In this assignment you are to write the body of a generic `Print_Number` subroutine to print a signed integer passed in register R0 on the console as a decimal value. The code framework appears on the next page, and is available on the class web site in the LAB4.ZIP archive (or you can type it in yourself if you are feeling particularly masochistic). Except for the lines containing your name in the program listing and the bodies of the `Print_Number` and the `PrintLF` subroutines, you are not allowed to make any changes to the existing code.

No locations in main memory are allowed; all program actions happen in the registers. Registers are saved to and restored from the stack only. The `Print_Number` and `PrintLF` subroutines must be completely transparent with respect to the registers they use, and transparency is achieved through use of the stack only.

Your first task is to rewrite the `PrintLF` subroutine used in previous assignments so that transparency is achieved by saving registers to and restoring them from the stack instead of to and from main memory:

Pushing register x onto the stack is done by: \[ \text{STR } R_x, [SP, #-4] \]
Pushing register x from the stack is done by: \[ \text{LDR } R_x, [SP], #4 \]

The main program calls `Print_Number` with arguments that step by 8s from -120 up through but not including 120 (the last number is 112). Each number is printed on its own line in the console window. Registers R1 through R11 are “preloaded” with special values so we can see if your routines preserve register values correctly.

`Print_Number` must handle both positive and (two’s-complement) negative numbers properly, and must be able to handle an arbitrary number of digits, with complete suppression of unnecessary leading zeroes. Negative numbers must be printed with a leading minus sign, while positive numbers and zero have no leading character (i.e., a plus sign is not required).

To support this subroutine I provide an existing subroutine called `UDiv10`, which divides the unsigned integer value in R0 by 10 and puts the quotient in R0 and the remainder in R1. As with many “provided” routines in real life, this routine is not completely transparent; it trashes the contents of R2 (and you aren’t allowed to fix it). Thus, your `Print_Number` routine must take pains to accommodate the deficiencies of `UDiv10`.

When working, print out the .ALI assembly listing and a screenshot containing the source code after execution with your name visible, the user registers, and the console window showing the resulting printed numbers. Staple the screenshot on top of the listing. Here are the point penalties for this 10-point assignment (no assignment will score less than zero):

1. -1 for cosmetic errors: printouts not stapled, or screenshot not on top.
2. -10 for name not visible on .ALI listing (did you write the code?)
3. -5 for not printing values -120 to 112 by 8s (does it work at all?)
4. -2 for non-transparent subroutines (are registers saved and restored?)
5. -2 for not showing the user registers (can we can check for transparency?)
6. -2 for modifying any existing code (did you change the assignment?)
7. -2 for using explicit memory locations (did you use the stack?)
8. -2 for incorrect digit order or not suppressing leading zeroes (number formatting).
; {Insert your name and ID here}

AREA  PROGRAM4, CODE
ENTRY

Start  MOV  R1,#129
      MOV  R2,#130
      MOV  R3,#131
      MOV  R4,#132
      MOV  R5,#133
      MOV  R6,#134
      MOV  R7,#135
      MOV  R8,#136
      MOV  R9,#137
      MOV  R10,#138
      MOV  R11,#139
      MOV  R0,#-120

Main_Loop  BL  Print_Number
          BL  PrintLF
          ADD  R0,R0,#8
          CMP  R0,#120
          BMI  Main_Loop
          SWI  &11

; {Insert your name and ID here}

Print_Number
;  {Replace these lines with your code,
;   including any data declarations}

PrintLF
;  {Nearly the same code as before}

UDiv10  SUB  R1,R0,#10
       SUB  R0,R0,R0,LSR #2
       ADD  R0,R0,R0,LSR #4
       ADD  R0,R0,R0,LSR #8
       ADD  R0,R0,R0,LSR #16
       MOV  R0,R0,LSR #3
       ADD  R2,R0,R0,LSL #2
       SUBS  R1,R1,R2,LSL #1
       ADDPL R0,R0,#1
       ADDMI R1,R1,#10
       MOV  PC,LR

;-------------------------------------------------------------------------------
END