

Notes: TRAPs and Subroutines

(Chapter 9)

A. Why?

TRAPs and subroutines use different mechanisms to go to to the code they execute but use similar mechanisms to return. Unlike normal subroutines, TRAPs allow controlled access to unsafe operations, like I/O.

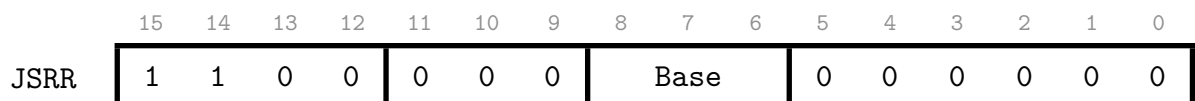
B. Outcomes

By the end of class you should:

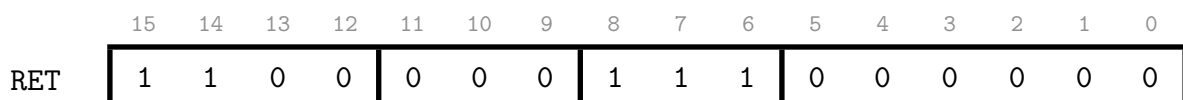
- Understand how the LC-3 jump to/return from subroutine commands work.
- How R7 is used in the LC-3 to hold return addresses.
- How the TRAP finds the code to execute for each given trap.

C. User-Written Subroutines

- The LC-3 uses the JSR and JSRR commands to Jump to a SubRoutine.
 - JSR takes assigned 11-bit PC offset to specify the address of the subroutine
 - JSRR uses a base register
 - Both instructions set $R7 \leftarrow PC$ before the jump to know where to return to.
 - JSR: $R7 \leftarrow PC$; $PC \leftarrow PC + \text{Sext}(\text{PCoffset11})$
 - JSRR: $R7 \leftarrow PC$; $PC \leftarrow \text{Base register}$



- Your subroutine code should save R7 in a local `.BLKW` if it changes R7.
 - Whic will happen if you call another subroutine (via JSR/JSRR)
 - Or if you call a TRAP (it also sets/uses R7 to hold the return address).
- To return from a subroutine call, use jump with R7 as the base register.
 - JMP R7 or the LC-3 assembler shorthand opcode, RET



Example: A subroutine that sets R0 to its negative

```

    ...
    LD    R0, X
    JSR   NEGATIVE    ; R0 <- -R0
    ST    R0, NEGX    ; Establish NEGX
    ...
X      .FILL 135      ; (random number for illustration)
NEGX   .BLKW 1       ; NEGX = -X
    ...
; Subroutine NEGATIVE
; Set R0 <- arithmetic negative of R0
; Affects no other registers.
;
NEGATIVE NOT    R0, R0    ; flip bits
        ADD    R0, R0, #1 ; add one
        RET                    ; return to caller

```

- Use JSRR for the call if you're not sure that NEGATIVE is close enough to use JSR:

```

    LD    R0, X
    LD    R1, LNEGATIVE ; pt R1 -> NEGATIVE
    JSRR R1                ; R0 <- -R0
    ST    R0, NEGX
    ...
LNEGATIVE .FILL NEGATIVE

```

D. How TRAPs Are Executed

- The TRAP command is similar to a subroutine jump because it saves R7.
- The trap vector determines the address to jump to
 - If the trap vector is *vec*, then we jump to the location specified at location V.
- TRAP *vec*: $R7 \leftarrow PC$; $PC \leftarrow M[\text{Zero extend}(vec)]$.
 - E.g. for TRAP x21, we set $R7 \leftarrow PC$ then $PC \leftarrow M[x0021]$.
- Memory locations x0000 – x00FF comprise a table of addresses to handle TRAPs.

E. Practical Aspects of Subroutines

- If you're writing a subroutine
 - Begin by saving the registers you will modify (unless you will store results in them).
 - Save R7 if you're going to call any other subroutines or TRAPs.
 - Save R0 if you're going to use TRAP x20 or x23 (they input a character).
 - Before you return from the subroutine, restore the registers you saved.
 - Then use a return instruction (RET = JMP R7).

F. Sample program

```

; Test find character routine
;
    .ORIG x3000
    LD    R0, LOOKFOR
    LEA   R1, STR1
    JSR   FindChar
    TRAP  x25          ; halt

LOOKFOR    .STRINGZ "X"
STR1      .STRINGZ "abcxyzXYZ X"

; FindChar: subroutine to find first occurrence of a char
;
; R0 = char to search for
; R1 pt location to start search at
; on return, R2 pt location of character if found (pt end of string
o/w)
;
FindChar   ST    R3, FCR3          ; Save registers
           ST    R4, FCR4          ; Save original char

           NOT   R4, R0            ;
           ADD   R4, R4, #1        ; R4 = - char to search for
           ADD   R2, R1, #0        ; Init ptr -> 1st char in string

FC1  LDR   R3, R2, #0 ; Get curr character
      BRz  FC2          ; If null, search fails
      ADD  R3, R3, R4   ; See if matches input char
      BRz  FC2          ; If yes, search succeeds
      ADD  R2, R2, #1   ; If no, pt next char in string
      BR   FC1          ; ... and continue loop

FC2  LD    R3, FCR3          ; Restore registers
      LD    R4, FCR4          ; ...
      RET                          ; ... and return

FCR3 .BLKW 1 ; Save area for R3
FCR4 .BLKW 1 ; Save area for R4
.END

```