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 a VIP 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.

(4) Give the instruction to retrieve the 3rd word-sized item (i.e. **`0x000`**) on the stack and place it into **R0**. You are not to pop any items from the stack.

	Address	Contents
(SP) →	0xFFB	0xFFE
	0xFFC	0xAD3
	0xFFD	0x000
	0xFFE	0x820
	0xFFF	0x123

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:

```
Code
Address
0x000  Start  MOV   SP,#0xFFF      ; (a1) Initialize stack pointer
0x002         PSH   [0x100]        ; (b1) NumX to stack
0x004         PSH   [0x101]        ; (b2) NumY to stack
0x006         PSH   #0x102         ; (b3) Ans to stack
0x008         CALL  MySub          ; (c1)
0x00A         MOV   R0,[SP+0xFFF]  ; (d1)
0x00C          ?     ?             ; (e1) Remove stack parameters
                 :
                 :
0x020  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
              RET                  ; (s6) Return to calling program
```

	Address	Contents
NumX	0x100	0x004
NumY	0x101	0x003
Ans	0x102	0x00C
	0x103	0x000
	0x104	0x000

Fig. 3.2 – A partially completed VIP assembly language program

(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 VIP instruction at (e1) that would remove all the parameters pushed to the stack?

(5) The instruction **`CALL MySub`**, in line (c1) uses absolute jump. Suggest how you could replace the instruction in (c1) with another implementation that uses a relative jump.

Note: You are free to use as many VIP instructions as you wish to replace **`CALL MySub`**.

(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.

(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 VIP 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    SUB     R0,#1
        JNE     Loop
        RET
```

Fig 3.3 – Delay subroutine

3.4 Multi-precision Arithmetic

A VIP 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 **N** 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 VIP mnemonic that will implement the corresponding functionality described by each of the comments shown.

```
Start   ?                    ; push the value in N on to the stack                   (m1)
        ?                    ; push the address of M1 on to the stack                (m2)
        ?                    ; make a call to subroutine SubA                        (m3)
        :
        :
SubA    ?                    ; save value in register R2 on to the stack             (s1)
        ?                    ; create a 3-word stack frame on the stack              (s2)
        ?                    ; get the contents of N on the stack into R2            (s3)
     MOV [SP],R2             ; use 1st word (SF1) in stack frame to save value of N  (s4)
        ?                    ; get the address of M1 on the stack into R2            (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 N in (SF1)                         (s10)
     JEQ  Done               ; jump if N=0                 (s11)
     ADD  R2,#2              ; point to next integer       (s12)
     JMP  Loop               ; jump back to add again      (s13)
Done MOV  [R2+0],[SP+1]      ; copy result to              (s14)
     MOV  [R2+1],[SP+2]      ; last array item             (s15)
        ?                    ; collapse stack frame        (s16)
        ?                    ; restore original value of R2 (s17)
     RET
```

Memory Map

	Address	Contents
N	0x100	0x003
M1	0x101	0xFFF
	0x102	0x111
M2	0x103	0x002
	0x104	0x600
M3	0x105	0x222
	0x106	0x100

Fig 3.4 – An incomplete VIP 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 format instead. Change only two instructions.

(3) Give the two 12-bit hexadecimal values in memory addresses 0x105 and 0x106 at the end of the execution of the VIP code segment shown in Fig 3.4.