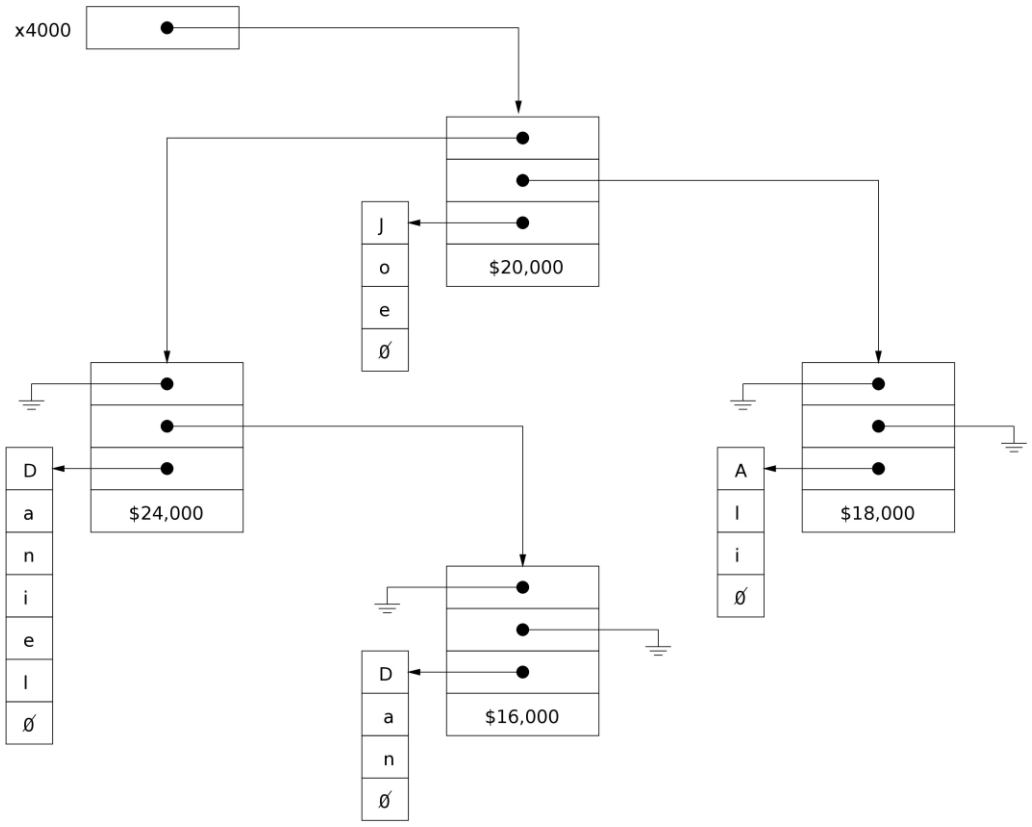
binsearch

Write a program in LC-3 assembly language that prompts the user to enter the name of a professor, searches a database for that professor, and prints to the screen that professor's salary. The database of professors will be stored in memory as a binary tree. We refer to the "top" node of a binary tree as the ROOT of the tree. The address of the root can be found at memory location x4000.

The Binary Tree

A binary tree is a special case of a tree where each node has at most two child nodes. We refer to the two children as the left child and the right child. In this lab, each node of the tree will consist of 4 words: a pointer to the left child if one exists, a pointer to the right child if one exists, a pointer to the ASCII string containing the professor's name, and a 2's complement number representing the professor's salary. If there is no left or right child, the corresponding word will contain the null pointer.

Note that a binary tree consists of a root and up to two children, each of which is in itself the root of a binary tree. In the example below, Joe is the root of the binary tree, its child Daniel is the root of a binary tree often referred to as the left subtree, and its child Ali is the root of a binary tree often referred to as the right subtree. Here is an example tree:



Input/Output Requirements
Described below are detailed requirements about the inputs and outputs of your program. All prompts and outputs should be printed to the screen EXACTLY as shown below to receive full credit.

Input: Your program should print the following prompt to the screen: `Type a professor's name and then press Enter:`

The user will input either a professor's name or the letter 'd', signifying that the user has no more professors to look up, followed by the Enter key. You must echo each character (including the enter key) to console.

Output: If there is a match, print the professor's salary to the console. If there is no match, print `No Entry` to the console. If the letter 'd' is typed (followed by the enter key), do not print anything to the console. To assist with printing salaries larger than a single ASCII character, we have provided you with a subroutine that takes as input a 2's complement number in R0 and prints it to the console. You may use this subroutine in any way you choose.

Example:

```
Type a professor's name and then press Enter:Dan
16000
Type a professor's name and then press Enter:Dani
No Entry
Type a professor's name and then press Enter:Daniel
24000
Type a professor's name and then press Enter:dan
No Entry
Type a professor's name and then press Enter:d

----- Halting the processor ----
```

Solution

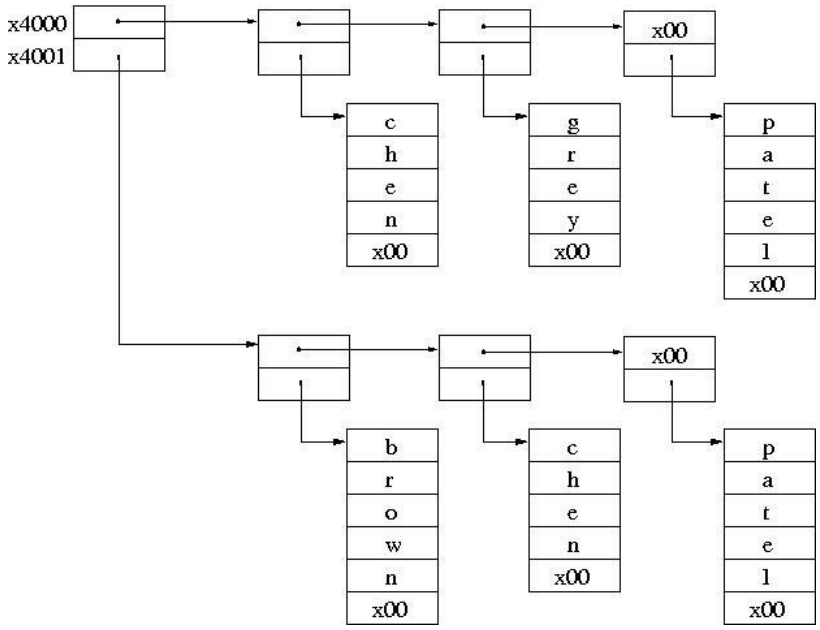
https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/binsearch.asm

intersection

A linked-list is a set of elements connected to each other via pointers. We often refer to these elements as "nodes." Each node in a linked-list is comprised of k+1 words. The first word contains a pointer to (i.e., the address of) the next node, and the remaining k words contain data associated with that node. The first word of the last node contains x0000, indicating that there are no more nodes in the list. We refer to x0000 as the null pointer.

In this assignment, each node in our linked lists will consists of two words. The first word will contain a pointer to the next node. The second word will contain a pointer to a character string representing the name of the person associated with that node. A character string consists of ASCII codes stored in consecutive memory locations, one ASCII code per location. The string is null-terminated, i.e., the end of a string is signified by the NULL character which is ASCII code x00.

Below is a graphical representation of two linked-lists. One consists of three nodes, representing three people: Chen, Grey, and Patel. The other consists of three nodes, representing Brown, Chen, and Patel. Note that each linked list terminates with the null pointer x0000. Note also that each character string terminates with a null entry x0000. Finally, note that each list has a list head (memory locations x4000 and x4001 in the example) which contain pointers to the first node in their respective linked lists.



Overview

A realtor is trying to rent out apartment units and is struggling to find the right tenants. They need to find tenants that earn at least $75,000/year. Additionally, they would prefer to rent to tenants that have at least three children. Fortunately, the realtor has a list of people, organized as a linked list, that earn over $75,000/year and another list of people, also organized as a

linked list, that have at least three children -- both lists are stored in memory. The realtor wants you to provide them with a list of all the people, also organized as a linked list, that earn over $75,000/year who have at least three children.

Your Job
Write a program in LC-3 assembly language that takes as input two linked lists stored in memory, the first one containing the names of all people who earn over $75,000/year, and the second one containing the names of all people who have at least three children, and produce as output a third linked list, also stored in memory, containing the names of people who earn over $75,000 and have at least three children. More specifically, the result of your program will be 3 linked lists: people who earn over $75,000/year who have fewer than three children, people that have at least three children and earn less than $75,000/year, and people who earn over $75,000/year and have at least three children. You will achieve this by removing from the first linked list those people who have at least three children, and from the second list those people who earn more than $75,000/year, and produce the third list from people who are removed from the first two lists.

In the example linked lists shown above, if the first list consisted of people earning more than $75,000 (Chen, Grey, and Patel), and the second list consisted of people with at least three children (Brown, Chen, and Patel), your result would be: The first list consists of the single node Grey, the second list consists of the single node Brown, and the third list consists of the two nodes Chen and Patel.

Finally, assume LC-3 is sufficiently constrained that you cannot waste space. That is, you are not allowed to create any additional nodes to help you solve this assignment, but instead you will construct your third list from nodes removed from your first list.

The address of the first node of the list (the list head) of people that earn over $75,000/year will be stored at location x4000. The address of the first node of people with at least three children will be stored at location x4001. You will store the address of the first node of the result at location x4002. A node will consist of two words: a pointer to the next node and a pointer to a character string. All nodes in each list will be ordered in alphabetical order, based on their names. Each of the three final lists must also be sorted in alphabetical order based on person's name. The string of characters will represent the name of the individual in question. For purposes of this lab, we will assume no two people have the same name. Thus, each name uniquely identifies one individual who either earns $75,000/year, has at least three children, or both. The string will be null-terminated and stored in memory like so: one ASCII character per memory location, the last character will be the null character.

Solution
Currently not provided

interrupt1

The user program
Your user program will consist of continually printing "Input a capital letter from the English alphabet:" on the display, ending each line with a newline character (ASCII code x000A).

Example output:

```
Input a capital letter from the English alphabet:
Input a capital letter from the English alphabet:
Input a capital letter from the English alphabet:
Input a capital letter from the English alphabet:
Input a capital letter from the English alphabet:
```

To ensure the output on the screen is not too fast to be seen by the naked eye, the user program should include a piece of code that will count down from 40000 each time a line is output on the screen. A simple way to do this is with the following subroutine DELAY:

```
DELAY   ST  R1, SaveR1
        LD  R1, COUNT
REP     ADD R1,R1,#-1
        BRnp REP
        LD  R1, SaveR1
        RET
COUNT   .FILL #40000
SaveR1  .BLKW 1
```

Feel free to change the number 40000 to speed up or slow down outputting the prompt.

The keyboard interrupt service routine
The keyboard interrupt service routine will examine the key typed to see if it is a capital letter in the English alphabet.

If the character typed is NOT a capital letter, the interrupt service routine will, **starting on a new line on the screen**, print "*<the input character>* is not a capital letter in the English alphabet.". For example, if the input key is '#', interrupt service routine will print

```
# is not a capital letter in the English alphabet.
```

If the character typed IS a capital letter, the interrupt service routine will, **starting on a new line on the screen**, print "The lower case character of *<the input character>* is: *<the lower case of the input>*". For example, if the input key is 'G', interrupt service routine will print

```
The lower case character of G is g.
```

The service routine would then print a line feed (x0A) to the screen, and finally terminate with an RTI.

Your interrupt service routine should start at the location x1500.

Hint: Don't forget to save and restore any registers that you use in the interrupt service routine.

The operating system enabling code
Unfortunately, we have not YET installed Windows or Linux on the LC-3, so we are going to have to ask you to do the following two enabling actions in your user program first, before your user program starts outputting the prompts. Normally, these would be done by the operating system before your user program starts executing.

1. Normally, the operating system establishes the interrupt vector table to contain the starting addresses of the corresponding interrupt service routines. You will have to do that for the keyboard interrupt. The starting address of the interrupt vector table is x0100 and the interrupt vector for the keyboard is x80. It is necessary for you to only provide the one entry in the interrupt vector table that is needed for this programming lab assignment.
2. Normally, the operating system would set the IE bit of the KBSR. You will have to do that as well.

You should write a TRAP routine to perform the two tasks listed above, since they require privileged access. The TRAP routine should start at location x2000 and it should be accessed by vector x30. The starting address of the trap vector table is x0000.

Solution

https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/interrupt1.asm

interrupt2

The user program
Your user program will consist of continually printing the following banner:

```
====================
*     *  *******
*     *      *
*     *      *
*     *      *
 *****       *

*****    ****  ****
*      *       *
****   *       ****
*      *       *
*****    ****  ****
====================
```

To ensure the output on the screen is not too fast to be seen by the naked eye, the user program should include a piece of code that will count down from 40000 each time the entire banner is output on the screen. A simple way to do this is with the following subroutine DELAY:

```
DELAY   ST  R1, SaveR1
        LD  R1, COUNT
REP     ADD R1,R1,#-1
        BRnp REP
        LD  R1, SaveR1
        RET
COUNT   .FILL #40000
SaveR1  .BLKW 1
```

Feel free to change the number 40000 to speed up or slow down outputting the prompt.

The keyboard interrupt service routine
The keyboard interrupt service routine, which starts at x1000, will examine the key typed to see if it it is a decimal digit.

If the character typed is NOT a decimal digit, the interrupt service routine will, **starting on a new line on the screen**, print "*<the input character>* is not a decimal digit." For example, if the input key is '#', the interrupt service routine will print

```
# is not a decimal digit.
```

The service routine would then print a line feed (x0A) to the screen, and finally terminate with an RTI.

If the character typed IS a decimal digit, the interrupt service routine will, **starting on a new line on the screen**, print out the number, *N*, N times. For example, if the input key is '4', the interrupt service routine will print

4444

If the input key is '5', the interrupt service routine will print

55555

The service routine would then print a line feed (x0A) to the screen, and finally terminate with an RTI.

The operating system enabling code
Unfortunately, we have not YET installed Windows or Linux on the LC-3, so we provide you with starter code that enables interrupts. **You must use the starter code for this assignment**. The locations to write the user program and interrupt service routine are marked with comments. The starter code does the following:

1. Initializes the interrupt vector table with the starting address of the interrupt service routine. The keyboard interrupt vector is x80 and the interrupt vector table begins at memory location x0100. The keyboard interrupt service routine begins at x1000. Therefore, we must initialize memory location x0180 with the value x1000.
2. Sets bit 14 of the KBSR to enable interrupts.
3. Pushes a PSR and PC to the system stack so that it can jump to the user program at x3000 using an RTI instruction.

Solution

https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/interrupt2.asm

nim

Nim is a simple two-player game that probably originated in China (although the name is thought be German, from *nimm* the German word for "take"). There are many variations of this game with respect to the type of counters used (stones, matches, apples, etc.), the number of counters in each row, and the number of rows in the game board.

Rules
In our variation of Nim, the game board consists of three rows of rocks. Row A contains 3 rocks, Row B contains 5 rocks, and Row C contains 8 rocks.

The rules are as follows:

- Each player takes turns removing one or more rocks from a single row.
- A player cannot remove rocks from more than one row in a single turn
- The game ends when a player removes the last rock from the game board. The player who removes the last rock loses.

What to do
At the beginning of the game you should display the initial state of the game board. Before each row of rocks you should output the name of the row, for instance "Row A:". You should use the ASCII character lowercase "o" (ASCII code x006F) to represent a rock. The initial state of the game board should look as follows:

```
ROW A: ooo
ROW B: ooooo
ROW C: oooooooo
```

Player 1 always goes first, and play alternates between Player 1 and Player 2. At the beginning of each turn you should output which player's turn it is, and prompt the player for her move. For Player 1 this should look as follows:

```
Player 1, choose a row and number of rocks:
```

To specify which row and how many rocks to remove, the player should input a letter followed by a number (they do NOT need to press Enter after inputting a move). The letter (A, B, or C) specifies the row, and the number (from 1 to the number of rocks in the chosen row) specifies how many rocks to remove. Your program must make sure the player's move has a valid row and number of rocks. If the player's move is invalid, you should output an error message and prompt the same player for a move. For example, if it is Player 1's turn:

```
Player 1, choose a row and number of rocks: D4
Invalid move. Try again.
Player 1, choose a row and number of rocks: A9
Invalid move. Try again.
```

```
Player 1, choose a row and number of rocks: A*
Invalid move. Try again.
Player 1, choose a row and number of rocks: &4
Invalid move. Try again.

Player 1, choose a row and number of rocks:
```

Your program should keep prompting the player until a valid move is chosen. Be sure your program echoes the player's move to the screen as they type it. After you have echoed the player's move, you should output a newline character (ASCII code x000A) to move the cursor to the next line.

After a player has chosen a valid move, you should check for a winner. If there is one, display the appropriate banner declaring the winner. If there is no winner, your program should update the state of the game board to reflect the move, re-display the updated game board, and continue with the next player's turn.

When a player has removed the last rock from the game board, the game is over. At this point, your program should display the winner and then halt. For example, if Player 2 removes the last rock, your program should output the following:

```
Player 1 Wins.
```

Solution

Currently not provided

polyroot

Given an interval (xlow, xhigh), a function f(x) that is monotonic in that interval, and either f(xlow) is positive and f(xhigh) is negative or f(xlow) is negative and f(xhigh) is positive, write an LC-3 assembly language program that uses binary search to find the zero of f(x) in the interval (xlow, xhigh), and store that value of x in x4000.

Input to your program will be found in a table in memory, starting at x4001:

Address	Content
x4001	xlow
x4002	xhigh
x4003	degree of the polynomial, say n
x4004 to x4004 + n	polynomial coefficients

For example, if f(x) is the polynomial Ax^3 + Bx^2 + Cx + D, the table will take the form:

Address	Content
x4000	output: x-position of zero
x4001	left bound
x4002	right bound
x4003	degree: 3
x4004	A
x4005	B
x4006	C
x4007	D

Solution

https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/polyroot.asm

pow2

In this assignment, you are asked to write a program in LC-3 machine language that checks a positive 2's complement number to determine if it is a power of 2. The number is stored in memory location x3050. If the number is a power of 2, your program should store x0001 in memory location x3051. If the number is not a power of 2, your program should store x0000 in memory location x3051. Your program should start at memory location x3000, and should halt the machine after storing the output.

Solution

https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/pow2.bin

rotate

In this assignment, you are asked to write a program in LC-3 machine language to rotate a bit pattern some number of bits to the left and store the result in memory. The *rotate amount* (number of bits you rotate the bit pattern to the left) is a non-negative integer between 0 and 16, inclusive. Your program should assume that the initial bit pattern is in memory location **x3100** and the rotate amount is stored in memory location **x3101**. Using those values, your program should perform the left rotation and store the result in memory location **x3102**. Your program should start at memory location **x3000**.

Solution

https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/rotate.bin

shift

In this assignment, you are asked to write a program in LC-3 machine language to shift a bit pattern some number of bits to the left and store the result in memory. The number of bits the bit pattern should be shifted is called the *shift amount*. Shift amount is a non-negative number between 0 and 16, inclusive (that is 0 and 16 are valid shift amounts). Your program should assume that the initial bit pattern to be shifted is in memory location **x3100** and the shift amount is stored in memory location **x3101**. Using those values, your program should perform the left shift and store the result in memory location **x3102**. Your program should start at memory location **x3000**.

Solution

https://github.com/chiragsakhuja/lc3tools/blob/master/src/test/tests/samples/solutions/shift.bin

sort

In this programming assignment, you are being asked to "sort an array."

Sorting is the process of arranging a set of elements in some order: numerical order, alphabetical order, height (shortest in front) order, etc.

An array is a collection of elements all having the same data type, and each element identified by the value of its index. (We will explain index momentarily). An array can be of any dimension. Most common are two-dimentional arrays and one-dimensional arrays. A two-dimensional array is organized as a set of rows and columns, and requires two index values to specify each element. For example, the first element of the array A would be A[0,0], i.e., top row, left-hand column. A[2,4] identifies the element that occupies row 2 (the third row), and column 4 (the fifth column). A one-dimensional array only requires one index value to specify each element. B[0] specifies the first element, B[1] specifies the second, etc.

Arrays are stored in sequential locations of memory. The address of the location of the first element of the array, for example A[0,0] or B[0], is referred to as the base address of the array. In order to store elements of a two-dimensional array into a one-dimensional memory requires a little thought beyond what we will do in this assignment. Something for another day. In this assignment we are working with a one-dimensional array, where the location of each element is determined pretty easily. B[0] is stored at the base address, B[1] is stored at the next address, and so on.

Your program will use an array NUMBERS, where NUMBERS[0] will be stored at the base address, in memory location x32F0. NUMBERS[1] will be stored in M[x32F1], NUMBERS[2] in M[x32F2]. etc. Since there are 16 elements in the array, and since each element occupies exactly one word of LC-3 memory, the last element NUMBERS[15] will be stored in M[x32FF].

Write a program in LC-3 assembly language to sort an array of 16 2's complement integers. Your program should assume that the first element of the array is stored in memory location x32F0 and the last element of the array is stored in memory location x32FF. Each memory location contains a single 2's complement integer. That is, each memory location contains a single element of the array. Your program should sort the array of 2's complement integers in ascending order and store the result back in memory locations x32F0 through x32FF. Your program should start at memory location x3000.

Solution

Currently not provided