



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 4

Processamento de Imagem: em C & Assembly

1. INTRODUÇÃO

Neste lab, irá construir projetos de programação RISC-V que executam rotinas de processamento de imagem. Os projetos incluirão múltiplos ficheiros-fonte, alguns dos quais escritos em C e outros em Assembly. Mostraremos como as funções C podem invocar rotinas Assembly e vice-versa.

2. TUTORIAL DE PROCESSAMENTO DE IMAGEM

Comece este lab examinando um programa fornecido que processa uma imagem RGB (lado esquerdo de Figura 1), e gera uma versão em escala de cinzentos dessa imagem (lado direito de Figura 1). O programa é escrito nas linguagens Assembly RISC-V e C e é configurado para correr no ambiente PlatformIO. Está disponível em:

[RVfpgaPath]/RVfpga/Labs/Lab04/ImageProcessing

O código fonte está na sub-pasta *src*.

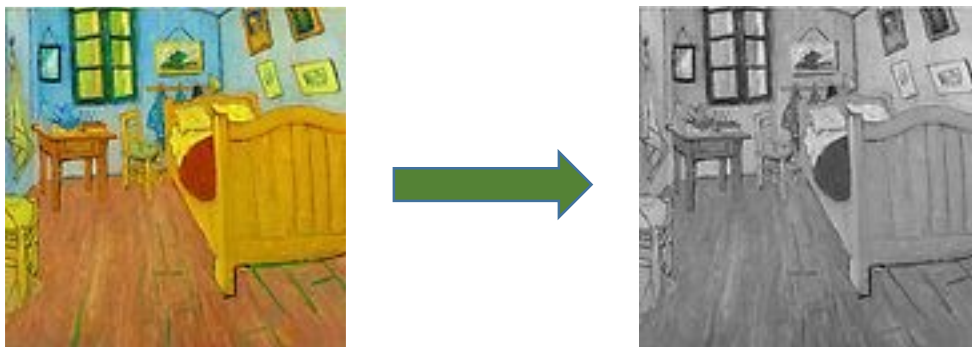


Figura 1. Transformação de uma imagem RGB para uma imagem em escala de cinzentos.

A. Estrutura e função principal do projeto (*main*)

O programa é constituído pelos seguintes ficheiros-fonte: **main.c**, **VanGogh_128.c** e **assemblySubroutines.S**. os ficheiros .c contêm funções (tais como as funções para executar as transformações de imagem) e declarações variáveis (tais como a imagem de entrada, declarada como uma matriz de caracteres sem assinatura). O ficheiro **assemblySubroutines.S** contém uma implementação em linguagem Assembly da função que transforma a imagem de rgb para escala cinzenta chamada: *ColourToGrey_Pixel*.

Figura 2 mostra a função `main` deste projeto. Invoca em primeiro lugar a função `initColourImage`, que cria uma matriz N x M com os dados da imagem de entrada. Transforma então a imagem colorida numa imagem em escala de cinzentos (função `ColourToGrey`). Finalmente, a função imprime uma mensagem e entra num loop infinito (`while (1);`).

```

49  int main(void) {
50      // Create an NxM matrix using the input image
51      initColourImage(ColourImage);
52
53      // Transform Colour Image to Grey Image
54      ColourToGrey(ColourImage, GreyImage);
55
56      // Initialize Uart
57      uartInit();
58      // Print message on the serial output
59      printfNexys("Created Grey Image");
60
61      while(1);
62
63      return 0;
64  }

```

Figura 2. Função *main* no projeto ImageProcessing

B. Imagens a cores (RGB) e em escala de cinzentos

Uma imagem consiste numa matriz de pixels, em que cada elemento da matriz representa o valor de um pixel numa determinada escala. Em RGB, cada pixel é composto por três valores, que correspondem à intensidade luminosa das componentes de vermelho - *red* (**R**), verde - *green* (**G**) e azul - blue (**B**). Portanto, cada pixel de uma imagem a cores será um vector de três componentes. Neste projeto, utilizamos a seguinte definição para o tipo de pixel RGB:

```

typedef struct {
    unsigned char R;
    unsigned char G;
    unsigned char B;
} RGB;

```

Este código define uma estrutura chamada *RGB*. Em C, um tipo de dados *struct* é um conjunto de variáveis, possivelmente de diferentes tipos, especificadas por um único nome. Esta estrutura contém três campos, todos do mesmo tipo (*unsigned char*), chamados *R*, *G* e *B*. Assim, cada canal de cor é representado por 8 bits, para que possamos distinguir entre 256 diferentes níveis de intensidade em cada canal de cor, para um total de 24 bits por pixel (24bpp). Este é um formato comum no atual processamento de imagem digital.

Para representar uma imagem em escala de cinzentos, um único valor (canal único) que varia de 0 a 255 indica o brilho de cada pixel. Neste projeto ImageProcessing, representamos a imagem em escala de cinzentos utilizando uma matriz bidimensional de caracteres:

```
unsigned char GreyImage[N][M];
```

C. Transformação de uma imagem a cores numa imagem em escala de cinzentos

A transformação entre os dois espaços de cor (RGB e Greyscale) é realizada utilizando a seguinte soma ponderada:

$$\text{grey} = 0.299 * R + 0.587 * G + 0.114 * B$$

Esta equação é baseada nos algoritmos descritos em:

<https://www.mathworks.com/help/matlab/ref/rgb2gray.html>.

Para cada pixel, calculamos o valor da escala de cinzentos multiplicando cada canal de cor pelo peso dado na equação. A soma dos pesos (0,299+0,587+0,114) é um, pelo que o valor da escala de cinzentos resultante estará dentro da gama 0-255, e assim pode ser representado com um único byte.

Para utilizar os pesos dados na equação, precisaríamos de operar com números reais, no entanto o processador **SweRV EH1** não inclui suporte de vírgula flutuante. Uma abordagem seria utilizar a emulação de vírgula flutuante, como no programa DotProduct mostrado na Secção 5.H do Guia de Introdução, no entanto, neste laboratório, utilizamos uma abordagem baseada em aritmética inteira. Os pesos são convertidos em números inteiros e a soma é uma potência de dois (no nosso caso, 2^{10}). Para converter os pesos em números inteiros, multiplicamos cada peso de ponto flutuante por 2^{10} e arredondamos para o número inteiro mais próximo:

- $0.299 \times 2^{10} = 306.176 \approx \mathbf{306}$ (peso para R)
- $0.587 \times 2^{10} = 601.088 \approx \mathbf{601}$ (peso para G)
- $0.114 \times 2^{10} = 116.736 \approx \mathbf{117}$ (peso para B)

Naturalmente, para reduzir o valor final da escala de cinzentos para o intervalo 0-255 temos de dividir a soma por 210, o que se completa facilmente deslocando o valor para a direita por 10 bits. Assim, a transformação final é obtida utilizando a seguinte fórmula:

$$\text{grey} = (306 \times R + 601 \times G + 117 \times B) \gg 10$$

Note-se que, dado que a soma das constantes (306+601+117) é 1024, o valor da escala de cinzentos resultante estará ainda dentro do intervalo 0-255.

A Figura 3 ilustra o código para a função *ColourToGrey* (lado esquerdo) e a sub-rotina *ColourToGrey_Pixel* (lado direito) que *ColourToGrey* invoca.



```

38  extern int ColourToGrey_Pixel(int R, int G, int B);
39
40  void ColourToGrey(RGB Colour[N][M], unsigned char Grey[N][M]) {
41      int i,j;
42
43      for (i=0; i<N; i++)
44          for (j=0; j<M; j++)
45              Grey[i][j] = ColourToGrey_Pixel(Colour[i][j].R, Colour[i][j].G, Colour[i][j].B);
46  }

```

```

1  .globl ColourToGrey_Pixel
2
3  .text
4
5  ColourToGrey_Pixel:
6
7      li x28, 306
8      mul a0, a0, x28
9
10     li x28, 601
11     mul a1, a1, x28
12
13     li x28, 117
14     mul a2, a2, x28
15
16     add a0, a0, a1
17     add a0, a0, a2
18
19     srl a0, a0, 10
20
21     ret
22
23 .end

```

Figura 3. Função *ColourToGrey* (implementada no ficheiro *main.c*) e sub-rotina *ColourToGrey_Pixel* (implementada no ficheiro *assemblySubroutines.S*).

Na linguagem Assembly, os símbolos (variáveis e funções/subrotinas) são locais por omissão, ou seja, são invisíveis a outros ficheiros. Para transformar esses símbolos locais em símbolos globais, temos de os exportar utilizando a diretiva assembly `.globl`. No lado

direito da Figura 3, a primeira linha (`.globl ColourToGrey_Pixel`) exporta a `ColourToGrey_Pixel` function, para que possa ser utilizado pela função `ColourToGrey`, que está num ficheiro diferente (`main.c`). No lado esquerdo da Figura 3, a primeira linha (`extern int ColourToGrey_Pixel(int R, int G, int B)`) declara a função `ColourToGrey_Pixel` como uma função externa a este ficheiro.


D. Execução do programa e visualização dos resultados

Após a conversão do código Grey estar completa, mas antes do fim da execução do programa, podemos despejar o conteúdo de algumas regiões de memória em ficheiros. Para tal, utilizamos o comando `dump` do depurador GDB. Siga os próximos passos para executar o código do projeto e obter os resultados da imagem:

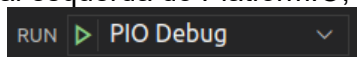
1. Abrir o VSCode e o PlatformIO.
2. Na barra de menu superior, clicar em *File* → *Open Folder* e navegue para o directório `[RVfpgaPath]/RVfpga/Labs/Lab04`. Selecione a pasta `ImageProcessing` (não o abra, mas apenas o selecione) e clique OK no topo da janela. PlatformIO irá agora abrir o projeto.
3. Abrir o ficheiro `platformio.ini` e descomente o `board_build.bitstream_file` e introduza a localização da pasta do ficheiro binário (bitfile). Por exemplo, utilize o ficheiro binário que criou em Lab 1.

```
board_build.bitstream_file =
[RVfpgaPath]/RVfpga/Labs/Lab01/Project1/Project1.runs/impl_1/rvfpganexys.bit
```

4. Abrir todos os ficheiros-fonte localizados na pasta `src` (`main.c`, `assemblySubroutines.S`) e analisá-los para que compreenda claramente como funciona o programa.
5. Configure o RVfpgaNexys na placa Nexys A7 clicando no ícone do PlatformIO na faixa do menu da esquerda, depois expandindo *Project Tasks* → *env:swervolf_nexys* → *Platform* e clique em *Upload Bitstream*. Lembre-se que também pode executar estes programas em simulação, usando Verilator e Whisper.
6. Executar o programa no PlatformIO. Pode fazê-lo ou na placa (neste caso deve primeiro configurar o RVfpgaNexys para a Nexys A7, como descrito no passo anterior) ou usando o simulador Whisper (como descrito no Guia de Primeiros

Passos do RVfpga). Em qualquer caso, clicar no botão "Executar".  , que está disponível na barra lateral esquerda do PlatformIO, e depois iniciar o depurador,

clicando no botão "Play"



A execução irá parar no início da função principal, por isso, retome-a clicando no

botão "Continue".



Após um curto período de tempo (cerca de 1 segundo), o programa terá completado as transformações de imagem em escala de cinzentos explicadas acima e terá atingido um ciclo infinito (`while(1);`) no fim (ver Figura 2). Pausa a execução

clcando no botão “Pause”



- Exportar a imagem cinzenta (GreyImage), executando os seguintes comandos na Consola de Depuração (Debug Console) (ver Figura 4o que mostra a execução destes dois comandos):

```
cd AdditionalFiles
```

```
dump value GreyImage.dat GreyImage
```

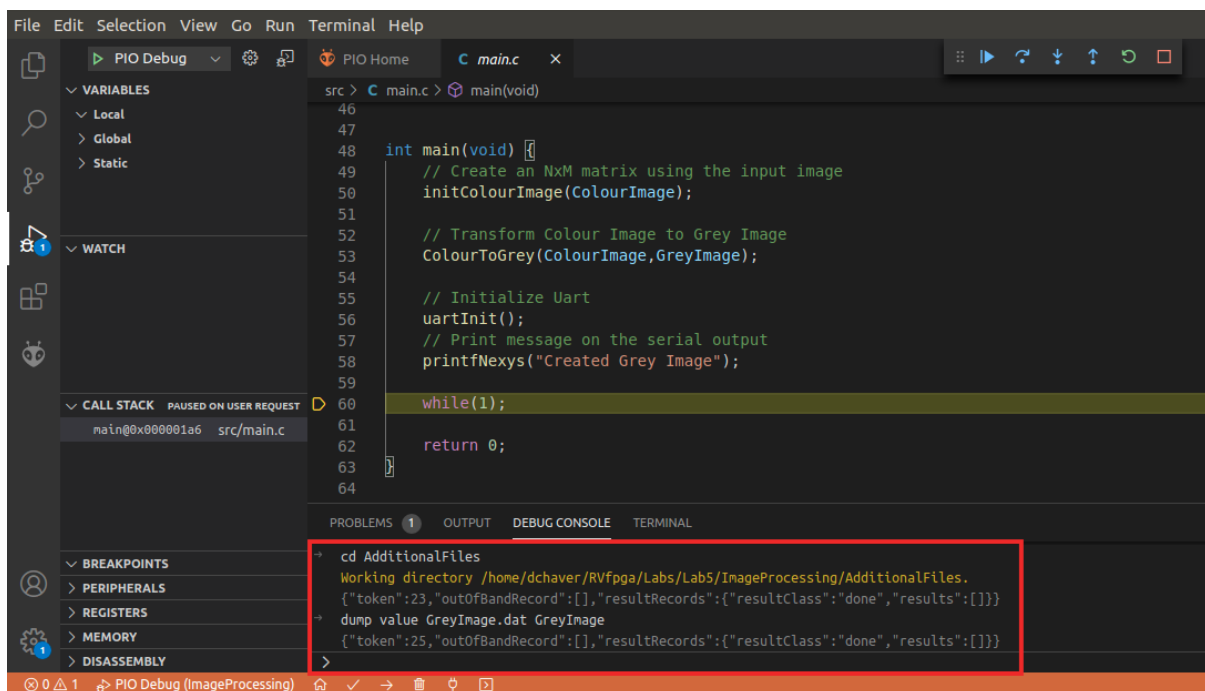


Figura 4. Exportar a imagem em escala de cinzentos para um ficheiro

- Transformar o ficheiro .dat em .ppm que pode visualizar no seu sistema.

Em **LINUX**: fazer isto abrindo um terminal e escrevendo os comandos (ver Figura 5):

```
cd [RVfpgaPath]/RVfpga/Labs/Lab04/ImageProcessing/AdditionalFiles

gcc -o dump2ppm dump2ppm.c

./dump2ppm GreyImage.dat GreyImage.ppm 128 128 1
```

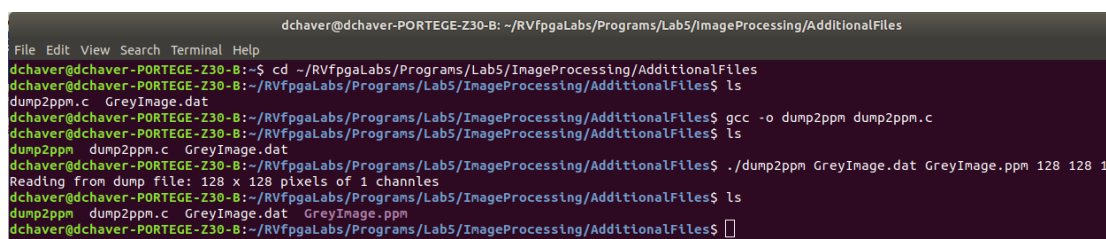


Figura 5. Transformar a imagem para o formato .ppm

Em WINDOWS: fazer isto via:

1. Usando o executável `dump2ppm.exe` fornecido in `[RVfpgaPath]\RVfpga\Labs\Lab04\ImageProcessing\AdditionalFiles`. Abra uma janela de comandos, entre nessa pasta, e execute o executável com os mesmos argumentos que acima:

```
dump2ppm.exe GreyImage.dat GreyImage.ppm 128 128 1
```

Ou

2. Usando o Cygwin (se o instalou como descrito no Guia de Introdução ao RVfpga) para compilar o programa `dump2ppm.c`. Depois executar o programa (`dump2ppm.exe`) no terminal Cygwin ou numa janela de comandos, como na opção 1 acima.

9. Abra o ficheiro `.ppm` usando o GIMP, o Programa de Manipulação de Imagens da GNU (GNU Image Manipulation Program). Se esse programa ainda não estiver instalado, vá para o seguinte website para descarregar o instalador:

<https://www.gimp.org/downloads/>

A imagem em escala de cinzentos deve parecer-se com a imagem mostrada no lado direito de Figura 1 (pode também aceder à imagem a cores de entrada em `[RVfpgaPath]/RVfpga/Labs/Lab04/ImageProcessing/AdditionalFiles/VanGogh_128.ppm`, que deve parecer-se com a que aparece no lado esquerdo de Figura 1).

3. Exercícios

Exercício 1. Execute o programa com uma imagem de entrada diferente. Pode utilizar a imagem fornecida em:

`[RVfpgaPath]/RVfpga/Labs/Lab04/ImageProcessing/src/TheScream_256.c` (Pode ver a imagem `.ppm` correspondente em:

`[RVfpgaPath]/RVfpga/Labs/Lab04/ImageProcessing/AdditionalFiles/TheScream_256.ppm`. Também irá criar esta imagem, executando o programa `dat2ppm`, como descrito anteriormente.)

Exercício 2. Crie uma função C que conte o número de elementos próximos do branco (>235) e próximos do preto (<20) na imagem em escala de cinzentos *VanGogh*. Imprima os dois números na consola de série utilizando as bibliotecas PSP e BSP da Western Digital, como explicado na Secção 3 do Lab 3.

Exercício 3. Transforme a sub-rotina Assembly **ColourToGrey_Pixel** numa função C, e a função C **ColourToGrey** numa sub-rotina Assembly que invoca a função **ColourToGrey_Pixel** C.

- Em C, todas as funções e variáveis globais são exportadas por omissão como símbolos globais, pelo que pode usar a função *ColourToGrey_Pixel* na sub-rotina *ColourToGrey*.
- Para aceder a uma matriz em linguagem Assembly, deve calcular o endereço de um elemento (i,j) , dado o endereço inicial da matriz. De acordo com a norma ANSI C, as matrizes bidimensionais são armazenadas em memória por filas. Assim, o endereço do pixel na linha i e coluna j é obtido adicionando o endereço inicial da matriz e o offset $(i*M + j)*B$, onde M é o número de colunas e B é o número de bytes ocupados por cada pixel: três bytes em RGB e apenas um em escala de cinzentos.

Exercício 4. Aplique um filtro de desfocagem (**Blur Filter**) à imagem a cores *VanGogh* (pode encontrar muita informação online; por exemplo, pode utilizar a informação disponível em: https://lodev.org/cgtutor/filtering.html#Find_Edges).

Note que para transformar a imagem `.dat` numa imagem `.ppm` é necessário modificar um pouco a invocação do comando `dump2ppm` para considerar 3 canais em vez de apenas 1:

```
./dump2ppm FilterColourImage.dat FilterColourImage.ppm 128 128 3
```

Além disso, é possível comparar a imagem filtrada com a original, que está disponível em:

`[RVfpgaLabsPath]/RVfpgaLabs/Programs/Lab04/ImageProcessing/AdditionalFiles/VanGogh_128.ppm`