Department of Computer Engineering
Faculty of Engineering, University of Peradeniya

CO224

Computer Architecture

2017

Project 1

Worth = 8%
Deadline = 01/09/2017 11.55 pm (late submissions will attract 20% penalty per day they are late)
This is a group project. However, we expect that both members will equally work on this.

Description

Image processing is a field where a digitized image is analyzed to obtain information or manipulated to improve the quality of the image. In image processing, the images are represented as a matrix. The RGB values of each pixel is represented by an element of the matrix. (i.e. A single pixel is represented using three values. Each of these values must be between 0-255)
However, a grayscale image is represented using a single value per pixel.
Hence for simplicity,

You'll be using only grayscale images
Instead of actual images we'll be using only the matrix representation :P

There are many operations involved in image processing. Here, we are interested in the following three.

1. Image inversion
2. Rotation180
3. Flip

Figure 1: Original image

1. Inversion

In this operation, the colors of an image is inverted. (Figure 2)

This is done by applying the following formula to the matrix representation of the image

Inverted_value = 255 - original_value

E.g

If the matrix representation of an image A of size 2x3 pixels is

imA = [43 45 123
164 234 12]

then the inverted image's matrix representation will be

imA_inv = 255 - [43 45 123 = [212 210 132
164 234 12] 91 21 243]

Figure 2: Inverted image

2. Rotation180

In this operation, the image is rotated by 180 degrees (Figure 3)
For this the matrix itself must be rotated.

E.g
If the matrix representation of an image A of size 3x4 pixels is

$$imA = \begin{bmatrix} 43 & 45 & 123 & 132 \\ 164 & 234 & 12 & 211 \\ 32 & 121 & 1 & 200 \end{bmatrix}$$

then the rotated image's matrix representation will be

$$rot\_imA = \begin{bmatrix} 200 & 1 & 121 & 32 \\ 211 & 12 & 234 & 164 \\ 132 & 123 & 45 & 43 \end{bmatrix}$$

Figure 3: Rotated image

3. Flip

In this operation, the image is flipped along the vertical axis (Figure 4)
For this the right side of the matrix must be switched with the left side.

E.g
If the matrix representation of an image A of size 3x4 pixels is

```
imA = [  43   45  123  132
        164  234   12  211
         32  121    1  200]
```

then the flipped image's matrix representation will be

```
flp_imA = [ 132  123   45   43
            211   12  234  164
            200    1  121   32]
```

Figure 4: Flipped image

Your task is to implement an image processing library consisting of these three simple operations using ARM Assembly.

You should not use any library functions except those for input/output (available in stdio.h)

Program inputs and outputs

The program should take the inputs in the following order:

1. Number of rows in the matrix
2. Number of columns in the matrix
3. Operation code (explained below)
4. Elements of the matrix

The program should give the following output:

1. Type of operation executed
2. Resulting matrix

Operation codes are as below.

Display the original without any change	:	0 (`Original`)
Invert the image	:	1 (`Inversion`)
Rotate the image by 180 degrees	:	2 (`Rotation by 180`)
Flip the image	:	3 (`Flip`)

If any other value is given for the operation code, the following message should be printed and the program must exit.

```
Invalid operation
```

e.g. Input matrix:

imA = [43 45 123 132
164 234 12 211
32 121 1 200]

Provide the inputs as below

3 4 1 43 45 123 132 164 234 12 211 32 121 1 200

The first 2 values denote the number of rows and columns, the 3rd value denotes the operation and the following values are the elements of the matrix

Hence the output should be as follows:

```
Inversion
212 210 132 123
91 21 243 44
223 134 254 55
```

**The string to be printed for each operation is given in the above op code table inside brackets

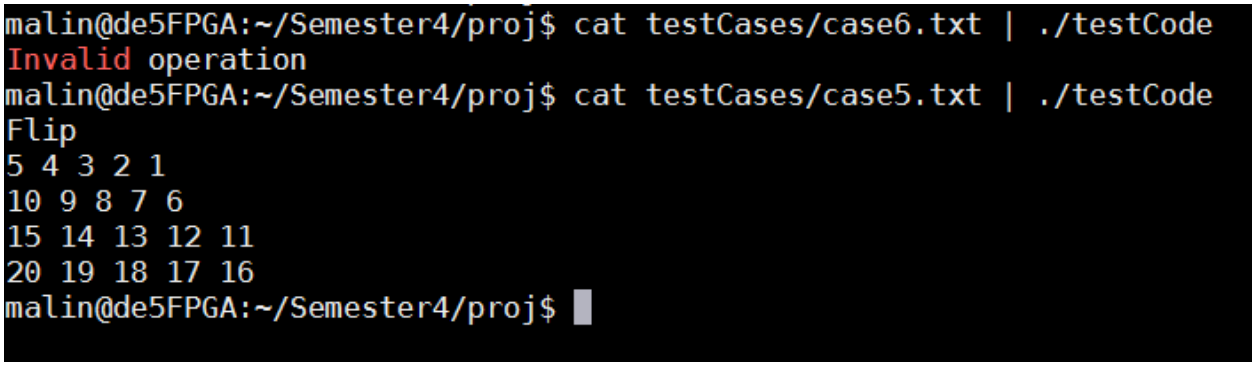

Figure 5: Sample output

Notes

When providing the inputs, put that as a string inside a text file (check the files in the folder testCases and look at figure 5) and pipe it to the program.

e.g. If your input file is inputs.txt

```
$ cat inputs.txt | qemu-arm -L /usr/arm-linux-gnueabi my_prog
```

Testing and Deliverables

By the deadline, you should submit the assembly source code in a single file (.s file) to FEeLS.

File name should be your e-number (e14xxx.s)

Check the given sample input with the sample output, as follows. Make sure it works according to the specification as we are auto marking.

```
$ cat case1.txt | qemu-arm -L /usr/arm-linux-gnueabi ./my_prog >>
myoutput.txt
$ diff sampleoutput.txt myoutput.txt
```

Further, we have provided you the binary file compiled for Intel x86_64 if you need to generate more test cases. (testCode)

Hints

- Break down the algorithms into functions, implement each function and test each function separately.
- How you implement or code is not a problem as long as you get the correct output.

Important

- This is a project and hence things are not straight forward. You will have to do some self-learning plus thinking as things are like that in real world.
- Assembly code debugging would take time and hence start early.
- Ask questions or doubts in the forums in FEeLS (Facebook and email is also okay but do not become a nuisance :P)
- Don't try to become smart by using gcc to convert from C to assembly or using obj-dump to convert the binary to assembly. We know how to catch them and you will get 0 marks.
- Don't copy from anyone. If caught everyone involved would get 0 marks.

HAPPY CODING!