Chapter 9
TRAP Routines and Subroutines

System Calls

Some ops. require specialized knowledge and protection

- **Knowledge of I/O device registers and how to use them**
  Programmers don't want to know this!
- **Protection for shared I/O resources**
  Want process isolation

Solution: *service routines* or *system calls*

- **Low-level, privileged operations performed by operating system**

System Call

1. User program invokes system call
2. Operating system code performs operation
3. Returns control to user program

In LC-3: done via *TRAP mechanism*

LC-3 TRAP Mechanism

1. Provides set of service routines

- **Part of operating system -- routines start at arbitrary addresses**
  (by convention system code is below x3000)
- **Up to 256 routines**

2. Requires table of starting addresses

- **Stored in memory (x0000 through x00FF)**
- **Used to associate code with trap number**
- **Called System Control Block or Trap Vector Table**

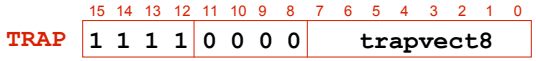
3. Uses TRAP instruction

- **Used by program to transfer control to operating system (w/ privileges)**
- **8-bit trap vector names one of the 256 service routines**

4. Uses "RTT" instruction

- **Returns control to the user program (w/o privileges)**
- **Execution resumes immediately after the TRAP instruction**

TRAP Instruction



Trap vector

- Identifies which system call to invoke
- Serves as index into table of service routine addresses
  - LC-3: table stored in memory at 0x0000 – 0x00FF
  - 8-bit trap vector zero-extended to form 16-bit address

Where to go

- Lookup starting address from table; place in PC

Enabling return

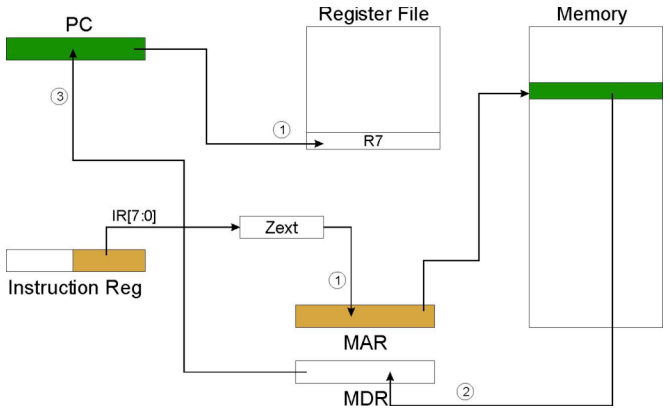
- Save address of next instruction (current PC) in R7

How to return

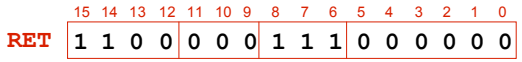
- Place address in R7 in PC

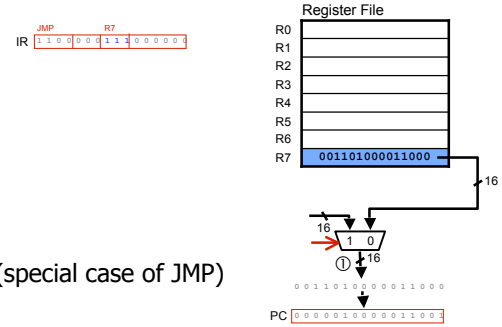
TRAP

NOTE: PC has already been incremented during instruction fetch stage.



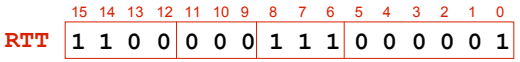
RET

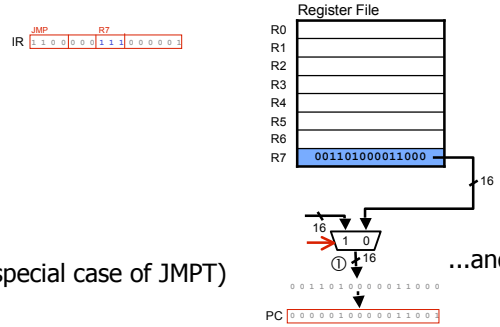




(special case of JMP)

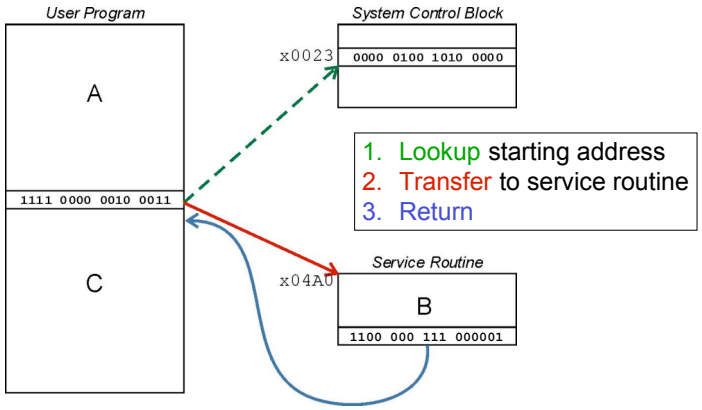
RTT





(special case of JMPT) ...and exit priv. mode

TRAP Mechanism Operation



TRAP Routines and Their Assembler Names

vector	symbol	routine
x20	GETC	Read single character (no echo)
x21	OUT	Output character to console
x22	PUTS	Write string to console
x23	IN	Print prompt to console, read and echo character from keyboard
x25	HALT	Halt program

Example: Using the TRAP Instruction

```
            .ORIG x3000
            LD    R2, TERM    ; Load negative ASCII '7'
            LD    R3, ASCII   ; Load ASCII difference
AGAIN       TRAP  x23         ; Input character
            ADD   R1, R2, R0  ; Test for terminating char
            BRz   EXIT        ; Exit if done
            ADD   R0, R0, R3  ; Change to lowercase
            TRAP  x21         ; Output to monitor...
            BRnzp AGAIN       ; ... and again repeat...
TERM        .FILL xFFC9       ; -'7'
ASCII       .FILL x0020       ; Lowercase bit
EXIT        TRAP  x25         ; Halt
            .END
```


Example: Character Output Service Routine

```
            .ORIG x0430       ; Syscall x21 address
             ST   R1, SaveR1  ; Save R1

; Write character
TryWrite    LDI   R1, DSR     ; Get status
            BRzp  TryWrite    ; Bit 15 says not ready?
WriteIt     STI   R0, DDR     ; Write char
; Return from TRAP
Return      LD    R1, SaveR1  ; Restore R1
            RET               ; Return from trap
DSR         .FILL xFE04
DDR         .FILL xFE06
SaveR1      .FILL 0
SaveR7      .FILL 0
            .END
```

stored in table, location x21

Another Example

```
          LEA   R3, Block   ; Init. to first loc.
          LD    R6, ASCII   ; Char->digit template
          LD    R7, COUNT   ; Init. 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_TASK
ASCII     .FILL xFFD0       ; Negative of x0030
COUNT     .FILL #10
Block     .BLKW #10
```

What's wrong with this code?

Saving and Restoring Registers

Must save the value of a register if. . .

- **Its value will be destroyed by service routine, and**
- **We will need to use the value later**

Who saves?

- **Caller of service routine?**
- **Called service routine?**

Caller of Service Routine

What does the caller know?

- **Knows what it needs later**
- **May/should not know what gets altered by service routine**

Example

```
          LEA     R3, Block  ; Init. to first loc.
          LD      R6, ASCII  ; Char->digit template
          LD      R7, COUNT  ; Init. 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_TASK
ASCII     .FILL   xFFD0      ; Negative of x0030
COUNT     .FILL   #10
Block     .BLKW   #10
```


Called Service Routine (Callee)

What does the callee know?

- **Knows what it alters**
- **Does not know what will be needed later (by calling routine)**

Example

```
          .ORIG x0430        ; Syscall x21 address
TryWrite  LDI    R1, DSR     ; Get status
          BRzp   TryWrite    ; Bit 15 says not ready?
WriteIt   STI    R0, DDR     ; Write char
Return    RET                ; Return from trap
DSR       .FILL  xFE04
DDR       .FILL  xFE06
          .END
```

Saving and Restoring Registers

Called routine ⇒ *"callee-save"*

- **Before start, save registers that will be altered** (unless altered value is desired by calling program!)
- **Before return, restore those same registers**
- **Values are saved by storing them in memory**

Calling routine ⇒ *"caller-save"*

- **If register value needed later, save register destroyed by own instructions or by called routines (if known)**
  - **Save R7 before TRAP**
  - **Save R0 before TRAP x23 (input character)**
- **Or avoid using those registers altogether**

LC-3: By convention, callee-saved when possible

Privilege

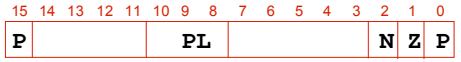
Goal: Isolation

- **OS performs I/O (in traps)**
- **Application can't perform I/O directly**

How is this enforced?

Privilege: Processor modes

- **Privileged (supervisor)**
- **Unprivileged (user)**
- **Encoded in 15th bit of program status register (PSR)**



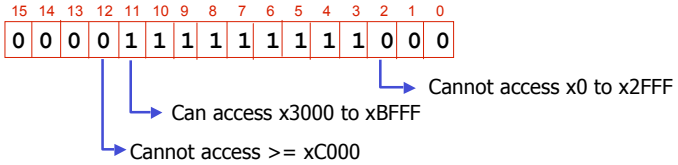
Supervisor Mode Versus User Mode

Supervisor mode

- **Program has access to resources not available to user programs**
- **LC-3: Memory**

User mode in LC-3

- **Memory access is limited by memory protection register (MPR)**
- **Each MPR bit corresponds to 4K memory segment**
- **1 indicates that users can access memory in this segment**



MPR

Note: MPR not in book!

Set (only) by OS

- **OS decides policy, HW enforces it**

Prevents user from. . .

- **Updating trap table**
- **Changing OS code (*i.e.,* trap handlers)**
- **Accessing video memory**
- **Accessing memory-mapped I/O registers (*e.g.,* DDR, DSR)**
- **Could be different for each application**

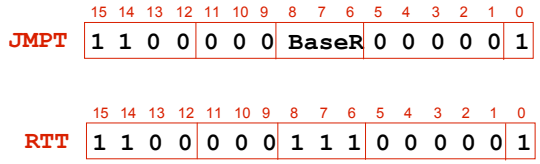
Managing Privilege

Who sets privilege bit in PSR?

- TRAP instruction

Who clears privilege bit?

- New instruction JMPT/RTT (Note: not in book!)



Question

Can a service routine call another service routine?

Can a service routine call itself?