CEC322 - Section 2

Lab 4 - The C and V Flags of an ARM MCU

Lab start date: 02/13/19

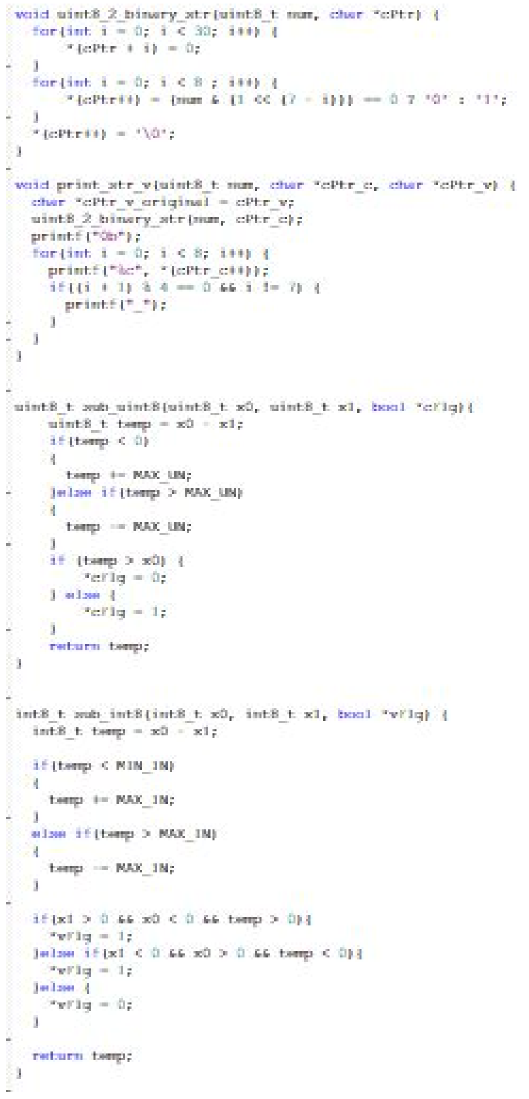
Report data: 02/20/19

Andrew Henderson and Tyler Wise

Introduction:

This lab's purpose is to illustrate to students one of the limitations of working with an ARM microprocessor, the lack of distinction by the ALU between signed and unsigned numbers. To still have the same effect, the microprocessor provides the C and V flags that students can set and utilize in their logic. This lab will show students how to set these flags so that they can be properly leveraged.

Code Snippet:



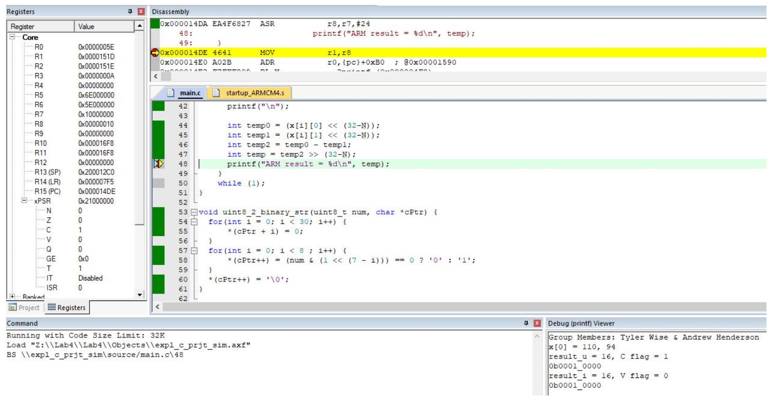
```c
void uint8_2_binary_str(uint8_t num, char *cPtr) {
  for(int i = 0; i < 30; i++) {
      *(cPtr + i) = 0;
  }
  for(int i = 0; i < 8 ; i++) {
      *(cPtr++) = (num & (1 << (7 - i))) == 0 ? '0' : '1';
  }
  *(cPtr++) = '\0';
}

void print_str_v(uint8_t num, char *cPtr_c, char *cPtr_v) {
  char *cPtr_v_original = cPtr_v;
  uint8_2_binary_str(num, cPtr_c);
  printf("0b");
  for(int i = 0; i < 8; i++) {
    printf("%c", *(cPtr_c++));
    if((i + 1) % 4 == 0 && i != 7) {
      printf("_");
    }
  }
}


uint8_t sub_uint8(uint8_t x0, uint8_t x1, bool *cFlg){
    uint8_t temp = x0 - x1;
    if(temp < 0)
    {
      temp += MAX_UN;
    }else if(temp > MAX_UN)
    {
      temp -= MAX_UN;
    }
    if (temp > x0) {
       *cFlg = 0;
    } else {
       *cFlg = 1;
    }
    return temp;
}


int8_t sub_int8(int8_t x0, int8_t x1, bool *vFlg) {
  int8_t temp = x0 - x1;

  if(temp < MIN_IN)
  {
    temp += MAX_IN;
  }
  else if(temp > MAX_IN)
  {
    temp -= MAX_IN;
  }

  if(x1 > 0 && x0 < 0 && temp > 0){
    *vFlg = 1;
  }else if(x1 < 0 && x0 > 0 && temp < 0){
    *vFlg = 1;
  }else {
    *vFlg = 0;
  }

  return temp;
}
```

Discussion:

In order to generate the values of x[i][0] and x[i][1] for each value of i, a random integer is generated by rand(). In order to cap this value to the maximum value of an unsigned integer as defined by the number of bits we are using, the random value is modulated by the maximum value of an unsigned integer for the number of bits, mapping the value to the range [0, MAX_UN). This is then offset by the minimum value of a signed integer to remap the values a final time to include negative values while also lowering the upper bound to match.

Results:

To verify the correct operation of our code, the provided snippet at the end of the for loop is used. The logic behind the calculation is also fairly simple. In order to have all of the bits in the same significant place before performing the subtraction as we do when performing it manually for ourselves, we must shift the bits over as can be seen in lines 38 and 39. Then, the actual subtraction occurs which should yield the same resulting C and V flags that we manually calculated. Then, the result of the subtraction can be shifted back to the right to get the true result of the calculation while also preserving the computed C and V flags in line 41. To verify that our code snippets operated correctly, the images below can be used.



Registers

```
Register      Value
Core
  R0          0x0000005E
  R1          0x0000151D
  R2          0x0000151E
  R3          0x0000000A
  R4          0x00000000
  R5          0x6E000000
  R6          0x5E000000
  R7          0x10000000
  R8          0x00000010
  R9          0x00000000
  R10         0x000016F8
  R11         0x000016F8
  R12         0x00000000
  R13 (SP)    0x200012C0
  R14 (LR)    0x000007F5
  R15 (PC)    0x000014DE
  xPSR        0x21000000
    N         0
    Z         0
    C         1
    V         0
    Q         0
    GE        0x0
    T         1
    IT        Disabled
    ISR       0
Banked
```

Disassembly

```
0x000014DA EA4F6827  ASR          r8,r7,#24
    48:                     printf("ARM result = %d\n", temp);
    49:     }
0x000014DE 4641      MOV          r1,r8
0x000014E0 A02B      ADR          r0,{pc}+0xB0  ; @0x00001590
```

main.c

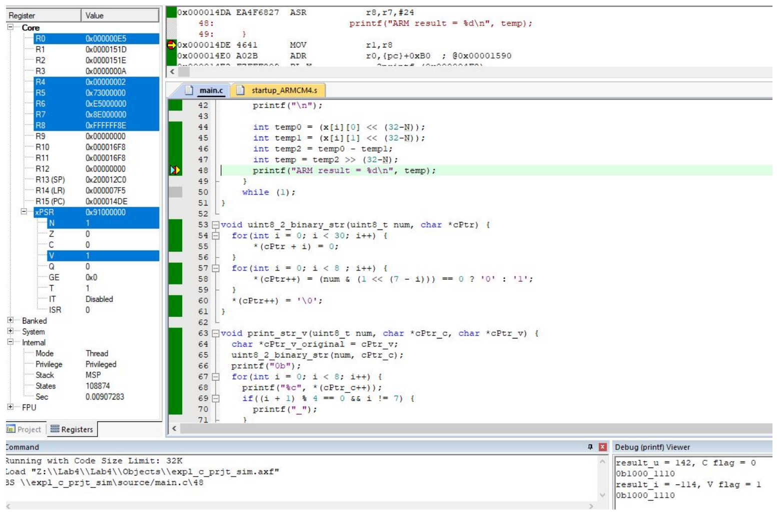
```c
        printf("\n");

        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);
}

void uint8_2_binary_str(uint8_t num, char *cPtr) {
  for(int i = 0; i < 30; i++) {
      *(cPtr + i) = 0;
  }
  for(int i = 0; i < 8 ; i++) {
      *(cPtr++) = (num & (1 << (7 - i))) == 0 ? '0' : '1';
  }
  *(cPtr++) = '\0';
}
```

Command

```
Running with Code Size Limit: 32K
Load "Z:\\Lab4\\Lab4\\Objects\\expl_c_prjt_sim.axf"
BS \\expl_c_prjt_sim\source/main.c\48
```

Debug (printf) Viewer

```
Group Members: Tyler Wise & Andrew Henderson
x[0] = 110, 94
result_u = 16, C flag = 1
0b0001_0000
result_i = 16, V flag = 0
0b0001_0000
```



```
Register      Value
Core
  R0          0x000000E5
  R1          0x0000151D
  R2          0x0000151E
  R3          0x0000000A
  R4          0x00000002
  R5          0x73000000
  R6          0xE5000000
  R7          0x8E000000
  R8          0xFFFFFF8E
  R9          0x00000000
  R10         0x000016F8
  R11         0x000016F8
  R12         0x00000000
  R13 (SP)    0x200012C0
  R14 (LR)    0x000007F5
  R15 (PC)    0x000014DE
  xPSR        0x91000000
    N         1
    Z         0
    C         0
    V         1
    Q         0
    GE        0x0
    T         1
    IT        Disabled
    ISR       0
Banked
System
Internal
  Mode        Thread
  Privilege   Privileged
  Stack       MSP
  States      108874
  Sec         0.00907283
FPU
```

```
0x000014DA EA4F6827  ASR          r8,r7,#24
    48:                     printf("ARM result = %d\n", temp);
    49:     }
0x000014DE 4641      MOV          r1,r8
0x000014E0 A02B      ADR          r0,{pc}+0xB0  ; @0x00001590
```

main.c

```c
        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);
}

void uint8_2_binary_str(uint8_t num, char *cPtr) {
  for(int i = 0; i < 30; i++) {
      *(cPtr + i) = 0;
  }
  for(int i = 0; i < 8 ; i++) {
      *(cPtr++) = (num & (1 << (7 - i))) == 0 ? '0' : '1';
  }
  *(cPtr++) = '\0';
}

void print_str_v(uint8_t num, char *cPtr_c, char *cPtr_v) {
  char *cPtr_v_original = cPtr_v;
  uint8_2_binary_str(num, cPtr_c);
  printf("0b");
  for(int i = 0; i < 8; i++) {
    printf("%c", *(cPtr_c++));
    if((i + 1) % 4 == 0 && i != 7) {
      printf("_");
    }
```

Command

```
Running with Code Size Limit: 32K
Load "Z:\\Lab4\\Lab4\\Objects\\expl_c_prjt_sim.axf"
BS \\expl_c_prjt_sim\source/main.c\48
```

Debug (printf) Viewer

```
result_u = 142, C flag = 0
0b1000_1110
result_i = -114, V flag = 1
0b1000_1110
```