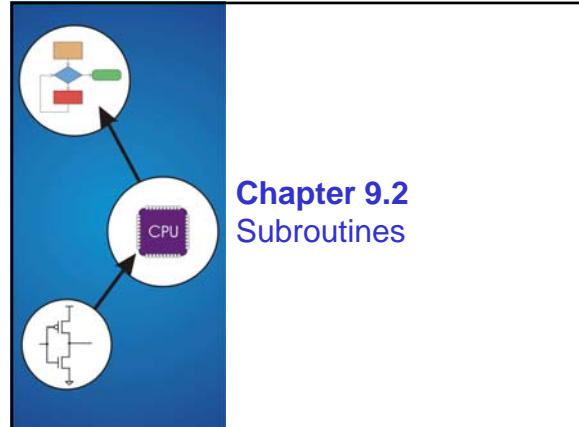




Introduction to Computer Engineering

ECE 252, Fall 2010
 Prof. Mikko Lipasti
 Department of Electrical and Computer Engineering
 University of Wisconsin – Madison



Chapter 9.2 Subroutines

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

Skipping Ahead to Chapter 9

You will need to use **subroutines** for programming assignments

- Read Section 9.2

A **subroutine** is a program fragment that:

- performs a well-defined task
- is invoked (called) by another user program
- returns control to the calling program when finished

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

7-3

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

JSR Instruction

JSR 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
 0 1 0 0 1 POffset11

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

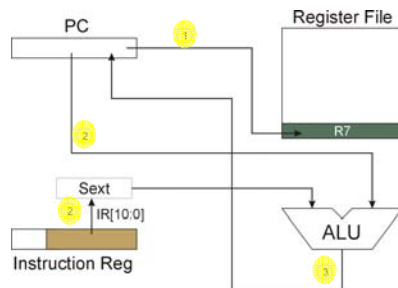
- saving the return address is called “linking”
- target address is PC-relative ($PC + \text{Sext}(\text{IR}[10:0])$)
- bit 11 specifies addressing mode
 - if =1, PC-relative: target address = $PC + \text{Sext}(\text{IR}[10:0])$
 - if =0, register: target address = contents of register IR[8:6]

NOTE: TRAP instruction also “links” return address by writing PC into R7

7-4

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

JSR



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

7-5

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

JSRR Instruction

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

Just like JSR, except Register addressing mode.

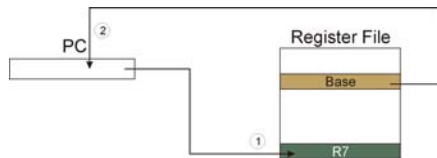
- target address is Base Register
- bit 11 specifies addressing mode

What important feature does JSRR provide that JSR does not?

- Subroutine (target) address can be anywhere in memory
- Target address can change
 - Virtual functions and function pointers

7-6

JSRR



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

7-7

RET (JMP R7)

How do we transfer control back to instruction following the subroutine?

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".

```
JMP 1 1 0 0 0 0 0 0 Base 0 0 0 0 0 0
```

Must make sure that subroutine does not change R7, or we won't know where to return.

9-8

Question

Can a subroutine call another subroutine or TRAP?

- Yes, of course
- Common in complex programs

If so, is there anything special the calling subroutine must do?

- Make sure its return address is not "lost"
- On entry to subroutine, R7 contains its return address
- When it invokes JSR again, R7 is overwritten
- Must save R7 in memory (store) before call
- Must restore R7 from memory (load) after call

9-9

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!

7-10

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.

7-11

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)

7-12

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.

7-13

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**.

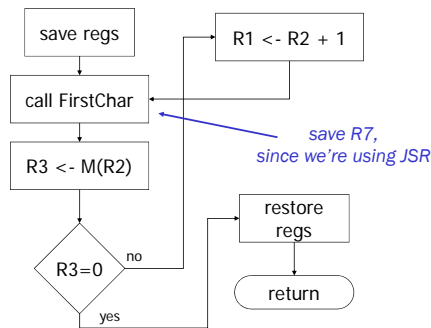
(2) Use FirstChar to write **CountChar**, which:

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!

7-14

CountChar Algorithm (using FirstChar)



7-15

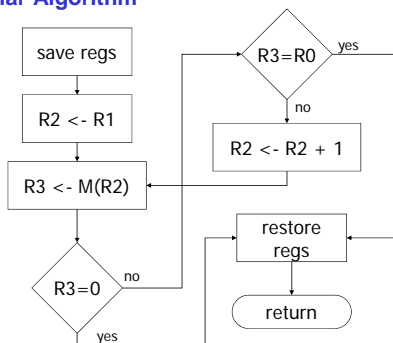
CountChar Implementation

```

; CountChar: subroutine to count occurrences of a char
CountChar
    ST    R3, CCR3    ; save registers
    ST    R4, CCR4    ; save original string ptr
    ST    R7, CCR7    ; JSR alters R7
    ST    R1, CCR1    ; save original string ptr
    AND   R4, R4, #0   ; initialize count to zero
    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          ; if not zero, loop
    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
    
```

7-16

FirstChar Algorithm



7-17

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   ; initialize ptr to beginning of string
    ADD   R2, R1, #0   ; read character
    FC1   LDR   R3, R2, #0
    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          ; if not zero, loop
    FC2   LD    R3, FCR3 ; restore registers
    LD    R4, FCR4
    RET                                ; and return
    
```

7-18

Summary

Subroutines – why and how

JSR/JSRR/RET

Passing arguments and return values

Saving and restoring registers