Lab #2 – Unsigned arithmetic

Goals:

Practice binary and decimal conversion and binary arithmetic

Write a simple program on the Discovery Board using MOVS and ADDS instructions

Reference: 2^0=1 2^1=2 2^2=4 2^3=8 2^4=16 2^5=32 2^6=64 2^7=128 2^8=256 2^9=512 2^{10}=1024 2^{11}=2048 2^{12}=4096 2^{13}=8192 2^{14}=16384 2^{15}=32768 2^{16}=65536 ...

Part 1: Refer to the Week 2 lecture slides for the following:

a) Go to https://smumustangs.com/sports/football/stats/2021 and look up the total number of points scored by both the Mustangs and by their opponents last season. Enter into the Canvas text box these two numbers, in decimal.

b) Convert each decimal number to binary, using 12 bit fields, and enter into Canvas.

c) Find the total number of points scored by summing the numbers using 12-bit binary arithmetic, and enter into Canvas.

d) Convert the sum back into decimal, and enter into Canvas.

e) Compare your result from d) with the sum of the two decimal numbers in a)

Part 2:

a) Study the example programs on pages 29-30 of the lecture notes for Week 2. How do the function arguments get passed from the C code into the assembly function code? Where is the function return value expected?

b) Download the template main C program Lab02-SumMain.c from the Canvas files area, and in the LabCode folder:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include "library.h"

extern uint32_t sum1(uint32_t,uint32_t,uint32_t,uint32_t);
extern uint32_t sum2(uint32_t,uint32_t,uint32_t,uint32_t);

int main(void) {
  uint32_t a,b,c,d,r;

  InitializeHardware(HEADER,"Move and Add");
  a = ;  // Complete this statement
  b = ;  // Complete this statement
```

```
    c = ;  // Complete this statement
    d = ;  // Complete this statement

    printf("sum = %lX\n",a + b + c + d + 0x480C);
    r = sum1(a,b,c,d);
    printf("sum1() return value = %lu\n",r);
    r = sum2(a,b,c,d);
    printf("sum2() return value = %lu\n",r);
}
```

Complete the four C statements to assign the hexadecimal values 0xB620, 0x39C4, 0x571A, and 0x41F8 to unsigned integer variables a, b, c, and d, respectively.

c) Download and edit the general framework of the assembly code in the file Lab02-sum.s available in the Canvas files section. Locate label statements sum1: and sum2:, the entry points for two functions coded in assembly language, and the two 'bx lr' instructions that return execution from each function back to the calling program. You will be filling in assembly instructions between the entry points and the returns for two functions. Write the assembly code in both functions sum1 and sum2 to compute the sum of all four unsigned integer function arguments, plus an additional fifth constant of 0x004C. Include the 'S' suffix on all MOV and ADD instructions to update the conditional flags after each instruction. First write the code in sum1 assuming the only addition instruction available on the processor is 'ADDS R0, R1'. Hint: This will require the use of some MOVS instructions, in addition to the ADDS instructions. Then write the code in sum2 using any ADDS instruction discussed in the lecture, but *no MOVS instructions*. The computed sum must be returned to the calling program by both the sum1 and sum2 functions as the function's value. **Upload your two completed source code files Lab02-SumMain.c and Lab02-sum.s into the file upload entry in Canvas.**

As needed for debugging, as in the Lab 1B example, insert any 'SVC #0' instructions you would like in either function to trace the register contents.

d) Now change the four constants in the C main program assigned to the variables a, b, c, and d to be the same value of 0x3FFFFFED. This constant is figured so that when all four arguments are summed one at a time, and then with the fifth constant 0x004C, in the two assembly functions sum1 and sum2, the total will increase and just reach back around to the overflow value of 0x00000000. Rebuild and rerun the program on your board. You should see the totals from both functions report a return value of 0x0.

e) Now edit the Lab02-sum.s assembly source code and insert an 'SVC #0' debug breakpoint instructions just before and after the last 'ADDS' instruction in function sum2. So now the last lines in the sum2 function should read like

```
    svc       #0        // Debug breakpoint
    adds ......
    svc       #0        // Debug breakpoint
    bx   lr             // Return to calling program
```

As you press the blue button on your board, you should see the highest order hex digit reported in the debug window for the xPSR register change when that last ADDS instruction is run. Report the two xPSR register values, before and after the last ADDS, in Canvas. Also record in Canvas what condition

bits are changed in the xPSR register (see lecture slide 20). Hint: Among other changes, you should see the carry bit turn to ‘1’ to report the unsigned arithmetic overflow.