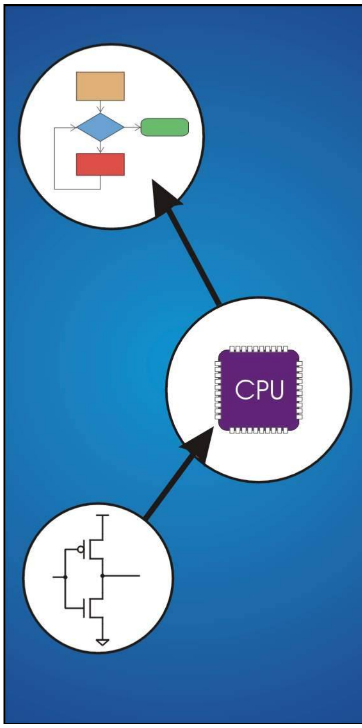


Chapter 9
TRAP Routines and Subroutines

Original slides from Gregory Byrd, North Carolina State University

Modified slides by C. Wilcox, S. Rajopadhye
Colorado State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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
- Solution: provide *service routines* or *system calls* (in operating system) to safely and conveniently perform low-level, privileged operations

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

System Call

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

In LC-3, this is done through the *TRAP mechanism*.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

LC-3 TRAP Mechanism

- *1. A set of service routines.*
  - 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 or Trap Vector Table
- *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 linkage back to the user program.*
  - want execution to resume immediately after the TRAP instruction

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

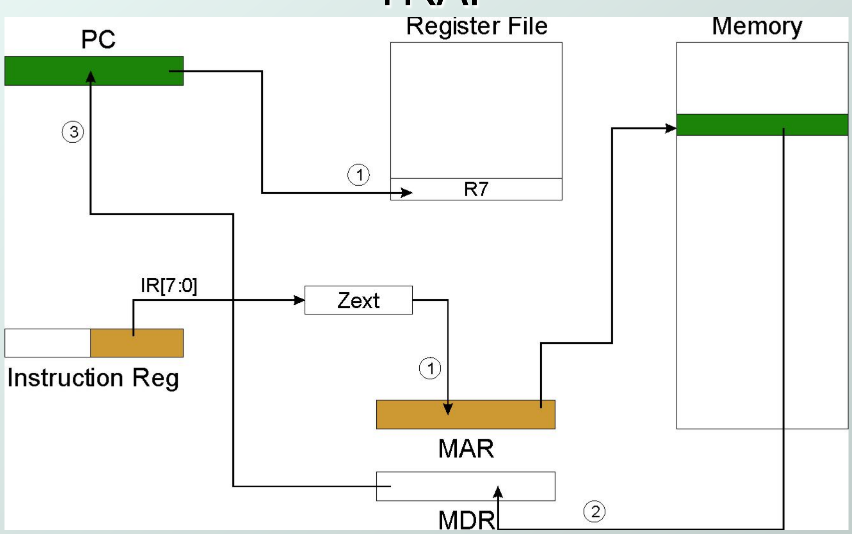
TRAP Instruction

	15	14	13	12	11	10	9	8	7–0
TRAP	1	1	1	1	0	0	0	0	trapvect8

- **Trap vector**
  - identifies which system call to invoke
  - 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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

TRAP



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

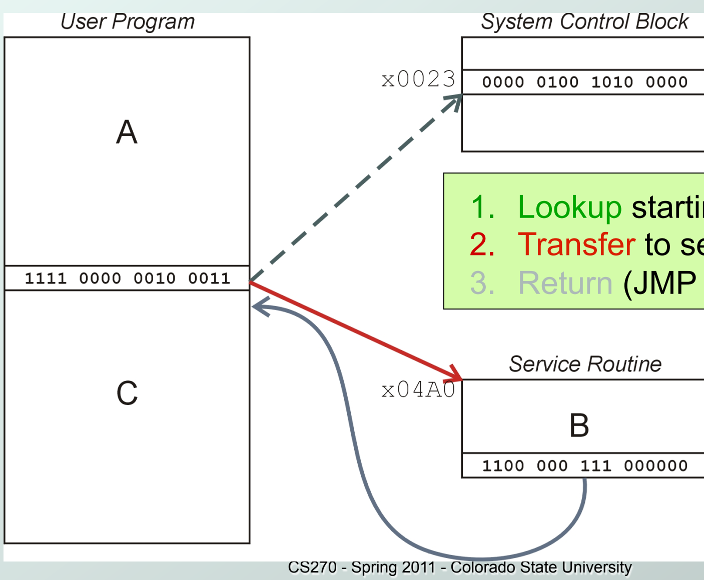
Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

RET (JMP 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.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

TRAP Mechanism Operation



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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Example 9.1: Using 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 terminate
        BRz EXIT         ; Exit if done
        ADD R0, R0, R3   ; Change to lowercase
        TRAP x21         ; Output to monitor...
        BRnzp  AGAIN     ; ... again and again...
TERM    .FILL xFFC9      ; -'7'
ASCII   .FILL x0020      ; lowercase bit
EXIT    TRAP x25         ; halt
        .END
```


Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Example: Output Service Routine

```
          .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 xF3FC
CRTDR          .FILL     xF3FF
SaveR1    .FILL      0
SaveR7    .FILL      0
          .END
```

stored in table, location x21

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

TRAP Routines and their Assembler Names

vector	*symbol*	*routine*
`x20`	`GETC`	**read a single character (no echo)**
`x21`	`OUT`	**output a character to the monitor**
`x22`	`PUTS`	**write a string to the console**
`x23`	`IN`	**print prompt to console, read and echo character from keyboard**
`x25`	`HALT`	**halt the program**

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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 after that action.
- Who saves?
  - caller of service routine?
    - knows what it needs later, but may not know what gets altered by called routine
  - called service routine?
    - knows what it alters, but does not know what will be needed later by calling routine

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Example

```
        LEA R3, Binary     ; load pointer
        LD  R6, ASCII      ; char to digit
        LD  R7, COUNT      ; initialize to 10
AGAIN   TRAP x23           ; get character
        ADD R0, R0, R6     ; convert to number
        STR R0, R3, #0     ; store number
        ADD R3, R3, #1     ; increment pointer
        ADD R7, R7, -1     ; decrement counter
        BRp AGAIN          ; more?
        BRnzp NEXT
ASCII   .FILL    xFFD0
COUNT   .FILL    #10
Binary  .BLKW #10
```

What's wrong with this routine?
What happens to R7?

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

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.***

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Question

- Can a service routine call another service routine?
- If so, is there anything special the calling service routine must do?

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

What about User Code?

- Service routines provide three main functions:
  1. Shield programmers from system-specific details.
  2. Write frequently-used code just once.
  3. Protect system resources from malicious/clumsy programmers.
- Are there any reasons to provide the same functions for non-system (user) code?

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Subroutines

- A **subroutine** is a program fragment that:
  - lives in user space
  - performs a well-defined task
  - is invoked (called) by another user program
  - returns control to the calling program when finished
- Like a service routine, but not part of the OS
  - not concerned with protecting hardware resources
  - no special privilege required
- Reasons for subroutines:
  - reuse useful (and debugged!) code without having to keep typing it in
  - divide task among multiple programmers
  - use vendor-supplied ***library*** of useful routines

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

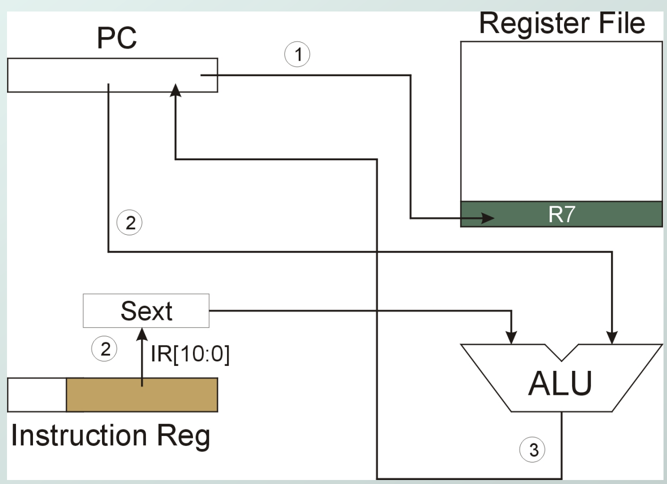
JSR Instruction

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSR	0	1	0	0	1	PCoffset11										

- Jumps to a location (like a branch but unconditional), and saves current PC (addr of next instruction) in R7.
  - saving the return address is called "linking"
  - target address is PC-relative **(PC + Sext(IR[10:0]))**
  - bit 11 specifies addressing mode
    - if =1, PC-relative: target address = PC + Sext(IR[10:0])
    - if =0, register: target address = contents of register IR[8:6]

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

JSR



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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

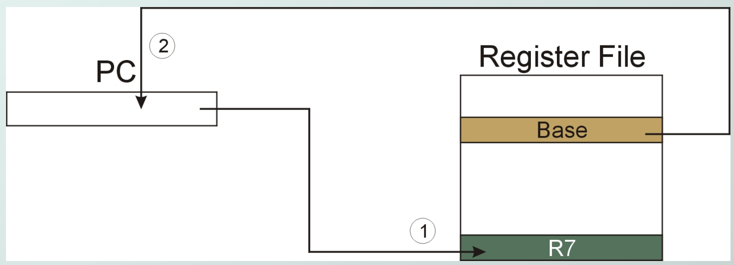
JSRR Instruction

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSRR	0	1	0	0	0	0	0	Base			0	0	0	0	0	0

- Just like JSR, except Register addressing mode.
  - target address is Base Register
  - bit 11 specifies addressing mode
- What important feature does JSRR provide that JSR does not?

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

JSRR



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

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Returning from a Subroutine

- RET (JMP R7) gets us back to the calling routine.
  - just like TRAP

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Example: Negate the value in R0

```
2sComp    NOT   R0, R0       ; flip bits
          ADD   R0, R0, #1   ; add one
          RET                ; return to caller
```

To call from a program (within 1024 instructions):

```
; need to compute R4 = R1 - R3
          ADD   R0, R3, #0   ; copy R3 to R0
          JSR   2sComp       ; negate
          ADD   R4, R1, R0   ; add to R1
          ...
```

Note: Caller should save R0 if we'll need it later!

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Passing Information to/from Subroutines

- Arguments
  - A value passed in to a subroutine is an argument.
  - This is a value needed by the subroutine to do its job.
  - Examples:
    - In 2sComp routine, R0 is the number to be negated
    - In OUT service routine, R0 is the character to be printed.
    - In PUTS routine, R0 is *address* of string to be printed.
- Return Values
  - A value passed out of a subroutine is a return value.
  - You called the subroutine to compute this value!
  - Examples:
    - In 2sComp routine, negated value is returned in R0.
    - GETC service routine returns char from the keyboard in R0.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Using Subroutines

- In order to use a subroutine, a programmer must know:
  - **its address** (or at least a label that will be bound to its address)
  - **its function** (what does it do?)
    - NOTE: The programmer does not need to know ***how*** the subroutine works, but what changes are visible in the machine's state after the routine has run.
  - **its arguments** (where to pass data in, if any)
  - **its return values** (where to get computed data, if any)

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Saving and Restore Registers

- Since subroutines are just like service routines, we also need to save and restore registers, if needed.
- Generally use "callee-save" strategy, except for return values.
  - Save anything that the subroutine will alter internally that shouldn't be visible when the subroutine returns.
  - It's good practice to restore incoming arguments to their original values (unless overwritten by return value).
- *Remember*: You MUST save R7 if you call any other subroutine or service routine (TRAP).
  - Otherwise, you won't be able to return to caller.

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Example

(1) Write a subroutine FirstChar to:

find the first occurrence
of a particular character (in R0)
in a string (pointed to by R1);
return pointer to character or to end of string (NULL) in R2.

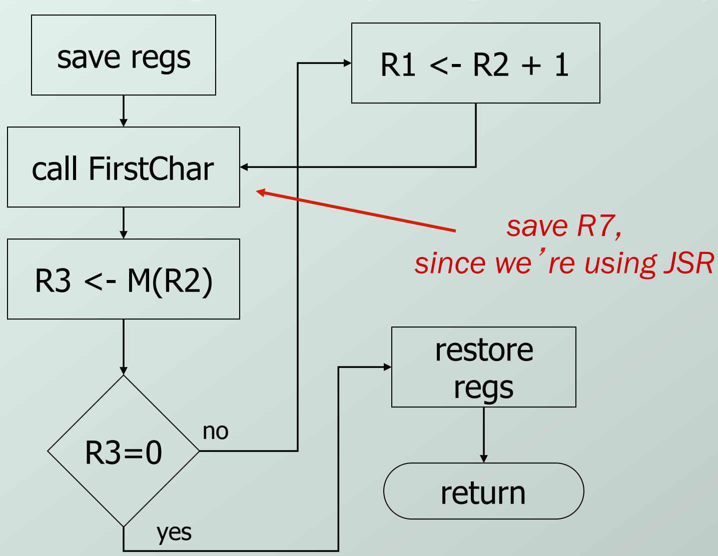
(2) Use FirstChar to write CountChar, which:

counts the number of occurrences
of a particular character (in R0)
in a string (pointed to by R1);
return count in R2.

- Can write the second subroutine first, without knowing the implementation of FirstChar!

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

CountChar Algorithm (using FirstChar)



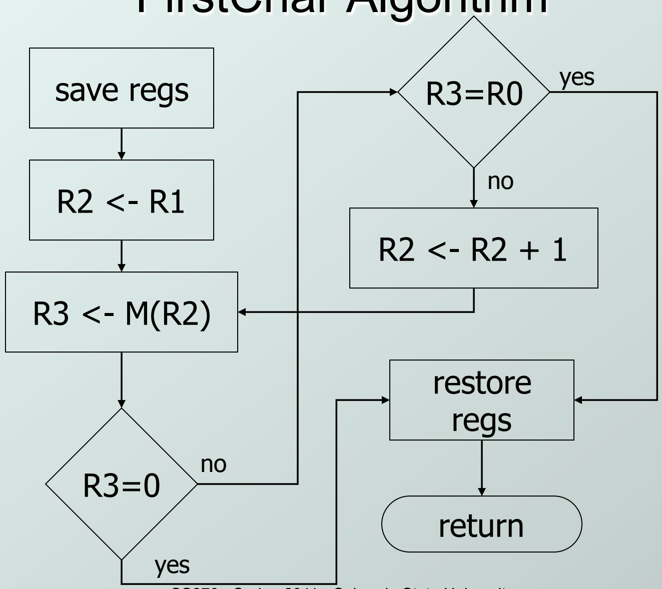
Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

CountChar Implementation

```
; subroutine to count occurrences of a char CountChar
        ST      R3, CCR3     ; save registers
        ST      R4, CCR4
        ST      R7, CCR7     ; JSR alters R7
        ST      R1, CCR1     ; save original pointer
        AND     R4, R4, #0   ; count = 0
CC1     JSR     FirstChar    ; find next occurrence
        LDR     R3, R2, #0   ; null?
        BRz     CC2          ; done if null
        ADD     R4, R4, #1   ; increment count
        ADD     R1, R2, #1   ; increment pointer
        BRnzp      CC1
CC2     ADD     R2, R4, #0   ; return value to R2
        LD      R3, CCR3     ; restore regs
        LD      R4, CCR4
        LD      R1, CCR1
        LD      R7, CCR7
        RET
```


Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

FirstChar Algorithm



Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

FirstChar Implementation

; subroutine to find first occurrence of a char `FirstChar`

```
      ST     R3, FCR3    ; save registers
      ST     R4, FCR4    ; save original char
      NOT    R4, R0      ; negate for comparisons
      ADD    R4, R4, #1
      ADD    R2, R1, #0  ; initialize pointer
FC1   LDR    R3, R2, #0  ; read character
      BRz    FC2         ; if null, we're done
      ADD    R3, R3, R4  ; see if matches input
      BRz    FC2         ; if yes, we're done
      ADD    R2, R2, #1  ; increment pointer
      BRnzp FC1
FC2   LD     R3, FCR3    ; restore registers
      LD     R4, FCR4
      RET
```


Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

Library Routines

- Vendor may provide object files containing useful subroutines
  - don't want to provide source code -- intellectual property
  - assembler/linker must support EXTERNAL symbols (or starting address of routine must be supplied to user)

```
        .EXTERNAL SQRT
...
        LD    R2, SQAddr   ; load SQRT addr
        JSRR  R2
...
SQAddr  .FILL     SQRT
```

- Using JSRR, because we don't know whether SQRT is within 1024 instructions.