CS1022 Exercise Set #2
Hilary Term Weeks 4 and 5
Stacks and Subroutines

1 Stacks

1.1 Stack Operations with LDR/STR

(a) Provide an ARM Assembly Language program that will push the contents of `R4`, `R5` and `R6` on to the system stack. You may not use the `LDM` or `STM` instructions.

(b) Illustrate the state of the stack after each push operation above.

(c) Provide an ARM Assembly Language program that will pop the three topmost words off the top of the stack, restoring their values into the registers from which they were originally stored in part (a) above. You may not use the `LDM` or `STM` instructions.

(d) Illustrate the state of the stack after each push operation above.

1.2 Stack Operations with LDM/STM

(a) Provide an ARM Assembly Language program that will push the contents of `R4`, `R5` and `R6` on to the system stack. You must use the `LDM` and `STM` instructions.

(b) Provide an ARM Assembly Language program that will pop the three topmost words off the top of the stack, restoring their values into the registers from which they were originally stored in part (a) above. You must use the `LDM` or `STM` instructions.

1.3 Understanding LDM and STM

Provide diagrams to illustrate the state of the stack (stack pointer and contents) after executing each of the following instructions and list the contents of any registeres modified by the operation. Begin by assuming some initial values stored in each register.

```
1        STMFD   SP!, {R4, R6, R8-R11}
2        LDMFD   SP, {R10, R11}                  ; Note - no !
3        LDMFD   SP!, {R8, R10, R11, R4, R6}
```

1.4 Value to Decimal String (using a stack)

For this part of the lab exercise we are revisiting the conversion of a binary value to an ASCII string representing the binary value in decimal form. Last term we saw that this could be done easily by repeatedly dividing the binary value by 10_{10}. Each division produces a quotient (whole part) and a remainder. The remainder can be converted to an ASCII digit while the quotient is further divided to produce the next digit.

The only problem with this approach is that it produces the digits in right-to-left order whereas we want to obtain the digits in left-to-right order.

Modify your divide program from last term, making use of a stack to temporarily store the digits as they are produced. You can then pop the digits off the stack and store them as a string in memory!

Verify that your program works by implementing it in uVision.

Here is some pseudo-code to help get you started!

```
buffer = deststr;
val = 365;
push(NULL);          // So we know when to stop!

// Compute characters and push them onto the system stack

remainder = val;
do {
    quotient = 0;
    while (remainder >= 10) {
        remainder = remainder - 10;
        quotient++;
    }
    char = remainder + '0';
    push(char);
    remainder = quotient;
} while (remainder != 0);

// Pop characters back off the system stack

do {
    char = pop();
    Memory.Byte[buffer++] = char;
} while (char != 0);     // Remember we pushed NULL earlier!!
```

2 Subroutines

2.1 Subroutine Basics

Consider the following ARM Assembly Language program. Explain in detail the execution of the program. Why will the program not work? Suggest how you would fix the problem.

```
1  ; Top level program
2  start
3          BL       sub1            ; call sub1
4
5  stop
6          B        stop
7
8  ; sub1 subroutine
9  sub1
10         ADD      R0, R1, R2      ; do something
11         BL       sub2            ; call sub2
12         ADD      R3, R4, R5      ; do something
13         BX       lr              ; return from sub1
14
15 ; sub2 subroutine
16 sub2
17         BX       lr              ; return from sub2
```

2.2 Subroutines and the System Stack

Consider the execution of the following ARM Assembly Language program. Illustrate the state of the system stack after the execution of each instruction that manipulates the stack. Note that the effect of the instructions on the system stack is cumulative.

```
1  start
2          BL       subroutine1
3  stop
4          B        stop
5
6  subroutine1
7          STMFD    sp!, {R1-R2,LR}
8          ADD      R0, R1, R2
9          MOV      R1, R0
10         BL       subroutine2
11         LDMFD    sp!, {R1-R2,PC}
12
13 subroutine2
14         STMFD    sp!, {LR}
15         MUL      R0, R1, R2
16         LDMFD    sp!, {PC}
```

2.3 Subroutine Interfaces

(a) For each of the following Java/C-like method declarations, design an appropriate ARM Assembly Language interface for a corresponding assembly language subroutine. The interface must include a specification of how each parameter is passed into the subroutine and how any return values are passed back to the calling program.

(i) `void zeroMemory(unsigned int startAddress, unsigned int length)`
(zero a range of addresses in memory)

(ii) `int factorial(unsigned int x)`

(iii) `int power(int x, unsigned int y)`

(iv) `int quadratic(int a, int b, int c, int x)`
(evaluate a quadratic function)

(b) For each of the subroutines listed above, show how you would invoke (call) the subroutine, assuming the variables to be passed as parameters are initially stored in registers other than R0–R3.

(c) For each of the subroutines listed above, show the state of the system stack immediately after the subroutine saves any registers on the system stack. (You may assume a set of registers that each subroutine uses in its implementation.)

See the BONUS EXERCISE below!

2.4 Writing Subroutines

Design, write and test an ARM Assembly Language subroutine that will divide two unsigned, non-negative integers and return the quotient (whole part) and remainder. Begin by designing and documenting a suitable interface for your subroutine. Provide a program to test your subroutine by invoking (calling) it.

3 GRADED EXERCISE: Bubble Sort

(a) Write an ARM Assembly Language subroutine, **swap(array, i, j)** that will swap two elements in a 1-dimensional array of word-size integers. Your subroutine should accept the address of the array and the indices of the two elements to be swapped as parameters. Begin by designing an appropriate interface for your subroutine. Write a program to test your subroutine.

(b) Translate the pseudo-code shown below into an ARM Assembly Language subroutine. You must make use of your **swap(array, i, j)** subroutine from Part (a).

```
sort(array, N)
{
    do
    {
        swapped = false;
        for (i = 1; i < N; i++)
        {
            if (array[i-1] > array[i])
            {
                swap(array, i-1, i);
                swapped = true;
            }
        }
    } while (swapped);
}
```

4 BONUS EXERCISE: Subroutine Implementations

Implement each of the subroutines listed in Section 2.3, taking care to hide any unintended side-effects from a calling program using LDM and STM instructions to save and restore registers on the system stack. Your subroutines must adhere to the interfaces that you defined above. Try to use registers R0–R3 to pass parameters and R4–R12 to store variables that are local to the subroutine.