Local Variables

Goal

- Keep values in register (simple and efficient)

More variables than register?

- Keep values in memory (load from memory to compute on them)

Example

```
        .ORIG x3800
Foo     . . .
        LD      R3, Val1
        ADD     R3, R3, #1
        ST      R3, Val1
        . . .
Val1    .FILL   #0
        . . .
        .END
```

What prevents another subroutine from accessing your local variables?

Global Variables

Just like local variables (labeled memory)

Problem

- LD only supports 9-bit offsets (-256 to 255)

Solution

- Keep *references* near subroutine and use indirect addressing

Example

```
        .ORIG x3800
Foo     . . .
        LDI     R3, Val1Ref
        ADD     R3, R3, #1
        STI     R3, Val1Ref
        . . .
Val1Ref .FILL   Val1
        . . .
Val1    .FILL   #0
        . . .
        .END
```

Note: Can be more than one reference to single datum

Note: All labels must be unique!

Example

(1) Write a subroutine FirstChar to. . .

Find first occurrence of particular character (in R0) in a string (pointed to by R1); return pointer to character or to end of string (NULL) in R5

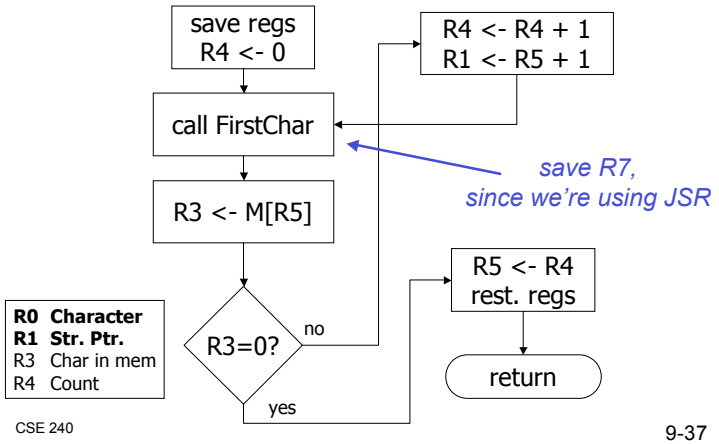
(2) Use FirstChar to write CountChar, which. . .

Counts number of occurrences of particular character (in R0) in a string (pointed to by R1); return count in R5

Strategy

• Write second subroutine first, without knowing the implementation of FirstChar!

CountChar Algorithm (using FirstChar)

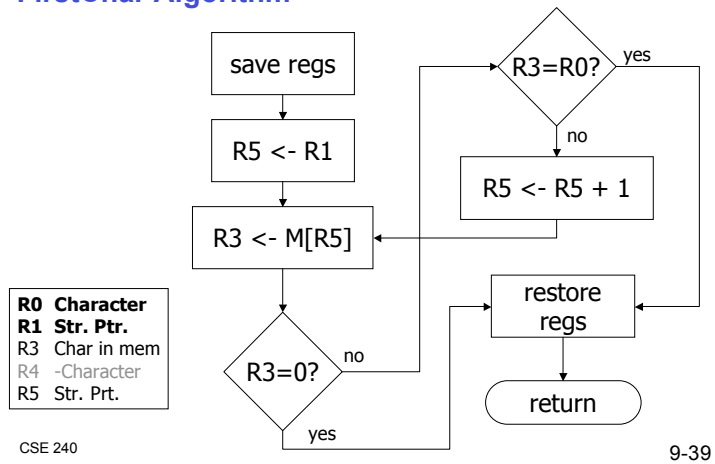


CountChar Implementation

R0	**Character**
R1	**Str. Ptr.**
R3	Char in mem
R4	Count

```
; 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 R1)
        LDR    R3, R5, #0    ; see if char or null
        BRz    CC2           ; if null, no more chars
        ADD    R4, R4, #1    ; increment count
        ADD    R1, R5, #1    ; point to next char in string
        BRnzp  CC1
CC2     ADD    R5, R4, #0    ; move return val (count) to R5
        LD     R3, CCR3      ; restore regs
        LD     R4, CCR4
        LD     R1, CCR1
        LD     R7, CCR7
        RET                  ; and return
```


FirstChar Algorithm



FirstChar Implementation

R0	**Character**
R1	**Str. Ptr.**
R3	Char in mem
R4	-Character
R5	Str. Prt.

```
; 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    R5, R1, #0    ; initialize ptr to beginning of string
FC1     LDR    R3, R5, #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    R5, R5, #1    ; increment pointer
        BRnzp  FC1
FC2     LD     R3, FCR3      ; restore registers
        LD     R4, FCR4      ;
        RET                  ; and return
```

What if we used CCR3?

Library Routines

Vendor provide object files containing useful subroutines

- **Don't want to provide source code (intellectual property)**
- **Assembler/linker must support EXTERNAL symbols**

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

Using JSRR, because SQRT likely not "nearby"