KETTERING UNIVERSITY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CE-426-01

Embedded C & Assembly Programming on Cortex-M3

Darek Konopka

April 29, 2021

Table of Contents

1. Objectives

- Create a new assembly project using uVision
- Learn the standard initialization steps of embedded software
- Perform basic ARM assembly programming

2. Program Source Code

2.1 Lab2.s

```
        AREA subroutines, CODE
        EXPORT findHighBitASM

findHighBitASM
        ; Initialize, check if you really need to initalize
        PUSH {r4-r6, lr} ; store link register
        MOV r3,r0 ; input array
        MOV r4,r1 ; nelements
        MOVS r0,#0x00 ; 32bit number we are looking at (using UXTH we will make this 32bit no worries)
        MOVS r1,#0x00 ; which bit we are looking at
        MOVS r2,#0x00 ; temp array value
        MOVS r5,#0x00 ; which number we are looking at
        ; NOTE: r6 will act as the other half of the 32bit for r0
        ; UXTH will help us do that
```

```
loop
        LDR r2,[r3,r1,LSL #2] ; create temp array value
        CBZ r2,endloop        ; psedo code for if temp != 0 then branch to end of loop
        MOVS r5,#0x00                 ; restore number we are looking at
        B nestedloop2 ; we want to skip the increment if we are at the last element of the array


nestedloop1
        ADDS r6,r5,#1 ; increment the number we are looking at
        UXTH r5,r6              ; make sure to store it as 32bit not 16bits
        LSRS r2,r2,#1   ; right shift temp


nestedloop2
        CMP r2,#0x01; increment pos2
        BHI nestedloop1         ; if the for loop is not done, go back
        ADD r6,r5,r1,LSL #5  ; pos1 = j*32 + pos2;
        UXTH r0,r6              ; r6=other half of r0 to make 32bit number


endloop
        ADDS r6,r1,#1           ; increment r6
        UXTH r1,r6    ; r6 is the other half of r1
        CMP r1,r4       ; compare i of for loop to nelements
        BCC loop ; branch back to beginning


        ; return position
```

```
        POP {r4-r6,pc}
        END
```

2.2 Lab2main.c

```
/*
Darek Konopka & Breanna Krywko
Lab 2
*/
//*********************************************************************************
*******************************
// Author: Girma Tewolde
//   Last modified: 4-9-2020 @5:52 P.M.
// Purpose: Framework for CE-426 Lab 2
//
// Use the array data defined in the global memory space and the main program in this file for
testing
// your two versions, i.e. assembly and C, of the findHighBit function.
// Test each of your two implementations with the given array data and make sure your answers
match
// with the ones provided in the comments for each array.
// You will also compare the two implementations on criteria such as execution speed and size of
// memory used by the corresponding functions.
//*********************************************************************************
*******************************

#include <stm32f10x.h>
```

```
uint32_t array0[] =
{0x00000001,0x00000020,0x00000400,0x00008000,0x00440000,0x02200000,0x12000000,0x8
0000000}; // answer = 255;

uint32_t array1[] =
{0x00000001,0x00000020,0x00000400,0x00008000,0x00440000,0x02200000,0x12000000,0x0
0000000}; // answer = 220;

uint32_t array2[] =
{0x00000001,0x00000020,0x00000400,0x00008000,0x00440000,0x02200000,0x00000000,0x0
0000000}; // answer = 185;

uint32_t array3[] =
{0x00000001,0x00000020,0x00000400,0x00008000,0x00440000,0x00000000,0x00000000,0x0
0000000}; // answer = 150;

uint32_t array4[] =
{0x00000001,0x00000020,0x00000400,0x00008000,0x00000000,0x00000000,0x00000000,0x0
0000000}; // answer = 111;

uint32_t array5[] =
{0x00000001,0x00000020,0x00000400,0x00000000,0x00000000,0x00000000,0x00000000,0x0
0000000}; // answer = 74;

uint32_t array6[] =
{0x00000001,0x00000020,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x0
0000000}; // answer = 37;

uint32_t array7[] =
{0x00000001,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x00000000,0x0
0000000}; // answer = 0;


uint32_t *arrays[] = {array0, array1, array2, array3, array4, array5, array6, array7};

uint32_t narrays = sizeof(arrays)/sizeof(uint32_t*);

uint8_t nelements = sizeof(array0)/sizeof(uint32_t);


uint32_t findHighBitASM(uint32_t*, uint32_t);
```

```
uint32_t findHighBitC(uint32_t*, uint32_t);

int main(void){
    int i;
    volatile int position;
    for (i = 0; i<narrays; i++)
    {
            position = findHighBitASM(arrays[i], nelements);
            //position = findHighBitC(arrays[i], nelements);
    }

    while(1){position++;    } // endless loop to keep micro from crashing
                                                    // position++ keeps position in scope for easier debugging
}

uint32_t findHighBitC(uint32_t* array, uint32_t nelements)
{

    // I used 16bit ints because there is no need to take up so much memory
    uint16_t pos1 = 0; // highest bit position
    uint16_t j = 0; // which 32bit number we are looking at
    uint32_t temp = 0; //temporary array value
    uint16_t pos2 = 0; // which bit we are looking at

    // look though each element of the array
```

```
    for(j = 0; j < nelements; j++) {

        temp = array[j]; // create temp array value

        // If 0 then we skip to the next element in the array
        if (temp != 0) {
            // Here we do the bitshift until we hit 1 to find the bit position in this element
            for(pos2 = 0; temp > 1; temp >>= 1) {
                pos2++;
            }
            // for each number we skiped, we multiply j*32 since they are 32bits
            pos1 = j*32 + pos2;
        }
    }
    // return final position
    return pos1;

}
```

3. Questions

3.1 Question 1

How many bytes of code memory does your assembly function take up? To find out how much memory is used, observe the addresses of the disassembly listing file of your assembly source code when you enter debug mode.

Start: 0x080003C4

End: 0x080003F4

Difference: 0x30 bytes of code are taken up by the assembly function

3.2 Question 2

How many clock cycles are required by the assembly function for each of the 8 inputs?

Each input	Clock Cycles	Differences
Beginning	1703	
1	2829	1126
2	3732	903
3	4433	701
4	4953	520
5	5313	360
6	5562	249
7	5735	173
8	5865	130

Figure 3.2.1

3.3 Question 3

How many bytes of code memory does your C function take up?

Start: 0x0800038C

End: 0x080003C2

Difference: 0x36 bytes of code are taken up by the assembly function

3.4 Question 4

How many clock cycles are required by the C function for each of the 8 inputs?

Each input	Clock Cycles	Differences
Beginning	1703	
1	2837	1134
2	3748	911
3	4457	709
4	4985	528
5	5353	368
6	5610	257
7	5791	181
8	5929	138

Figure 3.4.1

3.5 Question 5

What is the relationship between your answers for the execution times and memory usage of your assembly and C versions of the function implementation?

The assembly code takes less clock cycles and less memory, therefore we can conclude that writing code in assembly is more efficient for the embedded system.

4. Conclusions

We can conclude that creating code in assembly language is more efficient because it takes less clock cycles and therefore less time to execute the code.