Wiki at erau.us

Workshop 2

We have learned in class that an ARM processor does not distinguish between unsigned and signed numbers when performing additions and subtractions—it's the programmer's responsibility to interpret the numbers as unsigned or signed. To help the programmers, the processor provides both the C and V flags—C flag for unsigned operations and V flag for signed ones.

In this workshop, we simulate a 5-bit system using an ARM MCU. Note that we use the 5-bit system since the numbers are small enough to do easy verifications.

With the combination of unsigned/signed and addition/subtraction, we have four cases. In this workshop, we also used the idea of a state machine to track the state of the calculation. The four states and their functionality are as follows in the code snippet:

```
// Prototype of FOUR functions, each for a STATE.
// The func in State 1 performs addition of "unsigned numbers" x0 and x1.
int s1_add_uintN(int x0, int x1, bool *c_flg);
// The func in State 2 performs addition of "signed numbers" x0 and x1.
int s2_add_intN(int x0, int x1, bool *v_flg);
// The func in State 3 performs subtraction of "unsigned numbers" x0 and x1.
int s3_sub_uintN(int x0, int x1, bool *c_flg);
// The func in State 3 performs subtraction of "signed numbers" x0 and x1.
int s4_sub_intN(int x0, int x1, bool *v_flg);
```

Note that we also update the C or V flag according to the state.

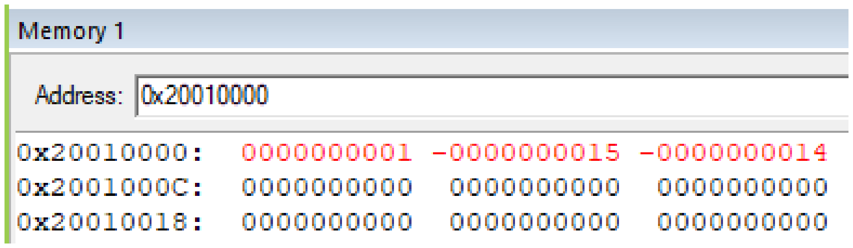
We use the Keil simulator to run the code. To be able to easily modify the numbers, we change the contents of memories directly. To this end, we need to make sure the memory space is available to the processor and is not used by the system. See the class demo for the setup of the memory address. The enclosed project files contain the setup already. Also enclosed is the source file which is shown below. To change the contents of the memory, we need to halt the program using a breakpoint; add one in a `printf` statement at the beginning of the `while (1)` loop.

Please read the source code below carefully to extract the requirements from the comments and printout messages.

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

// Prototype of FOUR functions, each for a STATE.
// The func in State 1 performs addition of "unsigned numbers" x0 and x1.
int s1_add_uintN(int x0, int x1, bool *c_flg);
// The func in State 2 performs addition of "signed numbers" x0 and x1.
int s2_add_intN(int x0, int x1, bool *v_flg);
// The func in State 3 performs subtraction of "unsigned numbers" x0 and x1.
int s3_sub_uintN(int x0, int x1, bool *c_flg);
// The func in State 3 performs subtraction of "signed numbers" x0 and x1.
int s4_sub_intN(int x0, int x1, bool *v_flg);

// We define the number of bits and the related limits of unsigned and
// and signed numbers.
#define N 5         // number of bits
#define MIN_U 0     // minimum value of unsigned N-bit number
#define MAX_U ((1 << N) - 1)   // maximum value of unsigned N-bit number
#define MIN_I (-(1 << (N-1)) ) // minimum value of signed N-bit number
#define MAX_I ((1 << (N-1)) - 1)   // maximum value of signed N-bit number

// We use the following three pointers to access data, which can be changed
// when the program pauses. We need to make sure to have the RAM set up
// for these addresses.
int *pIn = (int *)0x20010000U;   // the value of In should be -1, 0, or 1.
int *pX0 = (int *)0x20010004U;   // X0 and X1 should be N-bit integers.
int *pX1 = (int *)0x20010008U;

int main(void) {
    enum progState{State1 = 1, State2, State3, State4};
    enum progState cState = State1;         // Current State
    bool dataReady = false;
    bool cFlg, vFlg;
    int result;

    while (1) {
        dataReady = false;
        // Check if the data are legitimate
        while (!dataReady) {
            printf("Halt program here to provide correct update of data\n");
            printf("In should be -1, 0, and 1 and ");
            printf("X0 and X1 should be N-bit SIGNED integers\n");
            if (((-1 <= *pIn) && (*pIn <= 1)) &&
                    ((MIN_I <= *pX0) && (*pX0 <= MAX_I)) &&
                    ((MIN_I <= *pX1) && (*pX1 <= MAX_I))) {
                dataReady = true;
            }
        }
        printf("Your input: In = %d, X0 = %d, X1 = %d \n", *pIn, *pX0, *pX1);

        switch (cState) {
          case State1:
            result = s1_add_uintN(*pX0, *pX1, &cFlg);
            printf("State = %d, rslt = %d, Cflg = %d\n", cState, result, cFlg);
            cState += *pIn;
            if (cState < State1) cState += State4;
            break;
          case State2:
            result = s2_add_intN(*pX0, *pX1, &vFlg);
            printf("State = %d, rslt = %d, Vflg = %d\n", cState, result, vFlg);
            cState += *pIn;
```

```
              break;
            case State3:

            case State4:

            default:
              printf("Error with the program state\n");
          }
        }
    }

    int  s1_add_uintN(int x0, int x1, bool *c_flg) {
        if (x0 < 0) x0 = x0 + MAX_U + 1;
        if (x1 < 0) x1 = x1 + MAX_U + 1;
        int temp = x0 + x1;
        if (temp > MAX_U) {
            temp = temp - (MAX_U + 1);
            *c_flg = true;
        } else {
            *c_flg = false;
        }
        return temp;
    }

    int  s2_add_intN(int x0, int x1, bool *v_flg) {
        int temp = x0 + x1;
        if (temp < MIN_I) {
            temp = temp + MAX_U + 1;
            *v_flg = true;
        } else if (temp > MAX_I) {
            temp = temp - (MAX_U + 1);
            *v_flg = true;
        } else {
            *v_flg = false;
        }
        return temp;
    }

    int  s3_sub_uintN(int x0, int x1, bool *c_flg) {
        return 0;
    }

    int  s4_sub_intN(int x0, int x1, bool *v_flg) {
        return 0;
    }
```

The running results with the following input

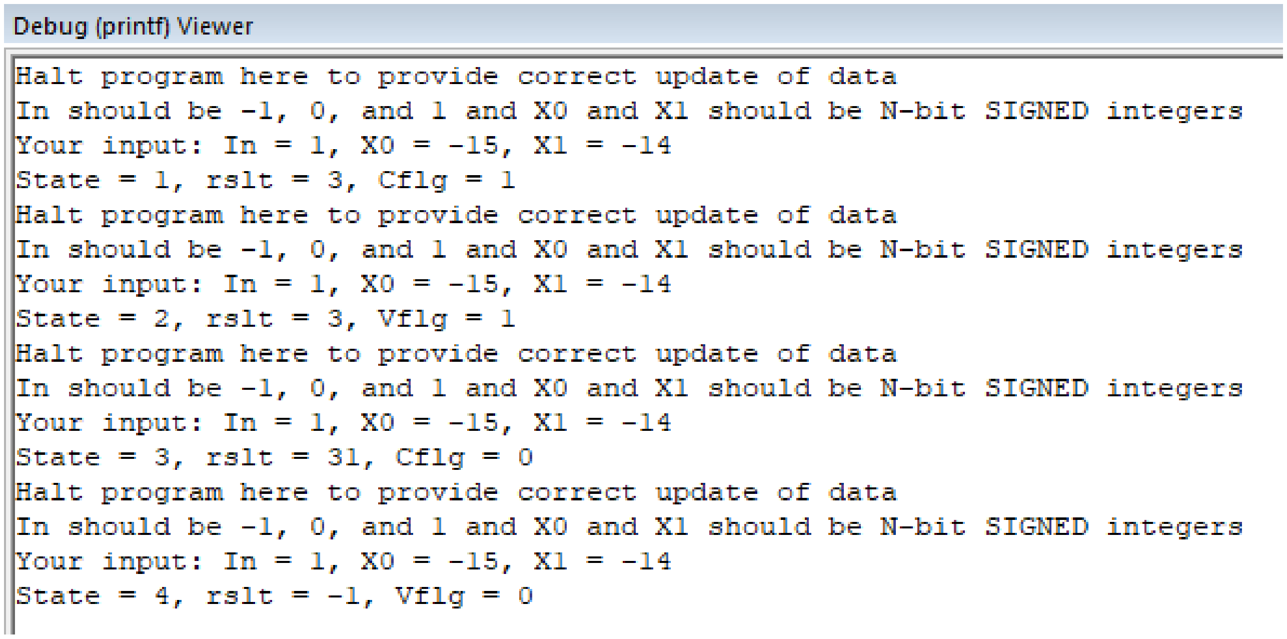


Memory 1

Address: 0x20010000

```
0x20010000:  0000000001 -0000000015 -0000000014
0x2001000C:  0000000000  0000000000  0000000000
0x20010018:  0000000000  0000000000  0000000000
```

is given below.


Debug (printf) Viewer

```
Halt program here to provide correct update of data
In should be -1, 0, and 1 and X0 and X1 should be N-bit SIGNED integers
Your input: In = 1, X0 = -15, X1 = -14
State = 1, rslt = 3, Cflg = 1
Halt program here to provide correct update of data
In should be -1, 0, and 1 and X0 and X1 should be N-bit SIGNED integers
Your input: In = 1, X0 = -15, X1 = -14
State = 2, rslt = 3, Vflg = 1
Halt program here to provide correct update of data
In should be -1, 0, and 1 and X0 and X1 should be N-bit SIGNED integers
Your input: In = 1, X0 = -15, X1 = -14
State = 3, rslt = 31, Cflg = 0
Halt program here to provide correct update of data
In should be -1, 0, and 1 and X0 and X1 should be N-bit SIGNED integers
Your input: In = 1, X0 = -15, X1 = -14
State = 4, rslt = -1, Vflg = 0
```

You are supposed to do the assignment in teams of two members. Your task is to finish the coding for States 3 and 4 so that the state machine as the calculations work correctly. Below is the point distribution of the assignment.

- 10 points each for correct handling in States 3 and 4.
- 35 points each for the contents of the last two functions.
- 10 points for following the assignment/submission requirements:
  - You need to submit a renamed version of the `main.c` file of the project. Use this convention: `cec320_sec_x_ws2_lastname1_firstname1__lastname2_firstname2.c`, where `x` is your section number.
  - You need also submit a pdf file which contains the screenshots of your C functions and the screenshot of the running results. The running results should be like the one given above but with the names of you and your team member displayed clearly. Use the same naming convention as above but with a suffix of `pdf` for this file.

cls/arm/workshop/workshop2.txt · Last modified: 2019/02/09 16:43 by Jianhua Liu [/wiki/user/jhl]