System Calls

- Certain operations require specialized knowledge and protection:
  - specific knowledge of I/O device registers and the sequence of operations needed to use them
  - I/O resources shared among multiple users/programs; a mistake could affect lots of other users!
- Not every programmer knows (or wants to know) this level of detail
- Provide service routines or system calls (part of operating system) to safely and conveniently perform low-level, privileged operations
System Call

1. User program invokes system call
2. Operating system code performs operation
3. Returns control to user program
4. In LC-3, this is done through the TRAP mechanism

LC-3 TRAP Mechanism

1. A set of service routines
   - A part of operating system -- routines start at arbitrary addresses
   - Convention is that system code is below x3000
   - Up to 256 routines
2. Table of starting addresses
   - Stored at x0000 through x00FF in memory
   - Called System Control Block in some architectures
3. TRAP instruction
   - Used by program to transfer control to operating system
   - 8-bit trap vector names one of the 256 service routines
4. A link back to the user program
   - Want execution to resume immediately after the TRAP instruction
Memory map

1. User program executes TRAP: load indirect address of TRAP routine code
2. Execute TRAP routine and RETurn to instruction following the TRAP in user program

TRAP

- Trap vector

TRAP 1 1 1 1 0 0 0 0  trapvect8

- 8-bit index into table of service routine addresses
- in LC-3, this table is stored in memory at 0x0000 – 0x00FF
- 8-bit trap vector is zero-extended into 16-bit memory address

- Where to go
  - lookup starting address from table; place in PC

- How to get back
  - save address of next instruction (current PC) in R7
How do we transfer control back to instruction following the TRAP?
We saved old PC in R7
  - `JMP R7` gets us back to the user program at the right spot
  - LC-3 assembly language lets us use `RET` (return) in place of `JMP R7`

Must make sure that service routine does not change R7, or we won’t know where to return.
Example of TRAP Instruction

```assembly
.ORIG x3000
LD R2, TERM ;
LD R3, ASCII ;
AGAIN TRAP x23 ; Input character
ADD R1, R2, R0 ;
BRz EXIT ;
ADD R0, R0, R3 ; Change to lowercase
TRAP x21 ; Output to monitor...
BRnzp AGAIN ;
TERM .FILL xFFC9 ; '7'
ASCII .FILL x0020 ; lowercase bit
EXIT TRAP x25 ; halt
.END
```

What does this program do?

Output Service Routine

```assembly
.ORIG x0430 ; syscall address
ST R7, SaveR7 ; save R7 & R1
ST R1, SaveR1 ;
; ----- Write character
TryWrite LDI R1, CRTSR ; get status
BRzp TryWrite ; look for bit 15 on
WriteIt STI R0, CRTDR ; write char
; ----- Return from TRAP
Return LD R1, SaveR1 ; restore R1 & R7
LD R7, SaveR7
RET ; back to user
CRTSR .FILL xFE04
CRTDR .FILL xFE06
SaveR1 .FILL 0
SaveR7 .FILL 0
.END
```

stored in table, location x21
TRAP Mechanism Operation

1. Lookup starting address.
2. Transfer to service routine.
3. Return (JMP R7).

TRAP Routines and their Assembler Names

<table>
<thead>
<tr>
<th>vector</th>
<th>symbol</th>
<th>routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>x20</td>
<td>GETC</td>
<td>read a single character (no echo)</td>
</tr>
<tr>
<td>x21</td>
<td>OUT</td>
<td>output a character to the monitor</td>
</tr>
<tr>
<td>x22</td>
<td>PUTS</td>
<td>write a string to the console</td>
</tr>
<tr>
<td>x23</td>
<td>IN</td>
<td>print prompt to console, read and echo character from keyboard</td>
</tr>
<tr>
<td>x25</td>
<td>HALT</td>
<td>halt the program</td>
</tr>
</tbody>
</table>
Saving and Restoring Registers

- Must save the value of a register if:
  - Its value will be destroyed by the service routine, and
  - The value will be used after that action

- Who saves? Caller- vs callee-saved
  - The caller of the service routine?
    - Knows what it needs later, but may not know what gets altered by called routine
  - The called service routine (callee)?
    - Knows what it alters, but does not know what will be needed later by calling routine

Example

```
LEA R3, Binry
LD R6, ASCII ; char->digit template
LD R7, COUNT ; initialize to 10
AGAIN: TRAP x23 ; Get char
ADD R0, R0, R6 ; convert to number
STR R0, R3, #0 ; store number
ADD R3, R3, #1 ; incr pointer
ADD R7, R7, #1 ; decr counter
BRp AGAIN ; more?
BRnzp NEXT
ASCII .FILL xFFD0
COUNT .FILL #10
Binry .BLKW #10
```

What's wrong with this routine?
Saving and Restoring Registers

- Called routine -- “callee-save”
  - Before start, save any registers that will be altered (unless altered value is desired by calling program!)
  - Before return, restore those same registers
- Calling routine -- “caller-save”
  - Save registers destroyed by own instructions or by called routines (if known), if values needed later
    - Save R7 before TRAP
    - Save R0 before TRAP x23 (input character)
  - Or avoid using those registers altogether
- Values are saved by storing them in memory

Recommended exercises

- Ex 9.2, 9.4, 9.5, 9.11
- Ex 9.17, 9.18