[12 marks] 2. Consider the ARM assembly-language program that is shown below.

```
          .global  _start
_start:
          MOV      R0, #0
          MOV      R2, #N
          LDR      R3, [R2]
LOOP:     ADDS     R3, R3
          BCC      CONT
          ADD      R0, #1
CONT:     MOVS     R3, R3
          BNE      LOOP

          STR      R0, [R2, #4]
END:      B        END

N:        .word    0xAF13          // the data
Result:   .word    0               // space for the result
```

(a) Using as few words as possible explain what this code "does". That is, for a given value of N, what corresponding value will the program generate and store in *Result*? Note that the condition `CC` used in the `BCC` instruction means *Carry Clear*. Hence, the `BCC` branch will be taken when the *c* flag is 0.

Answer:

Part (b) of this question is on the next page . . .

[10 marks] 3. For this question you are to write an assembly-language program for the ARM processor. The program should perform the following task: in an infinite loop read from the SW port (address 0xFF200040). If the value read from SW is 0, 1, 2, or 3 then you should display the characters 'g', 'o', 'o', and 'd', respectively, on the HEX0 display (address 0xFF200020). You should *error-check* the value read from the SW port; if it is greater than 3 then you should display a '-' on HEX0.

For the 7-segment display recall that segment 0 is at the top and then the segments are ordered from 1 to 5 in the clockwise direction, with segment 6 in the middle.

Show your ARM assembly-language code in the space below:

[8 marks] 4. Consider the ARM code shown below. Note that the address of each instruction in the memory is shown to the left of the code. The directive `.asciz "ha ho ha ho"` places the given ASCII characters (bytes) in memory, including a 0 byte to designate the end of the string.

```
                        .text
                        .global _start

00000000 _start:     MOV     R0, #S1
00000004             MOV     R1, #S2
00000008             BL      Goober
0000000C Stop:       B       Stop

00000010 Goober:     MOV     R2, R0
00000014 Freakshow:  LDRB    R3, [R2]
00000018             CMP     R3, #0
0000001C             BEQ     DoDat
00000020             ADD     R2, #1
00000024             B       Freakshow

00000028 DoDat:      CMP     R2, R0
0000002C             BEQ     NoMoe

00000030             SUB     R2, #1
00000034             LDRB    R3, [R2]
00000038             STRB    R3, [R1]
0000003C             ADD     R1, #1
00000040             B       DoDat
00000044 NoMoe:      MOV     R3, #0
00000048             STRB    R3, [R1]
0000004C             MOV     PC, LR

00000050 S1:        .asciz "ha ho ha ho"
0000005C S2:        .asciz " "

                    .end
```

(a) If this program is executed on the ARM processor, what would be the values that would be shown in a debugger the **first** time the code reaches the instruction at address 0x28.

R0		R1		R2	
R3		R14		R15	

(b) What does this code "do"? That is, given the string "ha ho ha ho" what does the program produce?

Answer

[7 marks] 5. An ARM program is shown below, which is supposed to work as follows. There is a main program that reads two integers, A and B, from the memory, multiplies them to produce $C = A \times B$, and then stores the result in memory. Instead of using the ARM `MUL` instruction, this code does the multiplication by using a subroutine, called `MULTIPLY`, which performs repeated addition (similar to the code that you wrote in your lab exercises that used repeated subtraction to do division).

Unfortunately, the `MULTIPLY` subroutine has been written by an aging professor, who has "lost it," and the subroutine code contains some errors. Your job is to find the errors, and fix them.

```
1             .text
2             .global _start
3  _start:    MOV     R0, #A
4             LDR     R0, [R0]
5             MOV     R1, #B
6             LDR     R1, [R1]
7             BL      MULTIPLY
8             MOV     R2, #C
9             STR     R0, [R2]
10 END:       MOV     R15, #END
11
12 MULTIPLY:  MOV     R3, R1
13            BEQ     EMULT
14            MOV     R3, R0
15 CONT:      CMP     R1, #0
16            BNE     EMULT
17            ADD     R3, R3
18            SUB     R1, #1
19            B       CONT
20 EMULT:
21            MOV     PC, #END
22
23 A:         .word   10
24 B:         .word   10
25 C:         .word   0
```

Answer the questions on the next page.

(a) How many errors did you find in the subroutine code?

Answer . . . . . .

(b) Briefly describe each of the errors that you found, in the space below. Note that the lines of code are numbered for convenience of reference.

(c) In the space below provide a corrected version of the `MULTIPLY` subroutine.

```
MULTIPLY:
```