COMP 8551
Fall 2016 – Instructor: Borna Noureddin
Assignment #3

<u>All work should be done in pairs.</u>

Total marks: 125

a) [35 marks] Start with the SomeNEONTests.zip XCode project. The app lets the user select a photo from their photo library and converts it to a grayscale image. The convertToGrayscale() function performs the grayscale transformation in C++. Implement the convertToGrayscaleNEON_intrinsics() function to perform the same function using NEON intrinsics. Note that you must run this code on an ARM-processor-equipped device, such as an iPod touch, iPhone or iPad. The simulators use an x86 instruction set, and the code will not compile if targeted for the simulator. How does the performance of the NEON code compare with the C++ code?

b) [25 marks] Now implement the convertToGrayscaleNEON_asm() function to perform the same function using NEON assembly code instructions. How does the performance compare with the C++ code and the NEON intrinsics code? Which is the most efficient?

c) [40 marks] Create an OpenGL ES app that performs a simple transformation on a cube or other simple object (you can use one of your previous assignments). Implement the matrix multiplication code in ARM NEON code (you can use intrinsics or assembly code). Compare the performance of the NEON code with the GLKMatrix4Multiply() code. Can you achieve better performance? If so, by how much, and if not, why not?

d) [10 marks] Write a windows application that reads in a colour image and a "kernel" image. The kernel image should be much smaller (e.g., $1/10^{th}$ the size of the original image). The user should be provided with a simple user interface (in a window) to specify the two image files and a blending factor. The blending factor is a value between 0 and 1 (or 0% and 100%) that determines how the kernel image is blended with the original image, according to the following:

```
pBlendImg[i][j] = pOrigImg[i][j] * blendFac + pKernelImg[i][j] * (1 -
blendFac);
```

where **pOrigImg** is the original image, **pKernelImg** is the kernel image, **pBlendImg** is the resulting image, and **blendFac** is the blending factor. The code should use a simple for loop to iterate over each pixel in **pOrigImg**. The program should provide the user with a button

to start the blending, then display the original, kernel and blended images, each in its own window. The total time taken to blend the images should be reported to the user where they specified the input parameters (filenames, blending factor, etc.).

e) [8 marks] Write two additional functions to perform the blending, providing the user with the option to select either of these methods over the two you have already provided above. The first function should use MMX intrinsics (see http://msdn.microsoft.com/en-us/library/kcwz153a(v=vs.80).aspx, http://concatenative.org/wiki/view/SSE, and the tutorial and sample code at http://www.softkam.ro/index.php?pid=alphablend). The second function should use SSE intrinsics. The user should be given the ability to choose from either of these methods, or the one above.

f) [3 marks] You should now have four different ways of performing the blending, each written in its own function. Run each one on the same image and kernel (e.g., the ones provided), and report on the difference (if any) in the execution time using each method. Is there any difference? Why or why not? How does the disassembled code of each one compare to the other? Remember that you can step through your code even with optimization, and right-click to show the disassembly code.

g) [2 marks] Write a simple function that takes an object as an argument by value. Write code to call the function and show the disassembly code associated with the function call. Now write the same function, but have it take the argument by reference, pass the object by reference and show the new disassembly code. What is the difference?

h) [2 marks] Write a simple function that takes no arguments and performs a simple math function. Show the disassembly code associated with the function. Now make the function inline and show the new disassembly code. What is the difference?

The code must be written in Objective-C, C, C++ or ARM/NEON assembly instructions for questions 1-3, and C++ or Intel MMX/SEE assembly instructions for questions 4-8. All files required to build and deploy the app must be provided. Submit all your project files and any documentation to the D2L dropbox folder as a single ZIP file with the filename A00ABC_A00XYZ_Asst4.zip, where ABC and XYZ are your A00 numbers. All required documentation (README file, code comments, etc.) must be included.