#cec32xNotes, cec32x_lab_28_Lab4–C_and_V_flags

Lab 5. The C and V Flags of an ARM MCU

Introduction

We have learned in the class that **an ARM processor does not distinguish between the unsigned and signed numbers in the ALU during the addition and subtraction operations**. It is the programmer's responsibility to interpret the numbers as unsigned or signed. With the two's complement expression, a negative signed number `A` is expressed as an unsigned number `B = A + 2^n`, where `n` is the number of bits of the number expression. Again, this is the programmers' responsibility for the interpretation. To help the programmers check if the addition/subtraction operations are correct in both the unsigned and signed forms, the ARM processor uses two flags simultaneously—the C flag for unsigned integers and the V flag for signed integers.

In this lab, we simulate an 8-bit processor following the C/V flag setting rules of an ARM processor. Note that we use the 8-bit system since the numbers are small enough to permit us doing easy verification by hand.

Lab Tasks

There are three tasks in this lab. Task 1 is to generate the numbers and reserve the spaces for the next two tasks, Task 2 is to simulate the addition and subtraction operations of an 8-bit ARM processor by writing code, and Task 3 verifies Task 1 using the calculations of an ARM MCU (using the simulator in Keil).

Please use the following code snippet to facilitate collaboration between team members:

```c
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>

void convert_uint8_number_to_binary_str(uint8_t num, char *cPtr);
void print_str_verbose(uint8_t num, char *cPtr_c, char *cPtr_v);
uint8_t sub_uint8(uint8_t x0, uint8_t x1, bool *c_flg);
int8_t sub_int8(int8_t x0, int8_t x1, bool *v_flg);

#define P 8 // the number of pairs of data to be operated on
#define N 8 // number of bits of the computer
#define MIN_UN 0                     // min value of unsigned N-bit num
#define MAX_UN ((1 << N) - 1)        // max value of unsigned N-bit num
#define MIN_IN (-(1 << (N-1)) )       // min value of signed N-bit num
```

```c
#define MAX_IN ((1 << (N-1)) - 1) // max value of signed N-bit num

char str_c[30], str_v[30];
int8_t x[P][2];
uint8_t ru[P];
int8_t ri[P];
bool c[P], v[P];

int main(void) {
    for (int i = 0; i < P; i++) {
        x[i][0] = rand() % MAX_UN + MIN_IN;
        x[i][1] = rand() % MAX_UN + MIN_IN;
        printf("x[%d] = %d, %d \n", i, x[i][0], x[i][1]);

        ru[i] = sub_uint8(x[i][0], x[i][1], &c[i]);
        printf("result_u = %d, C flag = %d\n", ru[i], c[i]);
        print_str_verbose(ru[i], str_c, str_v), printf("\n");

        ri[i] = sub_int8(x[i][0], x[i][1], &v[i]);
        printf("result_i = %d, V flag = %d\n", ri[i], v[i]);
        print_str_verbose(ri[i], str_c, str_v), printf("\n");

        int temp0 = (x[i][0] << (32-N));
        int temp1 = (x[i][1] << (32-N));
        int temp2 = temp0 - temp1;
        int temp = temp2 >> (32-N);
        printf("ARM result = %d\n", temp);
    }

    while (1);
}
```

There are three tasks in this lab.

(20 points) Task 1. Modify the code of a Workshop to program the following functions, which will be used in the display of the binary results of addition/subtraction operations later on:

```c
void convert_uint8_number_to_binary_str(uint8_t num, char *cPtr);
void print_str_verbose(uint8_t num, char *cPtr_c, char *cPtr_v);
```

Make sure these functions work correctly before moving forward to the next task.

(40 points) Task 2. Program the following functions to produce “correct” results according to an 8-bit processor and determine corresponding C and V flags. Assume the 8-bit

processor follows the flag-setting principles of an ARM processor:

```
// return x0 - x1
uint8_t sub_uint8(uint8_t x0, uint8_t x1, bool *c_flg);
// return x0 - x1
int8_t sub_int8(int8_t x0, int8_t x1, bool *v_flg);
```

Note that you need to take screenshots for the results of the above functions.

(10 points) Task 3. Verify the C and V flags of the above functions using the results from a true ARM processor. To this end, we can use the provided code below:

```
int temp0 = (x[i][0] << (32-N));
int temp1 = (x[i][1] << (32-N));
int temp2 = temp0 - temp1;
int temp = temp2 >> (32-N);
```

The work for this task include the following:

- Setup a breakpoint in the last line of the above. This way, we can halt the processor immediately after it executes the subtraction.
- At each halt, open up `xPSR` to check the values of C and V, as shown in the figure below. They should be the same as what you obtained in Task 2.
- Take screenshots showing the C/V flags in `xPSR`, the results provided by the ARM MCU.

Submission of Lab Report

- (10 points) For the report format/quality.
- (10 points) For explaining the specific operations to generate `x[i][0]` and `x[i][1]`.
- (10 points) For providing two screenshots to show the consistency of the results from your Task 2 and Task 3. Each of the screenshots should show all of the following: the C/V flags in `xPSR`, the printouts of C/V flags in the debug window, and the print out of the calculation results in the debug window.