COMP1003 Computer Organization

Lecture 11 Calling Sub-routines

United International College

Subroutine

- 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
- Why do we need a subroutine?
  - reuse
  - divide task among multiple programmers
  - use vendor-supplied library of useful routines

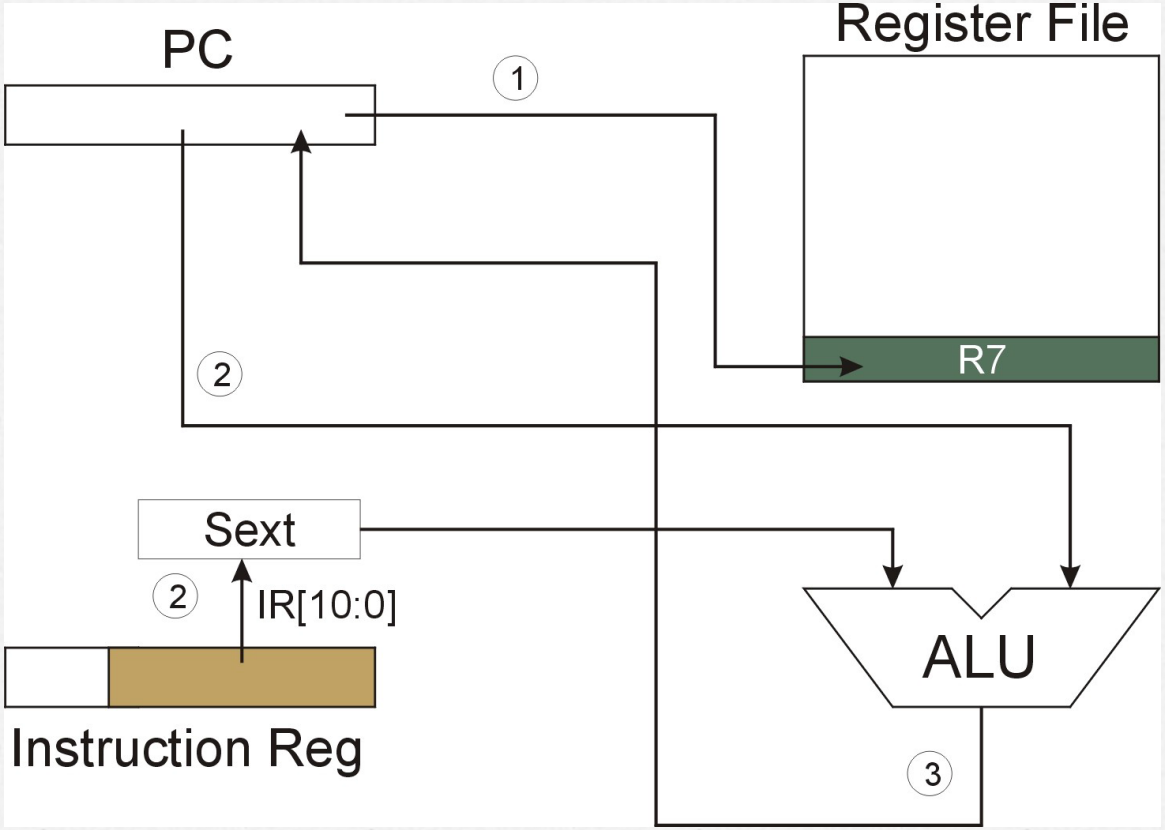
The Call/Return Mechanism

- The call mechanism
  - computes the starting address of the subroutine and saves it to PC
  - Save the return address of the calling program
  - JSR/JSRR instruction
- The return mechanism
  - Loads the PC with the return address
  - RET instruction

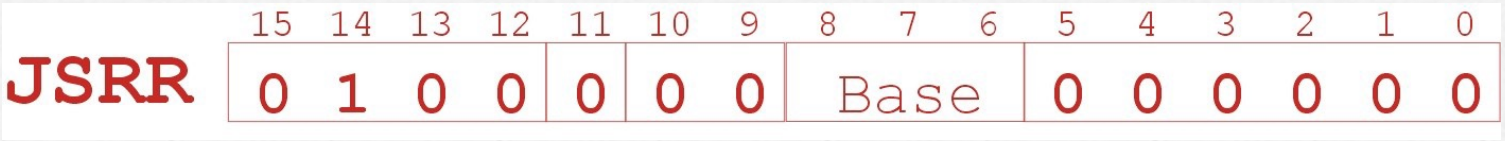
JSR Instruction

- Jumps to a location and save the current PC in R7
  - saving the return address (“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 Data Path



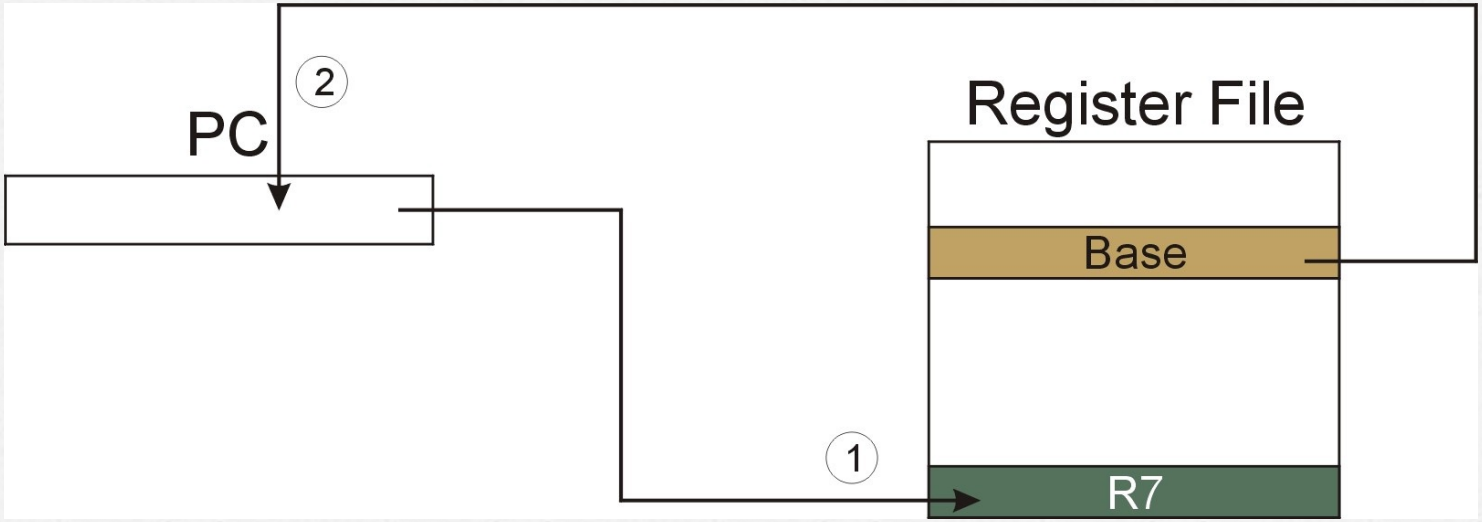
JSRR Instruction



- 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 Data Path



RET (JMP R7)

1100 000 111 000000

- Remember that the old PC was saved in R7 when the subroutine was called by JSR/JSRR
- 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 subroutine does not change R7, or we won't know where to return.

Example: 2sComp

```
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!

Passing Information to 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

Returning Information from Subroutines

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.

Using Subroutine

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

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).

Example

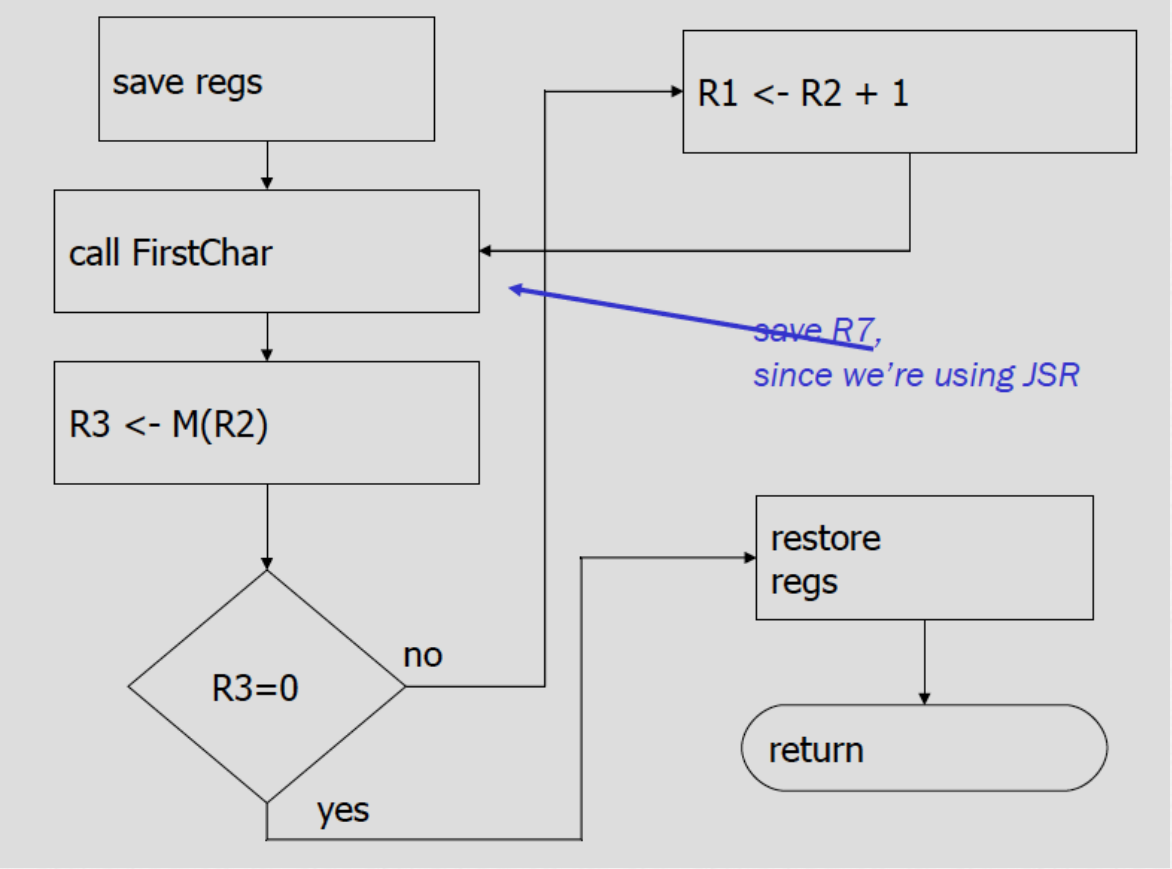
(1)Write a subroutine FirstChar to:

- 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.

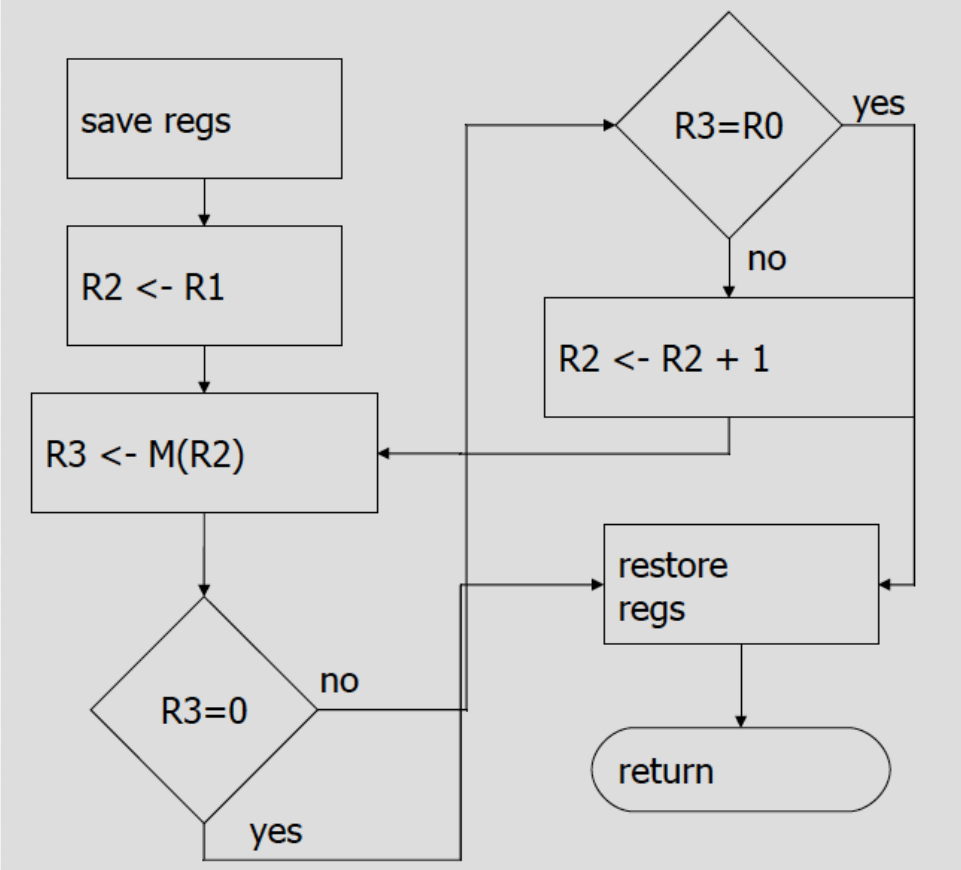
CountChar Algorithm (using FirstChar)



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
```

Exercises

- Write a program to print out “hello” for 10 times using loop.
- Write a subrotine to print out “Hello, World”
- Write a program myPUTS to replace PUTS to print out any string using loop. Assuming the string has been stored in the memory with a null terminator and the address of its first character is stored in R0.
- Write a program to input an integer N and print out “Hello world” for N times.

What's Next

- Input and Output: How to connect to the outside world?