Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics
CS1022 / Introduction to Computing II

Tutorial #2
Hilary Term Weeks 4 and 5
LDM, STM, Stacks and Subroutines

Work in groups of 6 students using one of the whiteboards.
Complete the exercises in your own time if necessary.
Revise the exercises to prepare for the next lab.

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.

2 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.

```
        STMFD   SP!, {R4, R6, R8-R11}
        LDMFD   SP,  {R10, R11}          ; Note - no ! in this instruction
        LDMFD   SP!, {R8, R10, R11, R4, R6}
```

3 Other uses of LDM/STM

Write an ARM Assembly Language program to copy 512 bytes of memory from the address in R0 to the address in R1. Take advantage of the LDM and STM instructions to load and store multiple words using a single instruction.

Calculate the number of memory accesses required by your program and compare it with the number of memory accesses required if using only LDR an STR instructions.

4 Subroutines

4.1 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.

4.2 Subroutine Implementations

Implement each of the subroutines listed in Section 4.1, 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 should 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.

Complete the exercises in your own time if necessary!

CS1022 Tutorial #2 SOLUTION
Hilary Term Weeks 4 and 5
LDM, STM, Stacks and Subroutines

1 Stacks

1.1 Stack Operations with LDR and STR

(a)
```
        STR     R4, [SP, #-4]!
        STR     R5, [SP, #-4]!
        STR     R6, [SP, #-4]!
```

(b) Diagram of memory showing stack state after above operations

(c) (Note the order – reverse of above)

```
        LDR     R6, [SP], #4
        LDR     R5, [SP], #4
        LDR     R4, [SP], #4
```

(d) Diagram of memory showing stack state after above operations

1.2 Stack Operations with LDM and STM

(a)
```
        STMFD SP!, {R4-R6}
```

(b) Note, the LDM instruction takes care of the order of registers for us. The rule is the lowest numbered register is always loaded/stored from/to the lowest address. So, the order of registers in the list doesn't matter in either LDM or STM!

```
        LDMFD SP!, {R4-R6}
```

1.3 Understanding LDM and STM

1 `STMFD SP!, R4, R6, R8-R11`

Diagram of memory showing stack state after above operation

2 `LDMFD SP, R10, R11`

Note – no !

Diagram of memory showing stack state after above operation

3 `LDMFD SP!, R8, R10, R11, R4, R6`

Note – order doesn't matter, we will still reverse the PUSH performed by the first instruction!!

Diagram of memory showing stack state after above operation

1.4 Other uses of LDM/STM

We will copy memory in 64 blocks of 8 words, using 8 registers to load and then store each block, thereby reducing the number of instruction executions required. We will assume that the memory regions are non-overlapping.

```
Nbytes  EQU     512                 ; number of elements

        AREA    globals, DATA, READWRITE

; N word-size values
dest    SPACE   Nbytes              ; copy destination


        AREA    RESET, CODE, READONLY
        ENTRY

        LDR     R1, =src                        ;
        LDR     R2, =dest

        MOV     R0, #0                          ; count = 0
wh1     CMP     R0, #Nbytes >> 3                ; while (count < Nbytes / 8 registers) {
        BHS     eWh1                            ;

; we can increment the src and destination address registers
;   explicitly using ADD instructions ...

;       LDMIA   R1, {R5-R12}                    ;   load 8 words into R5-R12
;       STMIA   R2, {R5-R12}                    ;   store 8 words from R5-R12
;       ADD     R1, R1, #8*4                    ;   advance src by 8 words
;       ADD     R2, R2, #8*4                    ;   advance dst by 8 words

; ... or implicitly using ! to advance (increment) the pointer

        LDMIA   R1!, {R5-R12}                   ;   load 8 words into R5-R12
        STMIA   R2!, {R5-R12}                   ;   store 8 words from R5-R12

        ADD     R0, R0, #1                      ;   count++

        B       wh1                             ; }
eWh1

STOP    B       STOP

src     DCD     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
        DCD     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

        ; ... 512 bytes in total ...

        DCD     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
        DCD     1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

        END
```

2 Subroutines

2.1 Subroutine Interfaces

(a) (i) zeroMemory

```
; zeroMemory - zero a range of addresses in memory
; Parameters:
;   R0: start address of range to zero
;   R1: length number of bytes to zero (unsigned word)
```

(ii) factorial

```
; factorial - compute x!
; Parameters:
;   R0: x (unsigned word)
; Return value:
;   R0: x!
```

(iii) power

```
; power - compute x^y
; Parameters:
;   R0: x (signed word)
;   R1: y (unsigned word)
; Return value:
;   R0: x^y
```

(iv) quadratic

```
; quadratic - evaluate a quadratic function of the form f(x) = ax^2+bx+c
; Parameters:
;   R0: a (signed word)
;   R1: b (signed word)
;   R2: c (signed word)
;   R3: x (signed word)
; return value:
;   R0: f(x) (signed word)
```

(b) (i) zeroMemory (e.g. zero 1024 bytes starting at the address in R4)

```
        ; assume some initial values in registers other that R0-R3
        LDR     R4, =tbz

        ; call zeroMemory using interface
        MOV     R0, R4
        LDR     R1, =1024
        BL      zeroMemory
```

(ii) factorial (e.g. compute 6!)

```
        ; assume some initial values in registers other that R0-R3
        LDR     R4, =6

        ; call factorial using interface
        MOV     R0, R4
```

```
        BL      factorial
        MOV     R7, R0
```

(iii) power (assume R4, R7, R9 have some meaning in a wider context)

```
        ; assume some initial values in registers other that R0-R3
        LDR     R4, =4
        LDR     R7, =3

        ; call power using interface
        MOV     R0, R4
        MOV     R1, R7
        BL      power
        MOV     R9, R0
```

(iv) quadratic (assume R4, R5, R7, R9, R11 have some meaning in a wider context)

```
        ; assume some initial values in registers other that R0-R3
        LDR     R4, =2
        LDR     R7, =3
        LDR     R9, =4
        LDR     R11, =5

        ; call quadratic using interface
        MOV     R0, R4
        MOV     R1, R7
        MOV     R2, R9
        MOV     R3, R11
        BL      quadratic
        MOV     R5, R0
```

2.2 Subroutine Implementations

(i) zeroMemory

```
; zeroMemory - zero a range of addresses in memory
; Parameters:
;   R0: start address of range to zero
;   R1: length number of bytes to zero (unsigned word)
zeroMemory
        STMFD   SP!, {R4-R6, LR}

        ; take a local copy of each parameter
        MOV     R4, R0
        MOV     R5, R1
        MOV     R6, #0

        ; fill memory with zeros
wh1     CMP     R5, #0
        BHS     eWh1
        STRB    R6, [R4], #1
        SUB     R5, R5, #1
        B       wh1
eWh1
        LDMFD   SP!, {R4-R6, PC}

        ; we could have optimised this by filling words untell we were left
        ; with < 4 bytes, at which point we finish off with bytes
```

(ii) factorial

```
; factorial - compute x!
; Parameters:
;   R0: x (unsigned word)
; Return value:
;   R0: x!
factorial
        STMFD   SP!, {R4, LR}

        ; take a local copy of each parameter
        MOV     R4, R0

        ; compute factorial
        MOV     R0, #1
wh2
        CMP     R4, #0
        BEQ     eWh2
        MUL     R0, R4, R0
        SUB     R4, R4, #1
        B       wh2
eWh2
        LDMFD   SP!, {R4, PC}
```

(iii) power

```
; power - compute x^y
; Parameters:
;   R0: x (signed word)
;   R1: y (unsigned word)
; Return value:
;   R0: x^y
power
        STMFD   SP!, {R4-R5, LR}

        ; take a local copy of each parameter
        MOV     R4, R0
        MOV     R5, R1

        ; compute x^y
        MOV     R0, #1
wh3
        CMP     R5, #0
        BEQ     eWh3
        MUL     R0, R4, R0
        SUB     R5, R5, #1
        B       wh3
eWh3
        LDMFD   SP!, {R4-R5, PC}
```

(iv) quadratic

```
; quadratic - evaluate a quadratic function of the form f(x) = ax^2+bx+c
; Parameters:
;   R0: a (signed word)
;   R1: b (signed word)
;   R2: c (signed word)
;   R3: x (signed word)
; return value:
;   R0: f(x) (signed word)
quadratic
        STMFD   SP!, {R4-R7, LR}

        ; take a local copy of each parameter
        MOV     R4, R0
        MOV     R5, R1
```

```
        MOV     R6, R2
        MOV     R7, R3

        ; compute quadratic
        MUL     R0, R3, R3
        MUL     R0, R4, R0

        MUL     R5, R7, R5
        ADD     R0, R0, R5

        ADD     R0, R0, R6

        LDMFD   SP!, {R4-R7, PC}
```