

15. Subroutines (Chapter 9)

October 22, 2018

• Review

- LC-3 TRAP Routines
- TRAP mechanism
- Saving and restoring registers

• Subroutines

- Calling and return
- Passing parameters
- Examples

TRAP Instruction

TRAP
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
1 1 1 1
0 0 0 0
trapvect8

Trap vector

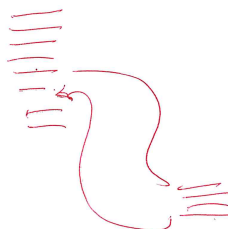
- Identifies which system call to invoke
- 8-bit index into table of service routine addresses
 - in LC-3, this table is stored in memory at **0x0000 – 0x00FF**
 - 8-bit trap vector is zero-extended into 16-bit memory address

Where to go

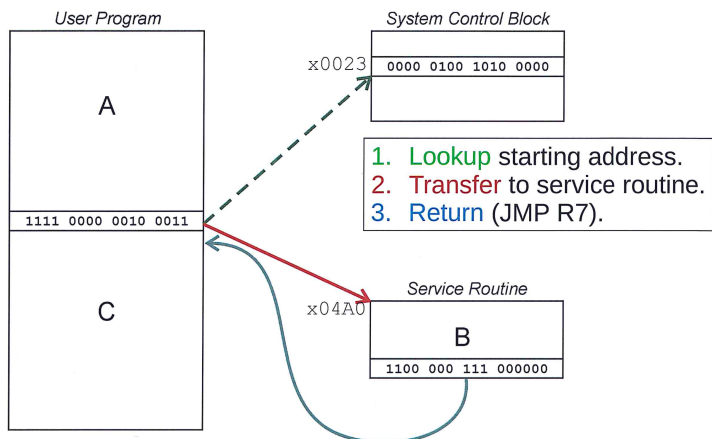
- Look up starting address from table; place in PC

How to get back

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



TRAP Mechanism Operation



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

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

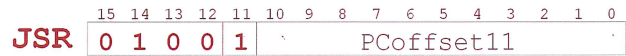
Like a service routine, but not part of the OS

- 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

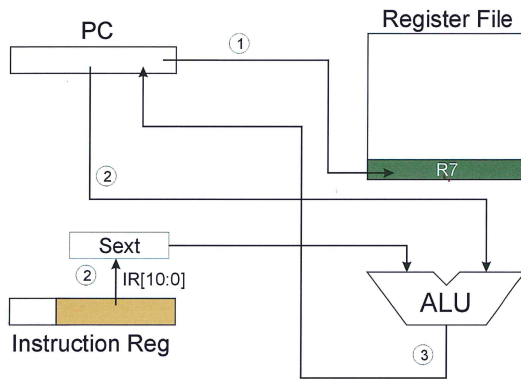
JSR Instruction



Jumps to a location (like a branch but unconditional), and saves current PC (addr of next instruction) in R7.

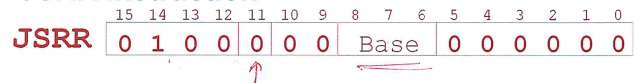
- saving the return address is called "linking"
- target address is PC-relative (PC + Sext(IR[10:0]))
- bit 11 specifies addressing mode
 - if =1, PC-relative: target address = PC + Sext(IR[10:0])
 - if =0, register: target address = contents of register IR[8:6]

JSR



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

JSRR Instruction

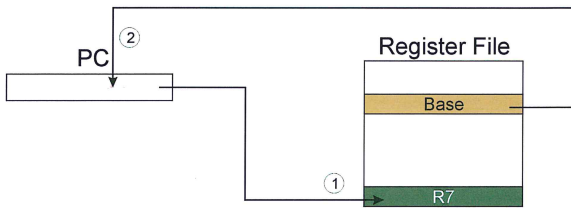


Just like JSR, except Register addressing mode.

- target address is Base Register
- bit 11 specifies addressing mode

What important feature does JSRR provide that JSR does not?

JSRR



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

Returning from a Subroutine

RET (JMP R7) gets us back to the calling routine.

- just like TRAP

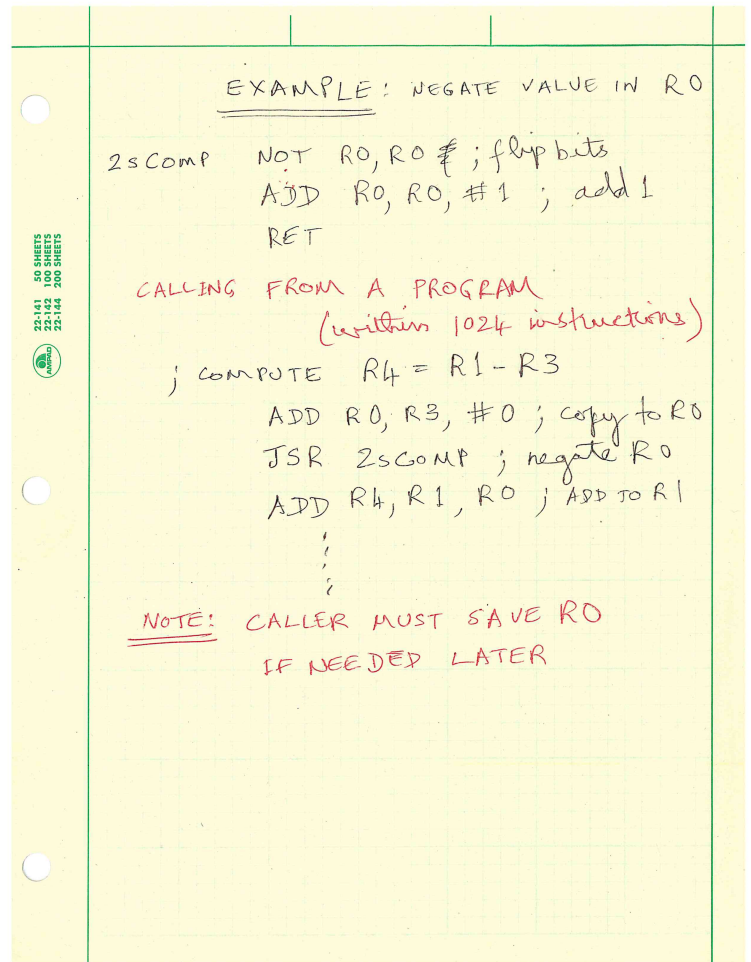
Passing Information to/from Subroutines

Arguments

- A value **passed in** to a subroutine is called 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 called a return value.
- This is the value that you called the subroutine to compute.
- Examples:
 - In 2sComp routine, negated value is returned in R0.
 - In GETC service routine, character read from the keyboard is returned in R0.



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)

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.

Example

(1) Write a subroutine **FirstChar** to:

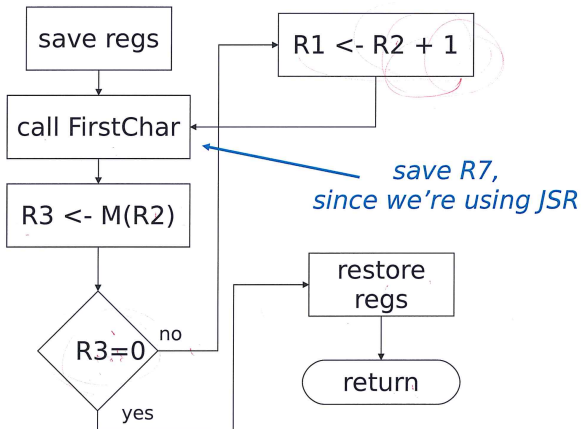
find the **first** occurrence of a particular **character** (in R0) in a **string** (pointed to by R1); return **pointer** to character or to end of string (NULL) in R2.

(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! ||

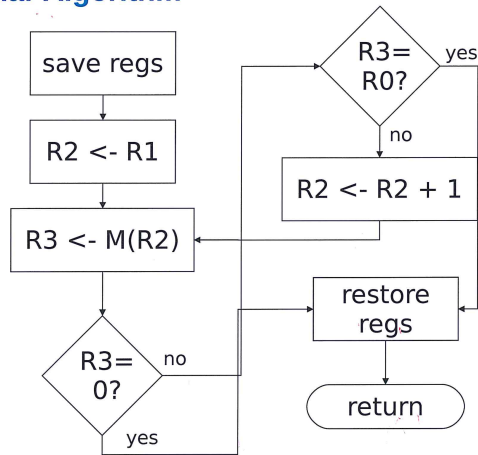
CountChar Algorithm (using FirstChar)



CountChar Implementation

```
; CountChar: 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 string ptr
    AND   R4, R4, #0  ; initialize count to zero
CC1     JSR   FirstChar ; find next occurrence (ptr in R2)
    LDR   R3, R2, #0  ; see if char or null
    BRz   CC2         ; if null, no more chars
    ADD   R4, R4, #1  ; increment count
    ADD   R1, R2, #1  ; point to next char in string
    BRnzp CC1
CC2     ADD   R2, R4, #0 ; move return val (count) to R2
    LD    R3, CCR3    ; restore regs
    LD    R4, CCR4
    LD    R1, CCR1
    LD    R7, CCR7
    RET                                ; and return
```

FirstChar Algorithm



FirstChar Implementation

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

```

ST    R3, FCR3    ; save registers
ST    R4, FCR4    ; save original char
NOT   R4, R0      ; negate R0 for comparisons
ADD   R4, R4, #1
ADD   R2, R1, #0  ; initialize ptr to beginning of string
FC1   LDR    R3, R2, #0 ; read character
      BRZ   FC2      ; if null, we're done
      ADD  R3, R3, R4 ; see if matches input char
      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   ; and return
  
```

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.

```

;: CountChar: subroutine to count occurrences of a char
.ORIG  x3000
CountChar
    ST    R3, CCR3
    ST    R4, CCR4
    ST    R7, CCR7
    JSR  alters R7
    ; save original string ptr
    ; initialize count to zero
    ; find next occurrence (ptr in R2)
    ; see if char or null
    ; if null, no more chars
    ; increment count
    ; point to next char in string
    ; move return val (count) to R2
    ; restore regs
    HALT

CC1   FirstChar
    ;: FirstChar: subroutine to find first occurrence of a char
    ; save registers
    ; save original char
    ; negate R0 for comparisons
    ; initialize ptr to beginning of string
    ; read character
    ; if null, we're done
    ; see if matches input char
    ; if yes, we're done
    ; increment pointer
    ; restore registers
    ; and return
    ST    R3, FCR3
    ST    R4, FCR4
    NOT   R4, R0
    ADD   R4, R4, #1
    ADD   R2, R1, #0
    FC1   LDR    R3, R2, #0
    BRZ   FC2
    ADD  R3, R3, R4
    BRZ  FC2
    ADD  R2, R2, #1
    BRnzp FC1
    LD    R3, FCR3
    LD    R4, FCR4
    RET

CCR1   .BLKW 1 ;
CCR3   .BLKW 1 ;
CCR4   .BLKW 1 ;
CCR7   .BLKW 1 ;
FCR3   .BLKW 1 ;
FCR4   .BLKW 1 ;
.END
  
```