3.1 Stack Manipulation Operations

Fig. 3.1 shows the contents on the system stack and the current position of the stack pointer (**SP**) of an ARM processor. With reference to Fig. 3.1, answer the following questions.

(1) What is the current content (in hexadecimal) of the register **SP**?

(2) Give the instruction to push the content in **R1** onto the system stack.

(3) Give a single instruction to pop off four word-sized items from the stack to registers R6-R9.

(4) Give the instruction to retrieve the 3rd word-sized item (i.e. **0x00000000**) on the stack and place it into **R0**. You are not to pop any items from the stack.

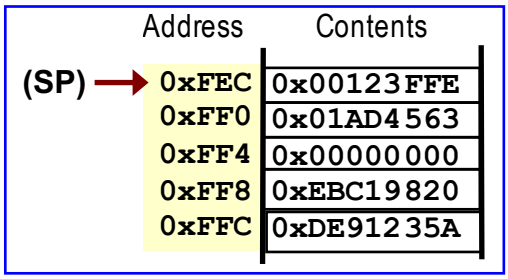

Fig. 3.1 – The system stack and its contents

3.2 Modular Programming – Subroutine Call and Parameter Passing

Fig. 3.2 shows a code segment for subroutine **MySub** and the calling program that makes use of it. With reference to Fig. 3.2, answer the following questions:

(1) How is each of the parameters **NumX**, **NumY** and **Ans** passed into the **MySub** subroutine? Is it by value or by reference? Give reasons for each of your answers.

(2) With the aid of the code address shown in Fig. 3.2, give the hexadecimal content of the **PC** and **SP** immediately after the execution of each of the instructions at (b1), (c1) and (s6). Assume execution begins at instruction (a1).

(3) What is the hexadecimal content in **R0** after instruction (d1) is executed?

(4) Give a single ARM instruction at (e1) that would remove all the parameters pushed to the stack?

(5) Replace the **BL MySub** instruction with a branch and any other supporting instruction that supports branching to subroutine.

(6) With the help of the comments given, complete the subroutine **MySub**. This subroutine implements the function **Ans = NumX * Num Y**. The unsigned word-sized values of **NumX** and **NumY** are multiplied and the resulting word-sized multiplication is stored directly to memory variable **Ans**. For example, given the values of 0x004 and 0x003 for **NumX** and **NumY** respectively, the subroutine **MySub** should compute the result 0x00C and place this result into the memory variable **Ans** as shown in Fig. 3.2.

```
Code
Address
0x000  Start  MOV   SP,#0xFFFFFFFC ; (a1) Initialize stack pointer
0x004         MOV   R0,#0x100
0x008         LDR   R1,[R0]
0x00C         STR   R1,[SP,#-4]!   ; (b1) NumX to stack
0x010         ADD   R0,R0,#4
0x014         LDR   R1,[R0]
0x018         STR   R1,[SP,#-4]!   ; (b2) NumY to stack
0x01C         ADD   R0,R0,#4
0x020         STR   R0,[SP,#-4]!   ; (b3) Ans to stack
0x024         BL    MySub          ; (c1)
              LDR   R0,[SP,#-4]    ; (d1)
               ?     ?             ; (e1) Remove stack parameters
                 :
                 :

0x040  MySub   ?     ?             ; (s1) Save registers R0,R1,R2,R3
               ?     ?             ; (s2) Retrieve NumX from stack
               ?     ?             ; (s3) Retrieve NumY from stack
                  :                ; Complete the segment of code to compute the
                  :                ; value of NumX*NumY using successive addition
               ?     ?             ; (s4a) Move the result directly to …
               ?     ?             ; (s4b) … the memory variable Ans
               ?     ?             ; (s5) Restore saved registers
              MOV PC,LR            ; (s6) Return to calling program
```

Address	Contents
NumX 0x100	0x004
NumY 0x104	0x003
Ans 0x108	0x00C
0x10C	0x000
0x110	0x000

Fig. 3.2 – A partially completed ARM assembly language program

---

(Question 3.3 and 3.4 need not be covered during the tutorial)

3.3 Generating Time Delays using Software

Fig. 3.3 shows a subroutine for generating a short time delay.

(1) Calculate the delay produced by the subroutine in terms of number of execution cycles.

(2) Given that the ARM processor system clock is 10MHz, what is the duration of the time delay produced by the routine? You may assume that each memory access cycle uses one cycle of the system clock.

(3) How can a time delay of 1ms (millisecond) be achieved?

```
Delay   MOV    R0,#10
Loop    SUBS   R0,R0,#1
        BNE    Loop
        MOV    PC,LR
```

Fig 3.3 – Delay subroutine

3.4 Multi-precision Arithmetic

An ARM assembly language program and the contents of several memory variables are given in Fig. 3.4. Each multi-precision integer **M1**, **M2** and **M3** is stored using two words in memory and in Little Endian format. The memory variable **N1** stores the number of two-word integers in the array. On completion of the multi-precision addition routine, the last two-word integer in the array is replaced by the total sum of all the two-word integers in the array.

(1) With reference to Fig. 3.4, for each "?", give the single ARM mnemonic that will implement the corresponding functionality described by each of the comments shown. (note: Some '?' require more than 1 instruction)

```
Start   ?            ; push the value in N1                          (m1)
        ?            ;  and M1                                       (m2)
        ?            ;                                               (m3)
        ?            ; on to the stack                               (m4)
        ?            ; make a call to subroutine SubA                (m5)
        :
        :
SubA    ?            ; save value registers R4-R7 on to the stack    (s1)
        ?            ; create a 3-word stack frame on the stack      (s2)
        ?            ; get the contents of N1 on the stack into R4   (s3)
      STR R4,[SP]    ; use 1st word (SF1) in stack frame to save value of N (s4)
        ?            ; get the address of M1 on the stack into R4    (s5)
        ?            ; clear 2nd word (SF2) in stack frame           (s6)
        ?            ; clear 3rd word (SF3) in stack frame           (s7)
Loop    ?            ; add lower word of integer and save result in (SF2) (s8)
        ?            ; add upper word of integer and save result in (SF3) (s9)
        ?            ; decrement value of N1 in (SF1)
      BEQ  Done      ; jump if N1=0                                  (s11)
      ADD  R4,R4,#8  ; point to next integer                         (s12)
      B  Loop        ; jump back to add again                        (s13)
Done  LDR R5,[SP,#4]   ; copy result to                              (s14)
      STR R5,[R4]                                                    (s15)
      LDR R5,[SP,#8]                                                 (s16)
      STR R5,[R4,#4]   ; last array item                             (s17)
        ?            ; collapse stack frame                          (s18)
        ?            ; restore original registers                    (s19)
      BX LR
```

	Address	Contents
N	0x100	0x003
M1	0x104	0xFFFFFFFF
	0x108	0x00000111
M2	0x10C	0x00000002
	0x110	0x00000600
M3	0x114	0x00000222
	0x118	0x00000100

Memory Map

Fig 3.4 – An incomplete ARM calling program and subroutine

(2) Describe what changes to the program in Fig. 3.4 you would make if the multi-precision integers are stored using the Big Endian (word wise) format instead.

(3) Give the two 32-bit hexadecimal values in memory addresses `0x114` and `0x118` at the end of the execution of the code segment shown in Fig 3.4.