```
;CIS 11 TEST SCORE CALCULATOR
;MANUEL CAMORLINGA
;MAURO ELIAS

.ORIG x3000

LEA     R0, WEL
PUTS                        ; Print welcome message
WEL    .STRINGZ "Enter 5 scores: (0 - 99)"
LD R0, NEWLINE
OUT                         ; Print newline
JSR GET_GRADE      ; Call GET_GRADE function to get input
LEA R6, GRADES
STR R3, R6, #0      ; Store grade 1 into GRADES array
JSR GET_LETTER     ; Get letter grade
JSR POP                     ; Pop from stack
LD R0, NEWLINE
OUT                         ; Print newline
JSR GET_GRADE
LEA R6, GRADES
STR R3, R6, #1
JSR GET_LETTER
JSR POP
LD R0, NEWLINE
OUT
JSR GET_GRADE
LEA R6, GRADES
STR R3, R6, #2
JSR GET_LETTER
JSR POP
LD R0, NEWLINE
OUT
JSR GET_GRADE
LEA R6, GRADES
STR R3, R6, #3
JSR GET_LETTER
JSR POP
LD R0, NEWLINE
OUT
JSR GET_GRADE
LEA R6, GRADES
STR R3, R6, #4
JSR GET_LETTER
JSR POP
```

```
LD R0, NEWLINE
OUT

; Calculate and display the maximum grade
CALCULATE_MAX
        LD R1, NUM_TESTS  ; R1 = NUM OF TESTS
        LEA R2, GRADES            ; R2 = GRADES ADDRESS
        LD R4, GRADES             ; G(0)
        ST R4, MAX_GRADE
        ADD R2, R2, #1
LOOP1       LDR R5, R2, #0              ; Pointer to GRADES
        NOT R4, R4
        ADD R4, R4, #1
        ADD R5, R5, R4
        BRp NEXT1
        LEA R0, MAX
        PUTS                      ; Print "MAX"
        LD R3, MAX_GRADE
        AND R1, R1, #0
        JSR BREAK_INT
        LD R0, SPACE
        OUT                       ; Print space
LD R0, NEWLINE
OUT                       ; Print newline
JSR CLEAR_REG             ; Clear registers

; Calculate and display the minimum grade
CALCULATE_MIN
        LD R1, NUM_TESTS  ; R1 = NUM OF TESTS
        LEA R2, GRADES            ; R2 = GRADES ADDRESS
        LD R4, GRADES             ; G(0)
        ST R4, MIN_GRADE
        ADD R2, R2, #1
        ADD R1, R1, #-1
LOOP2       LDR R5, R2, #0              ; Pointer to GRADES
        NOT R4, R4
        ADD R4, R4, #1
        ADD R5, R5, R4
        BRn NEXT2
        ADD R2, R2, #1
        LD R4, GRADES
        AND R5, R5,#0
        ADD R1,R1,#-1
        BRp LOOP2
        LEA R0, MIN
        PUTS                    ; Print "MIN"
        LD R3, MIN_GRADE
        AND R1, R1, #0
        JSR BREAK_INT
        LD R0, SPACE
        OUT                     ; Print space
LD R0, NEWLINE
OUT                     ; Print newline
JSR CLEAR_REG           ; Clear registers

; Calculate and display the average grade
CALC_AVG
        LD R1, NUM_TESTS ; R1 = NUM OF TESTS
        LEA R2, GRADES          ; R2 = GRADES ADDRESS
GEN_SUM LDR R4, R2, #0  ; Load grade
        ADD R3, R3, R4          ; R3 = SUM
        ADD R2, R2, #1          ; Move to next grade
        ADD R1, R1, #-1         ; Decrement counter
        BRp GEN_SUM                     ; Repeat until all grades processed
        LD R1, NUM_TESTS ; R1 = NUM OF TESTS
        NOT R1, R1              ; R1 = -5
        ADD R1, R1, #1          ; R1 = -5
        ADD R4, R3, #0          ; R4 = SUM
LOOP3           ADD R4, R4, #0          ; R4 = SUM
        BRnz DONE_AVG           ; If SUM is zero or positive, calculation done
        ADD R6, R6, #1          ; Increment counter
        ADD R4, R4, R1          ; Subtract 5 from total
        BRp LOOP3               ; Repeat until SUM < 0
DONE_AVE
        ST R6, AVERAGE_SCORE    ; Store average score
        LEA R0, AVG             ; Print "AVG"
        PUTS
        AND R3, R3, #0
        AND R1, R1, #0
        AND R4, R4, #0
        ADD R3, R3, R6          ; R3 = Average score
        JSR BREAK_INT           ; Print average score
JSR RESTART_PROG                ; Restart program
HALT

NEWLINE                 .FILL xA
SPACE                   .FILL X20
DECODE_DEC              .FILL #-48
```

```
DECODE_SYM              .FILL #48
DECODE_THIRTY           .FILL #-30
NUM_TESTS       .FILL 5
RESTART2        .FILL x3000
MAX_GRADE       .BLKW 1
MIN_GRADE       .BLKW 1
DONE_AVG        .BLKW 1
AVERAGE_SCORE           .BLKW 1

; Loop to store minimum grade
NEXT2
        LDR R4, R2, #0              ; Load grade
        ST R4, MIN_GRADE    ; Store minimum grade
        ADD R2, R2, #1              ; Move to next grade in GRADES array
        ADD R1, R1, #-1             ; Decrement counter
        BRnzp LOOP2                         ; Repeat until all grades processed

; Loop to store maximum grade
NEXT1
        LDR R4, R2, #0              ; Load grade
        ST R4, MAX_GRADE ; Store maximum grade
        ADD R2, R2, #1              ; Move to next grade in GRADES array
        ADD R1, R1, #-1             ; Decrement counter
        BRp LOOP1                   ; Repeat until all grades processed

GRADES      .BLKW 5                 ; Array to store grades
MIN     .STRINGZ "MIN "     ; String constant for "MIN"
MAX     .STRINGZ "MAX "     ;
AVG     .STRINGZ "AVG "     ; String constant for "AVG"
RESTART_PROG
        ST R7, SAVELOC1                     ; SAVE JSR LOCATION
        LD R1, LOWER_Y                      ; LOAD NEG VALUE OF Y
        LD R3, UPPER_Y
        LD R2, ORIGIN                       ; LOAD ORIGIN ( x3000)
        LD R0, NEWLINE
        OUT
        LEA R0 RESTARTPROG_STR                      ; RESTART PROMPT STRING
        PUTS
        LD R0, NEWLINE
        OUT
        GETC
        ADD R1, R1, R0                      ; COMPARE USER INPUT WITH -y
        BRz RESTART_TRUE                    ; IF TRUE BRANCH TO RESTART
        ADD R3, R3, R0                      ; COMPARE USER INPUT WITH -Y
        BRz RESTART_TRUE                ; IF TRUE BRANCH TO RESTART
HALT                                ; ELSE HALT PROGRAM
RESTART_TRUE
        JMP R2
RESTARTPROG_STR         .STRINGZ "PROGRAM FINISHED, DO YOU WANT TO RUN
THIS PROGRAM AGAIN? Y/N "
LOWER_Y                 .FILL xFF87     ; -121
UPPER_Y                 .FILL xFFA7     ; -89
ORIGIN                  .FILL x3000
SAVELOC1 .FILL X0
SAVELOC2 .FILL X0
SAVELOC3 .FILL X0
SAVELOC4 .FILL X0
SAVELOC5 .FILL X0
GET_GRADE ST R7, SAVELOC1               ; STORE JSR LOCATION
          JSR CLEAR_REG                 ; CLEAR REGISTERS
          LD R4, DECODE_DEC             ; LOAD TRANSLATION
          GETC                  ; GET FIRST CHAR
          JSR VALIDA
          OUT                   ; ECHO INPUT
          ADD R1, R0, #0                ; COPY INPUT TO R1
          ADD R1, R1, R4                ; TRANSLATE TO DECIMAL
          ADD R2, R2, #10               ; CLEAR R2
MULT10    ADD R3, R3, R1                        ; ADD INPUT TO R3 (MULT PROCESS)
          ADD R2, R2, #-1               ; DECREMENT COUNTER
          BRp MULT10            ; LOOP UNTIL COUNTER IS ZERO
          GETC                  ; GET SECOND CHAR
          JSR VALIDA
          OUT                   ; ECHO INPUT TO SCREEN
          ADD R0, R0, R4                ; TRANSLATE SECOND INPUT TO DECIMAL
          ADD R3, R3, R0                ; ADD FIRST INPUT(X10) TO SECOND INPUT
          LD R0, SPACE                  ; ADD  SPACE
          OUT                   ; PRINT SPACE
          LD R7, SAVELOC1               ; LOAD JSR RETURN LOCATION
RET                                 ; RETURN

BREAK_INT
        ST R7, SAVELOC1             ; STORE JSR RETURN LOCATION
        LD R5, DECODE_SYM           ; TRANSLATION TO CONVERT DECIMAL TO SYMBOL
        ADD R4, R3, #0              ; COPY INPUT TO R4 (PLATFORM)
DIV1    ADD R1, R1, #1              ; COUNTER FOR DIVISION (QUOTIENT)
        ADD R4, R4, #-10        ; SUBTRACT 10 FROM INPUT
        BRp DIV1                ; SUBTRACT 10 TILL INPUT IS 0 OR NEG
        ADD R1, R1 #-1              ; REMOVE EXTRA 1
        ADD R4, R4, #10                 ; ADD 10 TO GET REMAINDER
        ADD R6, R4, #-10
        BRnp POS
NEG     ADD R1, R1, #1
        ADD R4, R4, #-10
POS     ST R1, Q                        ; STORE QUOTIENT
        ST R4, R                        ; STORE REMAINDER (MOD 10)
        LD R0, Q                        ; LOAD QUOTIENT FOR PRINT
        ADD R0, R0, R5                  ; TRANSLATE DECIMAL TO SYMBOL
        OUT                             ; PRINT QUOTIENT
        LD R0, R                        ; LOAD REMAINDER FOR PRINT
        ADD R0, R0, R5                  ; TRANSLATE DECIMAL TO SYMBOL
        OUT                             ; PRINT REMAINDER (MOD 10)
        LD R7, SAVELOC1                 ; RESTORE JSR RETURN LOCATION
        RET
R .FILL X0
Q .FILL X0
PUSH    ST R7, SAVELOC2                 ; STORE JSR LOCATION
        JSR CLEAR_REG                   ; CLEAR REGISTERS
        LD R6, POINTER                  ; INITIALIZE POINTER
        ADD R6, R6, #0
        BRnz STACK_ERROR

        ADD R6, R6, #-1                 ; DECREMENT POINTER
        STR R0, R6, #0                  ; STORE NUMBER IN R0 TO STACK
        ST R6, POINTER                  ; SAVE POINTER LOCATION
        LD R7, SAVELOC2                 ; RESTORE LOCATION
RET
POINTER     .FILL X4000                 ; POINTER START LOCATION
POP     LD R6, POINTER                  ; LOAD POINTER LOCATION
        ST R1, SAVELOC5
        LD R1, BASELINE
        ADD R1, R1, R6
        BRzp STACK_ERROR
        LD R1, SAVELOC5
        LDR R0, R6, #0                  ; LOAD VALUE IN STACK INTO R0
        ST R7, SAVELOC4                 ; STORE JSR LOCATION
        OUT                     ; PRINT NUMBER FROM STACK
        LD R0, SPACE                    ; LOAD A SPACE
        OUT                     ; PRINT SPACE
        ADD R6, R6, #1                  ; INCREMENT POINTER
        ST R6, POINTER                  ; STORE POINTER LOCATION
        LD R7, SAVELOC4
RET
```

```
STACK_ERROR        LEA R0, ERROR
                   PUTS
                   HALT
BASELINE     .FILL xC000
ERROR                  .STRINGZ "STACK UNDERFLOW OR UNDERFLOW. HALTING
PROGRAM"
GET_LETTER
          AND R2, R2, #0                          ; CLEAR R2
A_GRADE      LD R0, A_NUM                                  ; LOAD NUMBER VALUE
             LD R1, A_LET            ; LOAD SYMBOL VALUE
             ADD R2, R3, R0                   ; COMPARE INPUT TO VALUE OF GRADE
             BRzp STR_GRADE               ; IF POS OR ZERO STORE GRADE
B_GRADE      AND R2, R2, #0
             LD R0, B_NUM
             LD R1, B_LET
             ADD R2, R3, R0
             BRzp STR_GRADE
C_GRADE      AND R2, R2, #0
             LD R0, C_NUM
             LD R1, C_LET
          ADD R2, R3, R0
             BRzp STR_GRADE
D_GRADE      AND R2, R2, #0
             LD R0, D_NUM
             LD R1, D_LET
             ADD R2, R3, R0
             BRzp STR_GRADE
F_GRADE      AND R2, R2, #0
             LD R0, F_NUM
             LD R1, F_LET
             ADD R2, R3, R0
             BRNZP STR_GRADE
RET
STR_GRADE  ST R7, SAVELOC1               ; SAVE JSR LOCATION
             AND R0, R0, #0                   ; CLEAR R0
             ADD R0, R1, #0                   ; ADD LETTER TO R0
             JSR PUSH                         ; PUSH LETTER TO STACK
             LD R7, SAVELOC1               ; RESTORE JSR LOCATION
RET                                           ; RETURN TO MAIN
A_NUM        .FILL #-90                 ; Numeric value for grade A
A_LET .FILL X41                 ; ASCII value for grade A
B_NUM        .FILL #-80                 ; Numeric value for grade B
B_LET .FILL X42                 ; ASCII value for grade B
C_NUM        .FILL #-70                 ; Numeric value for grade C
```

```
C_LET .FILL X43                 ; ASCII value for grade C
D_NUM       .FILL #-60                  ; Numeric value for grade D
D_LET .FILL X44                 ; ASCII value for grade D
F_NUM       .FILL #-50                  ; Numeric value for grade F
F_LET .FILL X46                 ; ASCII value for grade F
CLEAR_REG   AND R1, R1, #0                      ; Clear all registers
            AND R2, R2, #0
            AND R3, R3, #0
            AND R4, R4, #0
            AND R5, R5, #0
            AND R6, R6, #0
RET                                     ; Return from subroutine
VALIDA      ST R1, SAVELOC5                     ; Store variables
      ST R2, SAVELOC4
      ST R3, SAVELOC3
      LD R1, DATA_MIN                   ; Compare input to lowest acceptable decimal value
      ADD R2, R0, R1
      BRN FAIL                  ; Fail if out of range
      LD R1, DATA_MAX                   ; Compare input to highest acceptable decimal value
      ADD R3, R0, R1
      BRP FAIL                  ; Fail if out of range
      LD R1, SAVELOC5                   ; Restore variables
      LD R2, SAVELOC4
      LD R3, SAVELOC3
      RET
FAIL  LEA R0, FAIL_STR          ; Fail branch
      PUTS
      LD R0, NEWLINE2
      OUT
      LD R7, RESTART                    ; Load x3000 location
      JMP R7                            ; Restart program
FAIL_STR    .STRINGZ "INVALID ENTRY, RESTARTING..."
RESTART           .FILL X3000
DATA_MIN    .FILL #-48                  ; Minimum acceptable ASCII value
DATA_MAX    .FILL #-57                  ; Maximum acceptable ASCII value
NEWLINE2    .FILL XA                    ; ASCII code for newline character
.END
```