; Program to count the occurrences of a character in a string.
; Program inputs the character to count from keyboard.
; The string is provided starting from memory
; location x4000 (NULL terminated).
; e.g. if the string is "BRYN MAWR"
; and the input char is 'R'
; the output displayed will be 2
; Assume the count will always be less than 10 (why??)

 Algorithm

 count = 0
 nextCH = first character in string
 c = input character

 while nextCH != NULL do
   if nextCH = c then
     count = count + 1
   end if
   nextCH = next character in string
 output count

 Register Allocation

 count : R2
 c : R0
 nextCH : R1
 Address of string: R3 (x4000)

 LC-3 Algorithm

 R2 = 0
 R1 = M[R3+0]
 R0 = input from keyboard

 while R1 != 0 do
   if R1 = c then : R1 = R1 - R0
   end if
   R2 = R2 + 1
   R3 = R3 + 1 ; next char address
   R1 = M[R3 + 0]
 output R2
<table>
<thead>
<tr>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
<th>Character</th>
<th>Dec</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>01</td>
<td>A</td>
<td>10</td>
<td>41</td>
<td>l</td>
<td>76</td>
<td>4c</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>02</td>
<td>B</td>
<td>11</td>
<td>42</td>
<td>m</td>
<td>77</td>
<td>4d</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>03</td>
<td>C</td>
<td>12</td>
<td>43</td>
<td>n</td>
<td>78</td>
<td>4e</td>
</tr>
<tr>
<td>d</td>
<td>4</td>
<td>04</td>
<td>D</td>
<td>13</td>
<td>44</td>
<td>o</td>
<td>79</td>
<td>4f</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>05</td>
<td>E</td>
<td>14</td>
<td>45</td>
<td>p</td>
<td>80</td>
<td>50</td>
</tr>
<tr>
<td>f</td>
<td>6</td>
<td>06</td>
<td>F</td>
<td>15</td>
<td>46</td>
<td>q</td>
<td>81</td>
<td>51</td>
</tr>
<tr>
<td>g</td>
<td>7</td>
<td>07</td>
<td>G</td>
<td>16</td>
<td>47</td>
<td>r</td>
<td>82</td>
<td>52</td>
</tr>
<tr>
<td>h</td>
<td>8</td>
<td>08</td>
<td>H</td>
<td>17</td>
<td>48</td>
<td>s</td>
<td>83</td>
<td>53</td>
</tr>
<tr>
<td>i</td>
<td>9</td>
<td>09</td>
<td>I</td>
<td>18</td>
<td>49</td>
<td>t</td>
<td>84</td>
<td>54</td>
</tr>
<tr>
<td>j</td>
<td>10</td>
<td>0a</td>
<td>J</td>
<td>19</td>
<td>4a</td>
<td>u</td>
<td>85</td>
<td>55</td>
</tr>
<tr>
<td>k</td>
<td>11</td>
<td>0b</td>
<td>K</td>
<td>20</td>
<td>4b</td>
<td>v</td>
<td>86</td>
<td>56</td>
</tr>
<tr>
<td>l</td>
<td>12</td>
<td>0c</td>
<td>L</td>
<td>21</td>
<td>4c</td>
<td>w</td>
<td>87</td>
<td>57</td>
</tr>
<tr>
<td>m</td>
<td>13</td>
<td>0d</td>
<td>M</td>
<td>22</td>
<td>4d</td>
<td>x</td>
<td>88</td>
<td>58</td>
</tr>
<tr>
<td>n</td>
<td>14</td>
<td>0e</td>
<td>N</td>
<td>23</td>
<td>4e</td>
<td>y</td>
<td>89</td>
<td>59</td>
</tr>
<tr>
<td>o</td>
<td>15</td>
<td>0f</td>
<td>O</td>
<td>24</td>
<td>4f</td>
<td>z</td>
<td>90</td>
<td>5a</td>
</tr>
</tbody>
</table>

**LF = Line Feed**

**CRLF = \n\r**

*Campaign return*
Input is from the keyboard
Output is to the Console (Display Monitor)

All I/O is done using service routines (using TRAP instructions)

1. TRAP x23 ; input a char from keyboard into R0
2. TRAP x21 ; output char code in R0 to Console
3. TRAP x24 ; Output a string (null terminated) to Console
   ; Address of first char in string should be in R0
4. TRAP x25 ; HALT

LC-3 Assembly defines special instructions for these as well:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRAP x23 IN</td>
<td></td>
</tr>
<tr>
<td>TRAP x21 OUT</td>
<td></td>
</tr>
<tr>
<td>TRAP x24 PUTS</td>
<td></td>
</tr>
<tr>
<td>TRAP x25 HALT</td>
<td></td>
</tr>
</tbody>
</table>

e.g.

```
.ORIG x3000
START
   TRAP x23
   ADD R0, R0, #1
   TRAP x21
   OUT
   HALT
.END
```

Also:

```
.ORIG x3000
START
   IN
   ADD R0, R0, #1
   OUT
   LEA R0, MESG
   PUTS
   HALT

MESG .STRINGZ "Hello, world!"
.END
```
18 Subroutines

Subroutines
1. Call it
2. Return from it (to the just after the call)
3. May take 0 or more parameters
4. May return 0 or more resulting values

Θ. How is a subroutine defined?
In LC-3: it is a label

1. Calling a subroutine:
   JSR \( F \)
   JSRR Register
   \[ \text{PC} \rightarrow \text{F} \quad \text{F} \leq \text{PC} \]

2. Returning from a subroutine:
   JMP \( R7 \mid \text{PC} \leftarrow R7 \)

3. Parameters are passed in registers as well as return values
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Action</th>
<th>Addressing Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R2, R2, R3</td>
<td>R2 = R2 + R3</td>
<td>Register</td>
</tr>
<tr>
<td>ADD R2, R2, #1</td>
<td>R2 = R2 + 1</td>
<td>Immediate</td>
</tr>
<tr>
<td>AND R2, R2, R3</td>
<td>R2 = R2 AND R3</td>
<td>Register</td>
</tr>
<tr>
<td>AND R2, R2, #0</td>
<td>R2 = R2 AND 0</td>
<td>Immediate</td>
</tr>
<tr>
<td>BR[n][z][p] LABEL</td>
<td>If [n][z][p] Go to LABEL</td>
<td>CC, PC-relative</td>
</tr>
<tr>
<td>JMP R1</td>
<td>PC = R1</td>
<td>Register</td>
</tr>
<tr>
<td>JSR LABEL</td>
<td>R7 = PC, PC = SEXT(LABEL)</td>
<td>PC-relative</td>
</tr>
<tr>
<td>JSRR R6</td>
<td>R7 = PC, PC = R6</td>
<td>Base Register</td>
</tr>
<tr>
<td>LD R2, LABEL</td>
<td>R2 = m[ Label ]</td>
<td>Register, PC-relative</td>
</tr>
<tr>
<td>LDI R2, LABEL</td>
<td>R2 = m[ m[ Label ] ]</td>
<td>Register, Indirect</td>
</tr>
<tr>
<td>LDR R2, R0, #n6</td>
<td>R2 = m[ R0 + n ]</td>
<td>Base Register</td>
</tr>
<tr>
<td>LEA R2, LABEL</td>
<td>R2 = LABEL</td>
<td>Register, PC-relative</td>
</tr>
<tr>
<td>NOT R2, R1</td>
<td>R2 = NOT(R1)</td>
<td>Register</td>
</tr>
<tr>
<td>RET</td>
<td>Equiv to: JMP R7</td>
<td>Base Register R7</td>
</tr>
<tr>
<td>RTI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST R2, LABEL</td>
<td>m[ Label ] = R2</td>
<td>Register, PC-relative</td>
</tr>
<tr>
<td>STI R2, LABEL</td>
<td>m[ m[ Label ] ] = R2</td>
<td>Register, Indirect</td>
</tr>
<tr>
<td>STR R2, R0, #n6</td>
<td>m[ R0 + n ] = R2</td>
<td>Register, Base Register</td>
</tr>
<tr>
<td>HALT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Assembly Cheat Sheet

Friday, April 23, 2021
9:30 AM

240 Lectures Page 5
Program to divide two numbers by repeated subtraction to get quotient and remainder

```
.ORIG       x6000
A       .FILL       22
B       .FILL       7
.END

.ORIG       x600A
START   LD      R2, A
        LD      R3, B
        JSR     DIVMOD
        HALT

DIVMOD   ADD     R1, R2, #0  ; R1 = R2 (A)
        NOT     R4, R3      ; R4 = NOT(R3) and Adding 1 to R4
        ADD     R4, R4, #1
        AND     R0, R0, #0  ; R0 = 0
        ; while R1 >= R3 : R5 = R1 - R3 : R5 = R1 + R4
        LOOP    ADD     R5, R1, R4
        BRn     DONE
        ; do
        ADD     R1, R1, R4   ; R1 = R1 - R3
        ADD     R0, R0, #1   ; R0 = R0 + 1
        BR      LOOP
DONE    RET
.END
```
How to save & restore registers??

1. Saving

    \[
    \text{DIVMOD} \quad \text{// save R4+R5}
    \]

    \[
    \begin{align*}
    ST & R4, \text{Save}R4 \\
    ST & R5, \text{Save}R5 \\
    \vdots & \quad \vdots \quad \vdots \\
    \end{align*}
    \]

    \[
    \text{Restore values from saved registers}
    \]

    \[
    \begin{align*}
    LD & R4, \text{Save}R4 \\
    LD & R5, \text{Save}R5 \\
    \text{RET} \\
    \text{Save}R4 & .BLKW 1 \\
    \text{Save}R5 & .BLKW 1 \\
    \end{align*}
    \]

Try \implies \text{Implement your own PUTFs}

\implies \text{Inputs two numbers}
\implies \text{Compute their div + mod.