7. The Calculator
Chapter 10

December 5, 2018

- Calculator
  - High-Level View
  - Subroutine details
  - Example code
- Stack arithmetic

Calculator

- Commands
  - X: Exit the simulation
  - C: Clear (all values from the stack)
  - D: Display the value at the top of the stack

Note: This is a stack-based calculator

- Operations

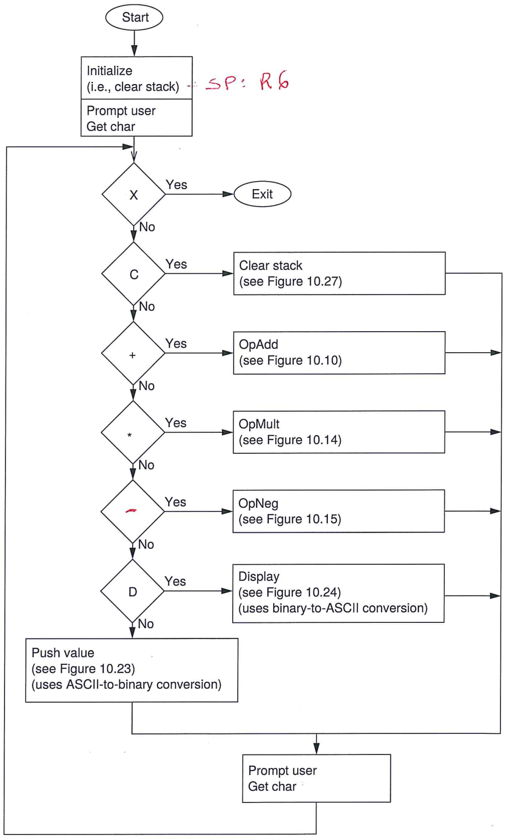
  +: Replace top two elements on the stack with their sum

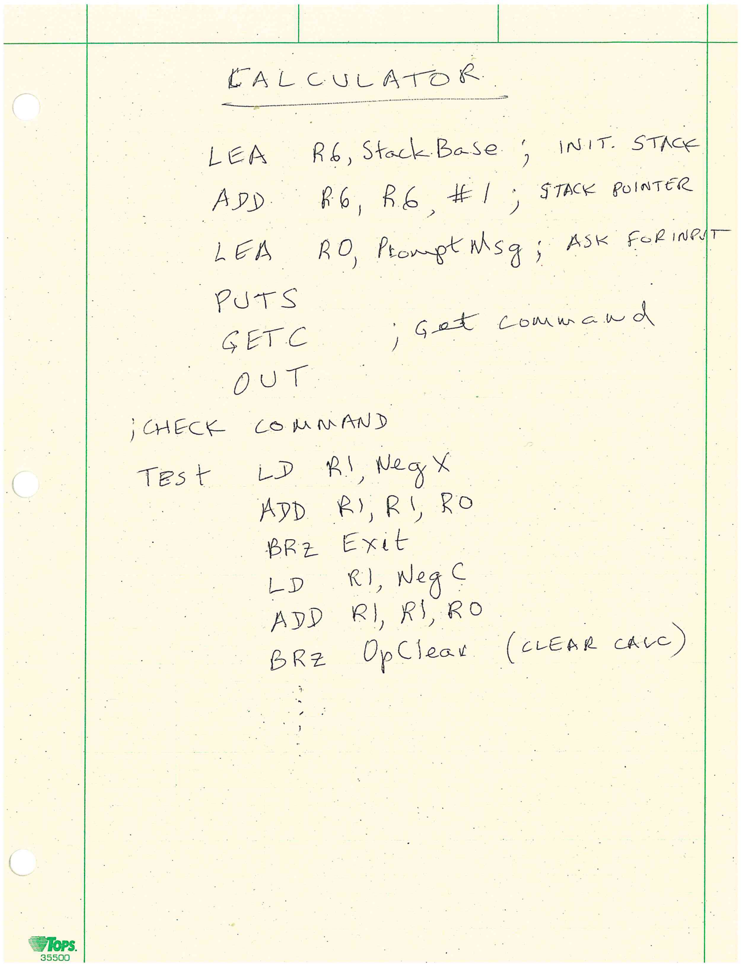
  *: Replace top two elements on stack with their product

  -: Negate the top element on the stack

  Enter: Push value typed on keyboard onto top of the stack

Overview of Calculator





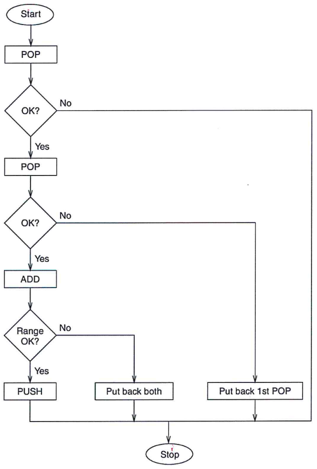
CALCULATOR

```
LEA   R6, StackBase ; INIT. STACK
ADD   R6, R6, #1 ; STACK POINTER
LEA   R0, PromptMsg ; ASK FOR INPUT
PUTS
GETC        ; Get command
OUT
;CHECK COMMAND
Test   LD R1, NegX
       ADD R1, R1, R0
       BRz Exit
       LD  R1, NegC
       ADD R1, R1, R0
       BRz  OpClear  (CLEAR CALC)
         :
         :
```

```
;
;  The Calculator, Main Algorithm
;
                     LEA       R6,StackBase  ; Initialize the Stack.
                     ADD       R6,R6,#1     ; R6 is stack pointer
                     LEA       R0,PromptMsg
                     PUTS
                     GETC
                     OUT
;
; Check the command
;
Test                 LD        R1,NegX       ; Check for X
                     ADD       R1,R1,R0
                     BRz       Exit
;
                     LD        R1,NegC       ; Check for C
                     ADD       R1,R1,R0
                     BRz       OpClear       ; See Figure 10.27
;
                     LD        R1,NegPlus    ; Check for +
                     ADD       R1,R1,R0
                     BRz       OpAdd         ; See Figure 10.10
;
                     LD        R1,NegMult    ; Check for *
                     ADD       R1,R1,R0
                     BRz       OpMult        ; See Figure 10.14
;
                     LD        R1,NegMinus   ; Check for -
                     ADD       R1,R1,R0
                     BRz       OpNeg         ; See Figure 10.15
;
                     LD        R1,NegD       ; Check for D
                     ADD       R1,R1,R0
                     BRz       OpDisplay     ; See Figure 10.26
;
; Then we must be entering an integer
;
                     BRnzp     PushValue     ; See Figure 10.23
;
NewCommand           LEA       R0,PromptMsg
                     PUTS
                     GETC
                     OUT
                     BRnzp     Test
Exit                 HALT
PromptMsg            .FILL     x000A
                     .STRINGZ "Enter a command:"
NegX                 .FILL     xFFA8
NegC                 .FILL     xFFBD
NegPlus              .FILL     xFFD5
NegMinus             .FILL     xFFD3
NegMult              .FILL     xFFD6
NegD                 .FILL     xFFBC
```

```
;
; Subroutines for carrying out the PUSH and POP functions.  This
; program works with a stack consisting of memory locations x3FFF
; (BASE) through x3FFB (MAX).  R6 is the stack pointer.
;
POP             ST      R2,Save2       ; are needed by POP.
                ST      R1,Save1
                LD      R1,BASE        ; BASE contains -x3FFF.
                ADD     R1,R1,#-1      ; R1 contains -x4000.
                ADD     R2,R6,R1       ; Compare stack pointer to x4000
                BRz     fail_exit      ; Branch if stack is empty.
                LDR     R0,R6,#0       ; The actual "pop."
                ADD     R6,R6,#1       ; Adjust stack pointer
                BRnzp   success_exit
PUSH            ST      R2,Save2       ; Save registers that
                ST      R1,Save1       ; are needed by PUSH.
                LD      R1,MAX         ; MAX contains -x3FFB
                ADD     R2,R6,R1       ; Compare stack pointer to -x3FFB
                BRz     fail_exit      ; Branch if stack is full.
                ADD     R6,R6,#-1      ; Adjust stack pointer
                STR     R0,R6,#0       ; The actual "push"
success_exit    LD      R1,Save1       ; Restore original
                LD      R2,Save2       ; register values.
                AND     R5,R5,#0       ; R5 <-- success.
                RET
fail_exit       LD      R1,Save1       ; Restore original
                LD      R2,Save2       ; register values.
                AND     R5,R5,#0
                ADD     R5,R5,#1       ; R5 <-- failure.
                RET
BASE            .FILL   xC001          ; BASE contains -x3FFF.
MAX             .FILL   xC005
Save1           .FILL   x0000
Save2           .FILL   x0000
```

ADD Operands on Stack

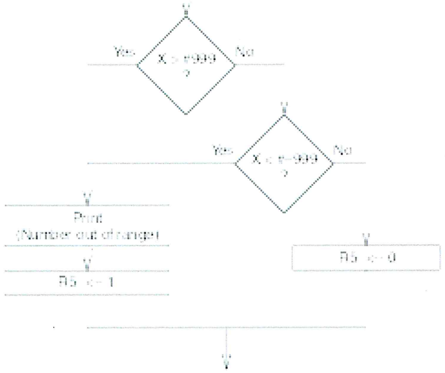


```
;
;         Routine to pop the top two elements from the stack,
;         add them, and push the sum onto the stack.  R6 is
;         the stack pointer.
;
OpAdd         JSR      POP              ; Get first source operand.
              ADD      R5,R5,#0         ; Test if POP was successful.
              BRp      Exit             ; Branch if not successful.
              ADD      R1,R0,#0         ; Make room for second operand
              JSR      POP              ; Get second source operand.
              ADD      R5,R5,#0         ; Test if POP was successful.
              BRp      Restore1         ; Not successful, put back first.
              ADD      R0,R0,R1         ; THE Add.
              JSR      RangeCheck       ; Check size of result.
              BRp      Restore2         ; Out of range, restore both.
              JSR      PUSH             ; Push sum on the stack.
              RET                       ; On to the next task...
Restore2      ADD      R6,R6,#-1        ; Decrement stack pointer.
Restore1      ADD      R6,R6,#-1        ; Decrement stack pointer.
Exit          RET
```

SHOULD NOT BE A SUBROUTINE

ELSE: NEED TO PUSH RETURN
ADDRESS (R7) ON STACK,
POP IT BEFORE RET

Check for Correct Range of Operands

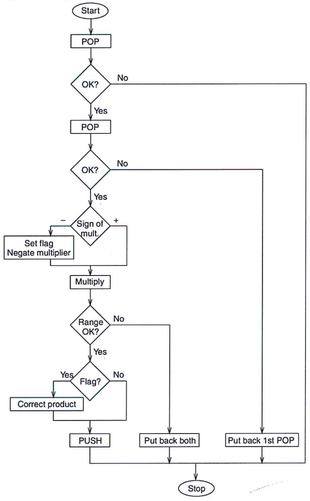


```
;
;     Routine to check that the magnitude of a value is
;     between -999 and +999.
;
RangeCheck     LD       R5,Neg999
               ADD      R4,R0,R5    ; Recall that R0 contains the
               BRp      BadRange    ; result being checked.
               LD       R5,Pos999
               ADD      R4,R0,R5
               BRn      BadRange
               AND      R5,R5,#0    ; R5 <-- success
               RET
BadRange       ST       R7,Save     ; R7 is needed by TRAP/RET
               LEA      R0,RangeErrorMsg
               TRAP     x22         ; Output character string
               LD       R7,Save
               AND      R5,R5,#0    ;
               ADD      R5,R5,#1    ; R5 <-- failure
               RET
Neg999         .FILL    #-999
Pos999         .FILL    #999
Save           .FILL    x0000
RangeErrorMsg  .FILL    x000A
               .STRINGZ "Error: Number is out of range."
```

!!
(JSRs)

BUT MAY WORK IF NO JSR,S IN CODE

OpMult (Multiply top two stack elements)



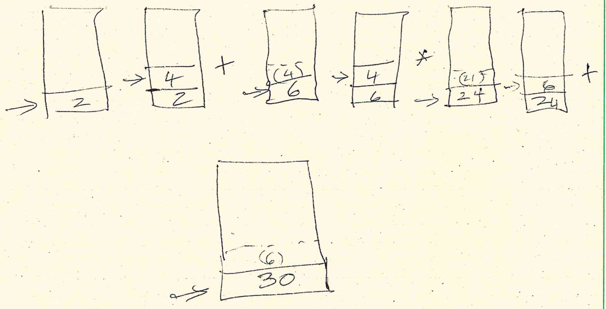
```
;
;    Algorithm to pop two values from the stack, multiply them
;    and if their product is within the acceptable range, push
;    the result on the stack.  R6 is stack pointer.
;
OpMult          AND    R3,R3,#0      ; R3 holds sign of multiplier.
                JSR    POP           ; Get first source from stack.
                ADD    R5,R5,#0      ; Test for successful POP
                BRp    Exit          ; Failure
                ADD    R1,R0,#0      ; Make room for next POP
                JSR    POP           ; Get second source operand
                ADD    R5,R5,#0      ; Test for successful POP
                BRp    Restore1      ; Failure; restore first POP
                ADD    R2,R0,#0      ; Moves multiplier, tests sign
                BRzp   PosMultiplier
                ADD    R3,R3,#1      ; Sets FLAG: Multiplier is neg
                NOT    R2,R2
                ADD    R2,R2,#1      ; R2 contains -(multiplier)
PosMultiplier   AND    R0,R0,#0      ; Clear product register
                ADD    R2,R2,#0
                BRz    PushMult      ; Multiplier = 0, Done.
;
MultLoop        ADD    R0,R0,R1      ; THE actual "multiply"
                ADD    R2,R2,#-1     ; Iteration Control
                BRp    MultLoop
;
                JSR    RangeCheck
                ADD    R5,R5,#0      ; R5 contains success/failure
                BRp    Restore2
;
                ADD    R3,R3,#0      ; Test for negative multiplier
                BRz    PushMult
                NOT    R0,R0         ; Adjust for
                ADD    R0,R0,#1      ; sign of result
PushMult        JSR    PUSH          ; Push product on the stack.
                RET
Restore2        ADD    R6,R6,#-1     ; Adjust stack pointer.
Restore1        ADD    R6,R6,#-1     ; Adjust stack pointer.
Exit            RET
```

NEED TO PUSH R7 ON STACK -

STACK ARITHMETIC

(2 + 4) * 4 + 6

PUSH/POP
~~2~~



REPRESENT AS:

2 4 + 4*6 +

3+4 * (5+6*(7+8))

⇓

3 4 5 6 7 8 + * + * +

A*(B+C)*D

B C + A * D *

A B C + * D *

D A B C + * *