10. Example Program, Debugging (Chapter 5)  

- Example programs
- LC3 Tools (Edit, Simulate)
Using Branch Instructions

Compute sum of 12 integers.
Numbers start at location x3100. Program starts at location x3000.

R1 ← x3100
R3 ← 0
R2 ← 12

R1: Points to location in memory where numbers are stored in sequence
R3: The sum
R2: Keeps track of integer to be added
R4: Loads current integer

R2 = 0?

NO

YES

R4 ← M[R1]
R3 ← R3 + R4
R1 ← R1 + 1
R2 ← R2 - 1
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>x3000</td>
<td>1110 001 011 1111111</td>
<td>$R1 \leftarrow x3000 [PC+0xFF]$</td>
</tr>
<tr>
<td>x3001</td>
<td>0101 011 011 1000000</td>
<td>$R3 \leftarrow 0$</td>
</tr>
<tr>
<td>x3002</td>
<td>0101 010 010 1000000</td>
<td>$R2 \leftarrow 0$ [WANT TO ADD 12 TO IT]</td>
</tr>
<tr>
<td>x3003</td>
<td>0001 010 010 1011000</td>
<td>$R2 \leftarrow 12$</td>
</tr>
<tr>
<td>x3004</td>
<td>0000 010 000 000101</td>
<td>IF Z, GO TO $x300A$ (PC+5)</td>
</tr>
<tr>
<td>x3005</td>
<td>0110 100 001 0000000</td>
<td>LOAD NEXT VALUE TO R4</td>
</tr>
<tr>
<td>x3006</td>
<td>0001 011 011 0001000</td>
<td>$R3 \leftarrow R3+R4$</td>
</tr>
<tr>
<td>x3007</td>
<td>0001 001 001 100001</td>
<td>INCREMENT R1 (POINTER)</td>
</tr>
<tr>
<td>x3008</td>
<td>0001 010 010 1111111</td>
<td>DECREMENT R2 (COUNTER)</td>
</tr>
<tr>
<td>x3009</td>
<td>0000 111 111 111110</td>
<td>GO TO $x3004$ (PC-6)</td>
</tr>
<tr>
<td>x300A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Using "Sentinel"

Compute sum of 12 integers.
Numbers start at location x3100. Program starts at location x3000.
Sentinel stored in x310C is -1

R1 ← x3100
R3 ← 0
R4 ← M[R1]

R1 = Sentinel ??

YES

NO

R3 ← R3 + R4
INCREMENT R1
R4 ← M[R1]

R1: LOCATION IN MEMORY WHERE NUMBERS STORED
R3: SUM
R4: CURRENT INTEGER
Solving Problems using a Computer

Methodologies for creating computer programs that perform a desired function.

Problem Solving

- How do we figure out what to tell the computer to do?
- Convert problem statement into algorithm, using *stepwise refinement*.
- Convert algorithm into LC-3 machine instructions.

Debugging

- How do we figure out why it didn’t work?
- Examining registers and memory, setting breakpoints, etc.

*Time spent on the first can reduce time spent on the second!*
Stepwise Refinement
Also known as systematic decomposition.

Start with problem statement:
"We wish to count the number of occurrences of a character in a file. The character in question is to be input from the keyboard; the result is to be displayed on the monitor."

Decompose task into a few simpler subtasks.

Decompose each subtask into smaller subtasks, and these into even smaller subtasks, etc.... until you get to the machine instruction level.
Problem Statement

Because problem statements are written in English, they are sometimes ambiguous and/or incomplete.

- Where is "file" located? How big is it, or how do I know when I've reached the end?
- How should final count be printed? A decimal number?
- If the character is a letter, should I count both upper-case and lower-case occurrences?

How do you resolve these issues?

- Ask the person who wants the problem solved, or
- Make a decision and document it.
Three Basic Constructs

There are three basic ways to decompose a task:

- Iterative
- Conditional
- Sequential
Sequential

Do Subtask 1 to completion, then do Subtask 2 to completion, etc.

- Count and print the occurrences of a character in a file
- Get character input from keyboard
- Examine file and count the number of characters that match
- Print number to the screen
Conditional
If condition is true, do Subtask 1; else, do Subtask 2.

Test character. If match, increment counter.

file char = input?

Count = Count + 1
Do Subtask over and over, as long as the test condition is true.

Check each element of the file and count the characters that match.

Check next char and count if matches.

More chars to check?
Problem Solving Skills


- Like a puzzle, or a “word problem” from grammar school math.
  - What is the starting state of the system?
  - What is the desired ending state?
  - How do we move from one state to another?

- Recognize English words that correlate to three basic constructs:
  - “do A then do B” ⇒ sequential
  - “if G, then do H” ⇒ conditional
  - “for each X, do Y” ⇒ iterative
  - “do Z until W” ⇒ iterative
Code for Conditional

Test Condition

- True
  - Subtask 1
  - Next Subtask

- False
  - Subtask 2

Unconditional branch to Next Subtask

Exact bits depend on condition being tested

PC offset to address C

PC offset to address D

Instruction

Generate Condition

Assuming all addresses are close enough that PC-relative branch can be used.
Code for Iteration

Assuming all addresses are on the same page.
Detailed Example

Count the occurrences of a character in a file

- Program begins at location x3000
- Read character from keyboard
- Load each character from a “file”
  - File is a sequence of memory locations
  - Starting address of file is stored in the memory location immediately after the program
- If file character equals input character, increment counter
- End of file is indicated by a special ASCII value: EOT (x04)
  - Sentinel
- At the end, print the number of characters and halt
  (assume there will be less than 10 occurrences of the character)
TRAP

Calls a service routine, identified by 8-bit “trap vector.”

<table>
<thead>
<tr>
<th>vector</th>
<th>routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>x23</td>
<td>input a character from the keyboard</td>
</tr>
<tr>
<td>x21</td>
<td>output a character to the monitor</td>
</tr>
<tr>
<td>x25</td>
<td>halt the program</td>
</tr>
</tbody>
</table>

When routine is done, PC is set to the instruction following TRAP.
(We’ll talk about how this works later.)
Example: Counting Characters

Initial refinement: Big task into three sequential subtasks.

A
START
Initialize: Put initial values into all locations that will be needed to carry out this task.
- Input a character.
- Set up a pointer to the first location of the file that will be scanned.
- Get the first character from the file.
- Zero the register that holds the count.

B
Scan the file, location by location, incrementing the counter if the character matches.

C
Display the count on the monitor.

START
Input a file, counting occurrences of that character. Finally, display on the monitor the number of occurrences of the character (up to 9).

STOP
Scan the file, location by location, incrementing the counter if the character matches.

Test character. If a match, increment counter. Get next character.

Done?

Yes

No

Refining B

Refining B into iterative construct.
Refining B1 into sequential subtasks.

Refining B1

B

Done?

Yes

No

Test character. If matches, increment counter.

B2

Get next character.

B3
Refining B2 and B3

B1

Yes
Done?

No

B2
Test character. If matches, increment counter.

B3
Get next character.

B2

Yes
R1 = R0?

No
R2 = R2 + 1

B3

R3 = R3 + 1
R1 = M[R3]

Yes
Done?

No

Conditional (B2) and sequential (B3). Use of LC-2 registers and instructions.
Text: ASCII Characters

ASCII: Maps 128 characters to 7-bit code.
- both printable and non-printable (ESC, DEL, ...) characters

<table>
<thead>
<tr>
<th>ASCII Code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 nul</td>
<td>10 dle</td>
</tr>
<tr>
<td>01 soh</td>
<td>11 dc1</td>
</tr>
<tr>
<td>02 stx</td>
<td>12 dc2</td>
</tr>
<tr>
<td>03 etx</td>
<td>13 dc3</td>
</tr>
<tr>
<td>04 eot</td>
<td>14 dc4</td>
</tr>
<tr>
<td>05 enq</td>
<td>15 nak</td>
</tr>
<tr>
<td>06 ack</td>
<td>16 syn</td>
</tr>
<tr>
<td>07 bel</td>
<td>17 etb</td>
</tr>
<tr>
<td>08 bs</td>
<td>18 can</td>
</tr>
<tr>
<td>09 ht</td>
<td>19 em</td>
</tr>
<tr>
<td>0a nl</td>
<td>1a sub</td>
</tr>
<tr>
<td>0b vt</td>
<td>1b esc</td>
</tr>
<tr>
<td>0c np</td>
<td>1c fs</td>
</tr>
<tr>
<td>0d cr</td>
<td>1d gs</td>
</tr>
<tr>
<td>0e so</td>
<td>1e rs</td>
</tr>
<tr>
<td>0f si</td>
<td>1f us</td>
</tr>
</tbody>
</table>