s5`** → Índice da coluna do *kernel* [3x3], usado para procurar a vizinhança do píxel que se está a visitar.
- **`s6`** → Registo auxiliar usado no cálculo das condições para não aplicar o filtro nos píxeis das margens da imagem original.
- **`s7`** → Contém o endereço do *array* onde são armazenados os valores usados no cálculo da mediana.
- **`s8`** → Contém o endereço de memória alocada onde se guarda a imagem final.
- **`t0`** → Contém o endereço memória do píxel que se está a visitar.
- **`t1`** → É usado em duas ocasiões. Com o valor do píxel que se está a visitar para guardar no *array* da mediana e, mais tarde, no cálculo do endereço da imagem final onde será guardado o píxel filtrado atual.
- **`t2`** → Registo auxiliar usado no cálculo das condições para não aplicar o filtro nos píxeis das margens da imagem original.
- **`t3`** → Contém uma cópia do endereço do *array* usado no cálculo da mediana. Este registo é incrementado, o que permite iterar sobre o *array*.
- **`t4`** → Índice da linha do *kernel* [3x3], usado para procurar a vizinhança do píxel que se está a visitar.
- **`t5`** → Registo auxiliar usado na condição para executar um ciclo $(linha + largura)$.
- **`t6`** → Registo auxiliar usado na condição para executar um ciclo $(coluna + 1)$.

Esta função não retorna nada, tendo apenas inserido os valores das intensidades dos píxeis filtrados no espaço em memória alocado à nova imagem.

3.3 FUNC_mean_filter

Esta função contém toda a dinâmica necessária à aplicação de um filtro de média a uma imagem no formato `.gray`, de forma a reduzir o seu ruído. Toda a lógica de leitura da matriz imagem, em memória é semelhante à que foi implementada na função `FUNC_median_filter`, já descrita.
De forma a calcular a média de cada píxel, todos os valores de intensidade dos oito píxeis que constituem a sua vizinhança, bem como do píxel que se está a visitar, são somados num registo auxiliar (doravante denominado acumulador). Este valor é, posteriormente, passado como argumento à função `FUNC_mean_of_nine`, que calcula a sua divisão por nove, retornando, efetivamente, o valor de intensidade média do píxel em questão e da sua vizinhança. Estas intensidades dos píxeis da vizinhança são, também elas, procuradas com uma lógica igual à que já foi implementada na função que aplica o filtro de mediana.
Por fim, o valor da média encontrado é armazenado no local correspondente ao píxel que se está a visitar, dentro do espaço de memória alocado à imagem final já filtrada.
Tal como na função que aplica o filtro de mediana, os píxeis das margens são "saltados", não lhes sendo aplicado nenhum filtro, nem sendo guardados quaisquer valores de intensidade nas suas respetivas posições na imagem final, que ficam a negro.
A função recebe como argumentos no registo `a0` o endereço de memória onde está guardada a imagem original, no registo `a1` a largura da imagem, no registo `a2` o endereço da memória alocada para guardar a imagem filtrada e no registo `a3` o tamanho total da imagem. Durante a sua execução, a função faz, ainda, uso dos seguintes registos:

- **`s0`** → Contém uma cópia do endereço onde se encontra a imagem original.
- **`s1`** → Contém uma cópia do valor da largura da imagem.
- **`s2`** → Contém uma cópia do tamanho da imagem.
- **`s3`** → Índice da coluna da imagem original que se está a visitar.
- **`s4`** → Índice da linha da imagem original que se está a visitar.
- **`s5`** → Índice da coluna do *kernel* [3x3], usado para procurar a vizinhança do píxel que se está a visitar.
- **`s6`** → Registo auxiliar usado no cálculo das condições para não aplicar o filtro nos píxeis das margens da imagem original.
- **`s7`** → Registo utilizado como acumulador, onde são somados os valores das intensidades dos píxeis da vizinhança e do píxel que se está a visitar.
- **`s8`** → Contém o endereço de memória alocada onde se guarda a imagem final.

- **t0** → Contém o endereço memória do píxel que se está a visitar.
- **t1** → É usado em duas ocasiões. Com o valor do píxel que se está a visitar para guardar no *array* da mediana e, mais tarde, no cálculo do endereço da imagem final onde será guardado o píxel filtrado atual.
- **t2** → Registo auxiliar usado no cálculo das condições para não aplicar o filtro nos píxeis das margens da imagem original.
- **t4** → Índice da linha do *kernel* [3x3], usado para procurar a vizinhança do píxel que se está a visitar.
- **t5** → Registo auxiliar usado na condição para executar um ciclo ($linha + largura$).
- **t6** → Registo auxiliar usado na condição para executar um ciclo ($coluna + 1$).

Esta função não retorna nada, tendo apenas inserido os valores das intensidades dos píxeis filtrados no espaço em memória alocado à nova imagem.

3.4 FUNC_write_gray_image

Esta função utiliza a instrução `ecall` para fazer chamadas ao sistema operativo de forma a escrever num ficheiro `.gray` a informação da imagem com o filtro aplicado.
Assim, são pedidos três serviços que permitem, respetivamente, abrir o ficheiro, escrever no ficheiro e voltar a fechá-lo.
Caso ocorra algum erro ao abrir o ficheiro, será executada a `macro` *Ending*, que terminará o programa com uma mensagem de erro e o código de erro `-2`.
Esta função recebe como argumento no registo `a0` o endereço de memória que contém a string com o caminho para o ficheiro `.gray` e faz uso dos seguintes registos:

- **a0** → Registo auxiliar usado para dar *input* à instrução `ecall`.
- **a1** → Registo auxiliar usado para dar *input* à instrução `ecall`.
- **a2** → Registo auxiliar usado para dar *input* à instrução `ecall`.
- **a7** → Especifica qual o serviço a requisitar ao Sistema Operativo.
- **s1** → Contém uma cópia do *file descriptor* após a abertura do ficheiro.
- **s2** → Contém uma cópia do endereço de memória onde está guardada a imagem.
- **s3** → Contém o valor `-1` que é utilizado para verificar erros na abertura do ficheiro.

Esta função não retorna nada.

3.5 FUNC_median_of_nine

Esta é uma função auxiliar, que faz o cálculo da mediana dos valores num *array* com nove elementos. Para isso, foi implementado o algoritmo de *Selection Sort* que ordena os primeiros cinco menores elementos da lista, sendo que o último elemento ordenado (de índice quatro) é o valor da mediana.
Esta função recebe como argumentos, no registo `a0` o endereço de memória que contém o *array* de nove elementos a ordenar e no registo `a1` o tamanho do *array* introduzido, que neste caso é sempre o valor nove. Nesta função, são, ainda, utilizados os seguintes registos:

- **`a0`** → Contém o endereço de memória da posição no *array* que se está a ordenar.
- **`s0`** → Contém o endereço de memória da posição para onde o iterador está a apontar.
- **`s1`** → Contém o endereço de memória do valor mais pequeno encontrado até ao momento.
- **`t0`** → Seletor: Contém o índice da posição que se está a ordenar.
- **`t1`** → Iterador que vai à procura do próximo valor mais pequeno que ainda não está ordenado.
- **`t3`** → Contém o $(indice + 1)$ do *array*, no qual se pretende parar de ordenar (neste caso, 5).
- **`t4`** → Contém o valor que está no endereço do registo `s0`, ou seja, o valor para onde o iterador está a apontar.
- **`t5`** → Contém o valor que está no endereço do registo `s1`, ou seja, o valor mais pequeno encontrado até ao momento.

Por fim, esta função retorna, no registo `a0`, o valor da mediana.

3.6 FUNC_mean_of_nine

Esta é uma função auxiliar que calcula a divisão inteira por nove do seu argumento, fazendo, efetivamente, o cálculo da média de nove valores. Para isso, e de maneira a garantir que este projeto utilize apenas a instruções presentes na arquitetura base de RISC-V, utiliza um algoritmo da divisão - sem recurso à instrução `div` -, que consiste em subtrações sucessivas.
Recebe como argumento no registo `a0`, o valor que se pretende dividir por nove, neste caso, um somatório de nove valores. Sendo, também, utilizados os seguintes registos:

- **`a0`** → Contador do número de vezes que já foram feitas subtrações.

- **t0** → Contém uma cópia do valor que se pretende dividir, neste caso o somatório dos nove valores.
- **t1** → Contém o valor pelo qual se pretende dividir, neste caso nove(9).

Por fim, esta função retorna, no registo a0, o número de vezes que foram feitas subtrações por nove, ou seja o resultado da divisão inteira por nove.

3.7 FUNC_menu

Esta função utiliza a instrução ecall para fazer chamadas ao sistema operativo de forma a abrir um menu de texto para a seleção do filtro que o utilizador deseja a aplicar.
Caso o utilizador deseje fechar o menu ("0" ou "Cancelar"), será executada a macro *Ending*, que terminará o programa com uma mensagem e o código -3. Esta função não recebe nenhum argumento e utiliza os seguintes registos:

- **a0** → Registo auxiliar usado para dar *input* à instrução ecall.
- **a1** → Registo auxiliar usado para dar *input* à instrução ecall.
- **a7** → Especifica qual o serviço a requisitar ao Sistema Operativo.
- **s0** → Contém o valor três(3) e é usado para verificar se o utilizador escolheu uma opção válida (0, 1 ou 2).

Retorna, no registo a0, o valor inteiro correspondente à escolha do filtro da média ("1") ou da mediana ("2").

3.8 main

Esta função contém toda a dinâmica do programa e é a partir dela que são chamadas todas as outras funções, de modo a aplicar o filtro escolhido pelo utilizador. Esta função, é também responsável por garantir que todas as outras funções recebem os argumentos corretos. No final, após a aplicação do filtro e a escrita da nova imagem em formato .gray, é executada a macro *Ending* que termina o programa com uma mensagem e o código 0. Esta foi a forma encontrada para não utilizar a instrução ret nesta função, que causa um erro no "RARS 1.6".
Esta função não recebe nenhum argumento e usa os registos:

- **a0-a3** → Registos usados para dar argumentos às várias funções.
- **s1** → Contém uma cópia do valor do filtro escolhido pelo utilizador.
- **t0** → Contém o valor dois(2) e é usado para chamar a função do filtro escolhido pelo utilizador.

4 Resultados

Os resultados da aplicação dos filtros da média e da mediana estão representados nas figuras 2 e 3, respetivamente. Tal como foi mencionado anteriormente, as margens das imagens finais foram "saltadas" (não foram filtradas) e, é por isso que estes resultados apresentam uma moldura com espessura de um(1) píxel de cor preta.

Figura 2: Imagem com o filtro da média aplicado à Figura 1.

Figura 3: Imagem com o filtro da mediana aplicado à Figura 1.

5 Discussão/Conclusão

Tendo aplicado com sucesso os filtros de média e mediana à imagem fornecida, existem ainda muitas coisas que podem ser melhoradas de modo a tornar este trabalho mais abrangente e dinâmico.

Num projeto mais abrangente, em que os filtros teriam de ser aplicados a diferentes imagens, com diferentes dimensões, haveria uma necessidade tornar o programa mais dinâmico. Desta forma, não seria possível deixar os valores de várias variáveis utilizadas, ou mesmo as dimensões do espaço alocado às imagens *hardcoded* diretamente no código fonte. Haveria, então, necessidade de encontrar outra forma de produzir estes valores em *runtime*, tais como pedir *input* das dimensões ao utilizador, ou mesmo, fazer uso de um formato de ficheiro que contivesse informação das dimensões da imagem, entre outras abordagens. Para além de estarem fora do que parece ser o âmbito do trabalho e de o tornarem muito mais complexo, estas abordagens representariam um grande desafio na fase de implementação, uma vez que seria necessário recorrer ao simulador "RARS 1.6", que apresenta algumas limitações, para fazer as testar.

No que diz respeito à implementação dos filtros, e como já foi referido, as margens das imagens finais poderiam ser melhoradas através de várias técnicas já discutidas. Contudo, também elas levariam a grandes desafios durante a sua implementação no simulador "RARS 1.6" pelo que o grupo optou por não as utilizar e apenas fazer a sua descrição neste relatório.

Muito mais haveria a fazer para tornar este projeto "pronto" a utilizar por um utilizador final, tal como tomar o nome dos ficheiros das imagens em *runtime*, entre outros fatores (alguns já discutidos) que tornariam este programa mais abrangente e flexível, contudo, o grupo entende que ficaram demonstradas as suas capacidades na manipulação de matrizes de imagens em *Assembly* RISC-V.