CECS 525 MICROCOMPUTER SYSTEMS DESIGN
PROJECT 3
Vector Floating Point unit (VFP11)

Yellow highlight points out lab related action required from the student.
Green highlight points out report related action required from the student.
Blue highlight emphasizes certain terms and information.

Caution: Be aware of electro-static discharge or ESD. ESD can weaken and or destroy electonic devices. Use the anti-static mats and wrist straps when handling components, circuits, boards, etc… Winter is particularly bad due to dry air, and Winter clothes.

Project 3 is designed to introduce the student to the high-speed floating-point math coprocessor found in modern microcomputer systems. Moreover, bare-metal Assembly code is used to understand and implement the math coprocessor hardware. The ARM based Raspberry PI (RPI) microcomputer includes a VFP11 math coprocessor that supports the IEEE-754 specification. The VFP11 can subtract, add, multiply, divide, and can find the square root of single and double precision floating-point numbers. And of course, more advanced mathematical operations can be derived. The VFP11 handles monadic and dyadic operations. Three pipelines do all the work: Multiply and Accumulate (FMAC), Load/Store(LS), Divide and Square Root(DS). The ARM core and the VFP11 process their own unique instructions independently and in parallel thereby increasing the overall system performance. Ultimately, the math coprocessor, is exploited for a large number of scientific and commercial applications/systems. High definition CAD drawings that took an hour to render without a math coprocessor now render in a few seconds.

In project 3, each group will exploit the VFP11, in bare-metal fashion, to advance their command-line calculator and by using single-precision floating-point operations.

Part 1: In main.c, engineer and add two C sub-routines to the math calculator. One routine is to convert signed floating-point numbers entered as ASCII on the command line into IEEE-754 single-precision floating-point numbers represented as 32-bit unsigned integers. The second routine performs the opposite function converting single-precision floating-point numbers to ASCII for display on the command line. With these two routines as part of the calculator's C code, floating-point numbers entered by the user can be sent to the VFP11 and the result from the VFP11 can be displayed. Reference the IEEE754.pdf document. In your report, discuss both conversion routines and how they work.

Example user entered floating-point numbers converted to 32-bit unsigned integers...

-0.2314 converts to 0xBE6CF41F

-23.89802 converts to 0xC1BF2F25

345.894 converts to 0x43ACF26F

1.0 converts to 0x3F800000

0.0 converts to 0x00000000

Part 2: In boot.s, engineer an assembly VFP11 driver (reference ARM1176JZF-S.pdf, IEEE754.pdf, VFP11.pdf and the ARM info center). The VFP11 driver is used by the calculator application in main.c as the interface to the VFP11 coprocessor. The VFP11 driver needs input from the calculator application such as the signal-precision floating-point number(s) and the math operation. The calculator application needs the single-precision floating-point result from the VFP11 via the driver to display on the command line. Exception handling code is also required as errors may occur such as NaN. Moreover, exception processing is necessary in order to prevent the system from crashing or locking up. The math app should successfully add, subtract, multiply, and divide two single-precision floating-point numbers. The math app should negate a single-precision floating-point number. The math app should find the square root of a single-precision floating-point number. Arithmetic exceptions such as NaN should be reported to the user via the command line. In the report, discuss your VFP11 driver and how it works to operate the VFP11 coprocessor according to the input from the user and calculator app. In your report, discuss how the arithmetic exceptions are processed by your code to prevent a system crash and to inform the user of an error.