# Little Man Computer

A computer simulation for the World Book TutorVision

Written by Michael Hayes

http://www.midnightblueinternational.com

Date of last modification: July 19, 2021

Cartridge Instructions

For color TV viewing only

## Notes

Words that are underlined have entries in the Glossary, starting on Page 25. Use the Glossary to look up the meanings of words you might not fully understand.

## Introduction

In 1965, Dr. Stuart Madnick created a theoretical computer model that draws on the analogy of a little man in a mail room. It is commonly used as a teaching tool for first-year Computer Science students, because it has all the essential features for computing. Within the mail room are 100 mailboxes, a Program Counter and Accumulator, and an Inbox and Outbox. For simplicity, the computer uses Decimal numbers as opposed to Binary numbers, and there is a limited instruction set.

This is a simulation of the computer model with a namesake Little Man, with you as his boss. He comes into the mail room at the start of each session and warms up with coffee while he waits to receive orders from you in the form of a program. At the push of a button, off he goes to execute your program until it is finished.

You can create programs either by manually entering numbers into the mailboxes, or you can write code and assemble it, which converts your code into the appropriate numbers for the mailboxes. In addition, you can view the Output tape where data was written to the Outbox during past program execution, and proactively write numbers onto the Input tape, which is fed through the Inbox for future execution.

What you are reading right now is more than an instruction manual; it is intended as an interactive tutorial as well. This is a departure from my usual documentation style because I am accustomed to writing manuals for games and computer software. As this software is meant to be an educational title rather than a game, going through this manual is the beginning of the “journey”. To fully enjoy Little Man Computer, you will want to understand all its features, which we will do through a few programming examples along the way. Once you get through all the examples, you are free to try writing programs of your own.

I urge you to put the Glossary to good use as you go along. Not understanding the terminology is a surefire way to become overwhelmed and then lose interest.

# Table of Contents

## Check Your Equipment

### Make sure:

- The World Book TutorVision is connected to your TV set and the power cord is plugged in.
- The Antenna Switch Box is set at GAME.
- The TV set is plugged in and properly adjusted.
- The Little Man Computer cartridge is placed in the TutorVision slot, firmly engaged.

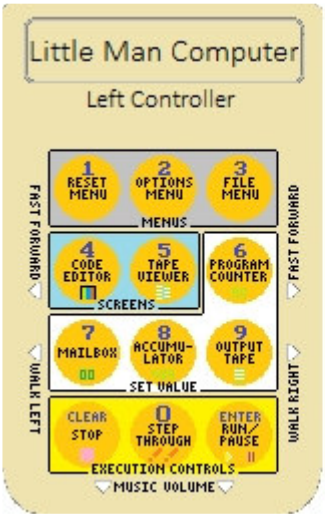
## Press Reset Button:

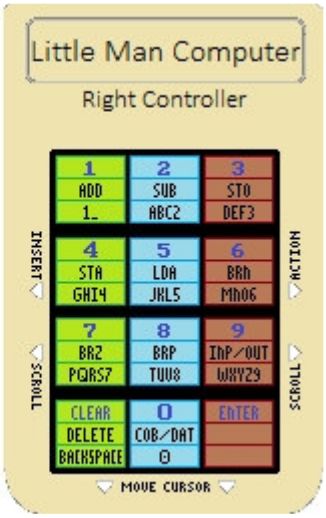
The title screen will appear. It should look like this: or this:



## Add Overlays

- Find the Little Man Computer keypad overlays in the cartridge package with this booklet.
- Take both hand controllers.
- Insert each overlay into the appropriate hand controller. Make sure overlays fit tight and are all the way in. The overlays will be your visual guide throughout the program.



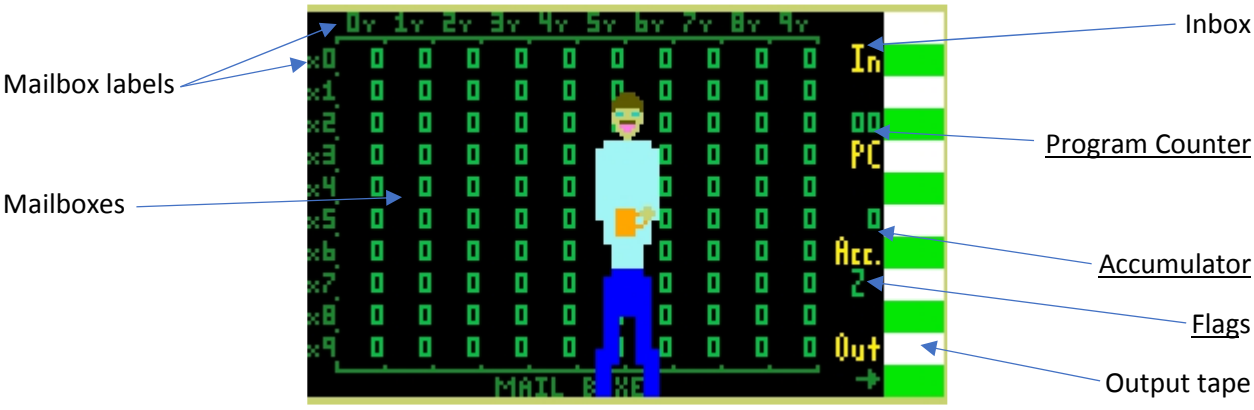




## Examine Your Controls

- The first time you play, you will see a Welcome screen. You will continue to see this screen after the Title screen until you save a file. From this screen, press any key to open the Simulation screen.
- The Simulation screen is the “main” screen of this program. This is where you will see the Little Man in the mail room, waiting to receive orders from you. Use the Left controller in this screen.
- Pressing 1, 2, or 3 on the Left controller from the Simulation screen will take you to one of the Menu screens. Use either controller to navigate through the menus.
- Pressing 4 on the Left controller from the Simulation screen will take you to the Code Editor. Pressing 5 on the Left controller from the Simulation screen will take you to the Tape Viewer. Use the Right controller in these screens, except to press 4 or 5 on the Left controller to exit.

## Activate Screen

After pressing any key on the Welcome screen, you will go to the Simulation screen. The Little Man runs to the center of the Mail room and enjoys his morning coffee while he waits for your orders.




- Each of the 100 mailboxes has an initial value of 0 and can hold up to 3 digits.
- Because there are 100 mailboxes, there is a 2-digit address for each one, as indicated by the labels.
- The Inbox and Outbox are initially empty. If the Inbox is non-empty, the next value to be read will be displayed above the word "In". The last 12 numbers written to the Output tape are shown on the right side of this screen.
- The Program Counter, labeled "PC", holds only 2 digits and indicates which mailbox to receive the next order from while the Little Man is executing a program.
- Below that is the Accumulator, which can hold 3 digits and can have simple addition and subtraction done to its current value.
- In addition to the Accumulator's current value, there are 3 flags that show a letter when they are "raised". In the illustration above, 'Z' is raised to indicate the Accumulator has a value of Zero.

## Adjust Settings

Pressing 2 on the Left controller will take you from the Simulation screen to the Options menu.


From there, you can access the Sound Test or change a few settings.

These are the settings you can change:

1. Mailbox Order. The default setting is "xy" which means the first ten mailboxes are listed along the left column (x=0) from the top down, and then each set of ten mailboxes is listed in the next column to the right. This is the default setting because it is easier to read. If you change this setting, you will be taken back to the Simulation screen, and you will see the Little Man perform magic to change the layout of the mailboxes.
2. The name of instruction 0. By default, it is called "Coffee Break". When 0 is read as an instruction, it means the Little Man is done executing your program and can relax and enjoy some more coffee. The alternate name is "Halt".
3. As a means of teaching the difference between signed and unsigned variables, the Accumulator handles negative values. The Inbox and mailboxes have a range between 0 and 999 (unsigned), but the Accumulator has a range between -500 and 499 (signed, because the minus sign can be present). By default, it displays negative values, but you can change this setting to "show large values" instead. When the Accumulator is showing a negative number, the 'M' flag is raised to signal "Minus". If it is set to show large values, anything higher than 499 will still raise the 'M' flag.
4. Because 499 is the highest value the Accumulator can handle (even if it is showing large values, they are still treated as negative), when an Add or Subtract instruction is carried out that causes the Accumulator's value to go above 499 or below -500, it raises the 'O' flag to signal "Overflow". The original Little Man Computer design did not specify what to do in the event of an Overflow, and so there are three choices here:
    a. "Hold Min/Max Value" is the default choice. It means the Accumulator value will remain at 499 if an Add instruction would take the value higher than that, or at -500 if a Subtract instruction would take it lower than that.
    b. "Wrap Around" means the Accumulator value goes between the highest value and the lowest value, or vice versa. For example, if its value is 499 and you attempt to add 1 to it, its value becomes -500. Again, even if it is displaying large values, you will see "500" but the 'M' flag will still be raised. To understand what "wrap around" means, think of an analog odometer on older cars that has only so many digits. When all of them are set to '9', the next thing that happens is they all reset to '0', a phenomenon also known as "rollover".
    c. "Error on Acc Range" means the program will halt with an error message if Overflow occurs.
5. Another behavior not specified by the original Little Man Computer design is what to do when attempting to store a negative value into one of the mailboxes. By default, "No Neg Value Error" means the equivalent large value will be stored instead, and the program will continue running. The other choice is "Error on Neg Value" which will halt the program with an error.
6. A third unspecified behavior is what to do if an invalid Opcode is read. "Error for Opcodes" is not the default choice but is what would normally happen in such a case: halt the program with an error. "NOP" stands for "No Operation" which instructs the Little Man to do nothing for that instruction, and so "NOP Invalid Opcode" means to ignore it.
7. The fourth and final unspecified behavior involves the Inbox. In the early days of computing, all the input values needed to be ready to feed to the computer ahead of time, usually through a tape or punch cards. If it carried out an instruction to read a value from the Inbox (often known as an "Input stream" or "tape") and there was nothing there, it would halt with an error. The default choice is more user-friendly; have the Little Man "Wait for Input" instead. He will point to the Inbox and the music will stop, letting you know that you need to enter a number before he can continue working.

## Programming

Each mailbox value is an encoded instruction for the Little Man to do something specific. Altogether, the mailboxes comprise a program. The instructions contained within the mailboxes are typically carried out in sequence, but some of the instructions allow for deviation from linear execution. Such deviation is known as "branching" because it may or may not happen depending on the Accumulator Flags.

The "hundreds place" digit specifies the instruction to be carried out, and the other two digits qualify that instruction, usually by specifying which mailbox to go to. Although various implementations of Little Man Computer have different defined sets of instructions, here is what we will be using:

| Number | Instruction | Mnemonic | Description |
|---|---|---|---|
| 0 | Coffee Break (Or Halt) | COB (Or HLT) | Stop and take a Coffee Break. |
| 100-199 | Add | ADD | Fetch the value in the mailbox specified by the last two digits and add that value to the Accumulator. If the Accumulator's value goes over 499, the 'O' flag is raised. |
| 200-299 | Subtract | SUB | Fetch the value in the mailbox specified by the last two digits and subtract that value from the Accumulator. If the Accumulator's value goes below -500, the 'O' flag is raised. |
| 300-399 | Store | STO | Fetch the value from the Accumulator and store it in the mailbox specified by the last two digits, overwriting what might already be there. If the Accumulator's value is negative, either convert to a large value or halt with an error. |
| 400-499 | Store Address | STA | Fetch the last two digits from the Accumulator and store them in the mailbox specified by the last two digits of this instruction, overwriting with might already be there. |
| 500-599 | Load | LDA | Fetch the value in the mailbox specified by the last two digits and replace the value in the Accumulator. |
| 600-699 | Branch | BRN | Change the Program Counter to the value specified by the last two digits. Note this happens unconditionally. |
| 700-799 | Branch if Zero | BRZ | Check the Accumulator Flags. If the 'Z' flag is raised, then change the Program Counter to the value specified by the last two digits. Otherwise do nothing. |
| 800-899 | Branch if Positive | BRP | Check the Accumulator Flags. If the 'M' flag is *not* raised, then change the Program Counter to the value specified by the last two digits. Otherwise do nothing. |
| 901 | Input | INP | Fetch the next value from the Inbox and replace the value in the Accumulator. |
| 902 | Output | OUT | Fetch the value from the Accumulator and write it to the Outbox. |

To program Little Man Computer, one thing you can do is write specific numeric values to each mailbox. We will try that with the example on the next page. If it seems daunting, it is. After the following example, we will learn an easier way to do programming, by writing code that can then be assembled and translated into the appropriate numeric values for each of the mailboxes.

## Example 1: Subtraction

To set a value for a single mailbox, press 7 on the Left controller. The border will turn dark green, and the Little Man will run to the left and point to a spot in the top left corner where a spinning '?' has just appeared. Enter a 2-digit value, representing the number of the mailbox whose value you want to change. The number must be 2 digits, so for the first ten mailboxes, you will preface the number with a '0'. For example, the first mailbox is "00". Use the guides along the top and left edges to help you.

Once you've specified the mailbox number, the border turns green (a lighter shade) and the Little Man will step out of the way while you enter the value you want in that mailbox. This value can be anywhere from 1 to 3 digits. After you enter the value, the border will turn tan again and the Little Man will come back into view. Repeat these steps until you have all the values entered in the first eight mailboxes.

| Mailbox | Value | Command | Description |
|---|---|---|---|
| 00 | 901 | Inbox → Accumulator | Fetch the next value from the Inbox and store it in the Accumulator. Because the Inbox is empty, the Little Man will wait for you to enter a value instead. |
| 01 | 308 | Accumulator → Mailbox 08 | Store the Accumulator value into Mailbox 08. |
| 02 | 901 | Inbox → Accumulator | Fetch another value from the Inbox and store it in the Accumulator. This will replace the previous value. |
| 03 | 309 | Accumulator → Mailbox 09 | Store the Accumulator value into Mailbox 09. |
| 04 | 508 | Mailbox 08 → Accumulator | Restore the Accumulator value with the first Input value which was stored in Mailbox 08. |
| 05 | 209 | Subtract Mailbox 09 from Accumulator | Retrieve the second Input value, which was stored in Mailbox 09, and subtract it from the Accumulator's value. |
| 06 | 902 | Accumulator → Outbox | Fetch the Accumulator value and write it to the Output tape. |
| 07 | 0 | Coffee Break | Done. |

To execute the program, press Enter on the Left controller. The border turns yellow, the music changes, and the Little Man springs into action. The first thing he does is go to the Program Counter and reset its value to 00. During execution, you can use the Left controller to perform the following actions:

| Control | Action | Border Color | Description |
|---|---|---|---|
| Top Action Button | Fast Forward | N/A | Execution speeds up while you hold the button. |
| Clear | Stop | Pink | Execution will stop after the next step is done. |
| 0 | Step Through | Orange | Execution auto-pauses between instructions. |
| Enter | Pause | Purple | Execution will pause after the next step is done. |

While execution is paused, you can press Clear to stop, 0 to resume in Step-Through Mode, or Enter to resume normally. As described above, Step-Through Mode is used to pause execution after each instruction is fully carried out, to give you a chance to visually examine the state of the computer to ascertain that everything looks correct. Each instruction is carried out in 7 steps, which are described on the next page. For now, let the program just run, enter each of the two input values when requested, and observe that the difference between the two values is written onto the Output tape.

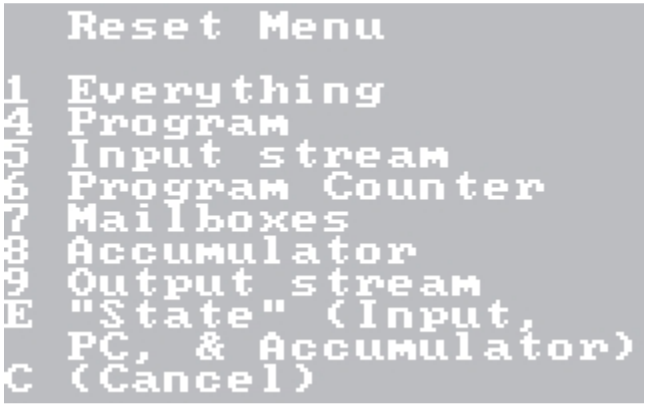
These are the steps of execution to perform each instruction:

1. Write the Program Counter's value onto an index card.
2. Go to the mailbox specified on the index card, and write the number stored there onto a second index card.
3. Return to the Program Counter and increment its value.
4. Read the number on the second index card.
5. If the new number is 0, then we are done. Otherwise go to wherever data needs to be fetched from, as specified by that number. It could be the Inbox, the Accumulator, or one of the mailboxes.
6. Carry out the instruction according to that number from the second index card and go to wherever the result needs to be stored. It could be the Outbox, the Accumulator, the Program Counter, or one of the mailboxes.
7. Go to the Program Counter and return to Step 1.

So, what are the shortcomings of writing programs in the above fashion? For one thing, you would have to always keep the Instructions chart from Page 6 handy, or else memorize it. Worse, you would have to meticulously keep track of which mailboxes are being used to store data, and which ones contain instructions to be carried out. What if you needed to insert one or more instructions earlier in the program somewhere along the way? You would have to rewrite some of the existing instructions to point to different mailboxes to store your temporary values. One solution would be to use the mailboxes at the end – starting at 99 and counting backwards – but that is not ideal. There must be an easier way to write programs than this.

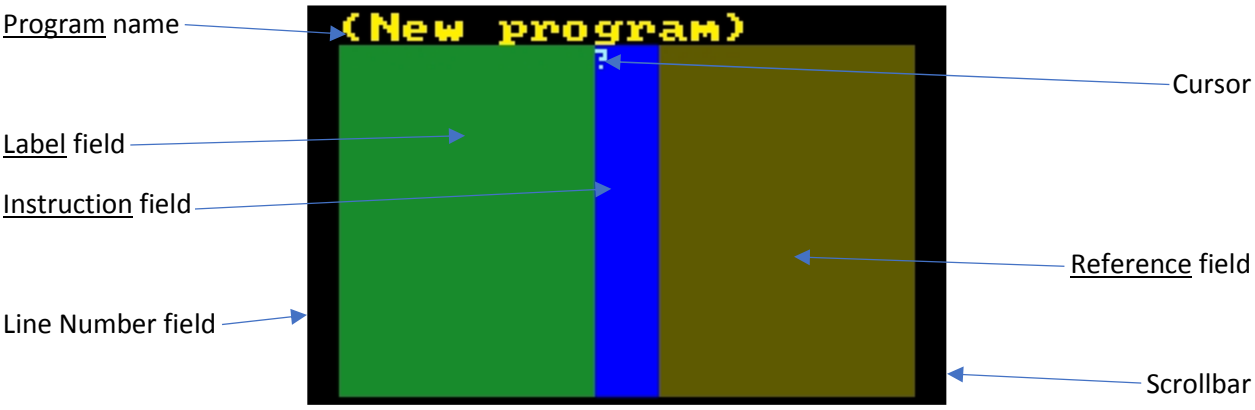
## Writing Code

First, wipe the slate clean. Press 1 on the Left controller to open the Reset Menu.




The Mailboxes, Program Counter, and Accumulator all have nonzero values, so we will want those reset to zero. In addition, there is something written onto the Output tape. We should just reset everything. Press 1 on the Left controller and watch as the Little Man does a small magic performance.

With that out of the way, press 4 on the Left Controller to open the Code Editor.




Each line of code has four fields as illustrated above: the Line Number (black), a possible Label (green), an Instruction (blue), and a possible Reference (brown). To the far right, arrows will appear when you have more code lines above and/or below the viewing area.

Within the Code Editor, you will use the Right controller. The Right controller's overlay will be vital as you refer to the various instructions while entering code.

## Example 1a: Subtraction (Using the Code Editor)

Here is the equivalent program as before, written in code. The line numbers are filled in automatically.

```
00          INP
01          STO   First
02          INP
03          STO   Second
04          LDA   First
05          SUB   Second
06          OUT
07          COB
08    First DAT   0
09    Second DAT  0
```

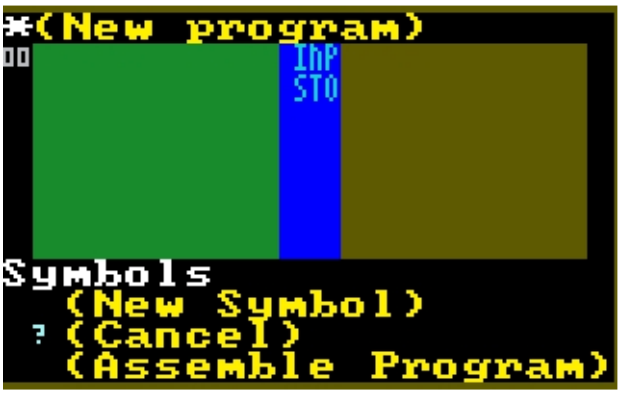
You should recognize the instructions by their mnemonics as listed on Page 6, except for "DAT" in the last two lines. DAT is not an instruction but is used to specify that "data goes here".

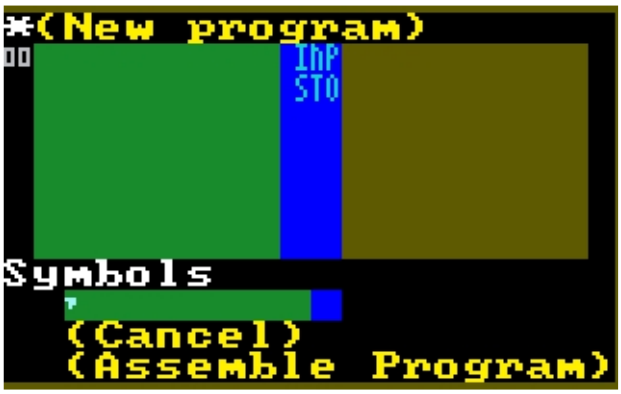
Right away, you should see an advantage to writing code. The line numbers are the mailboxes where the encoded value of each instruction goes. You can insert a line somewhere before the end of the program, and the line numbers for the DAT lines will be changed accordingly. "First" and "Second" are Symbols, which are the names we are giving to the first and second values retrieved from the Inbox. Code lines 01, 03, 04, and 05 use these Symbols as References. When we enter this program and then assemble it and return to the Simulation screen, the Little Man performs another magic trick to change all the mailbox values as specified by the code. There is no need for us to know that "First" is saved to Mailbox 08, and "Second" is saved to Mailbox 09. In more complex programs, that will take a weight off our shoulders to focus on what the program is supposed to be doing.

Now to enter the code. You will have to flip back and forth between Page 9 and the current page as you go along. Most instructions are entered by pressing the number on the Right controller's keypad that corresponds to its "hundreds place" value.

First, press 9. This creates the mnemonic "INP", generates the line number "00", and causes an asterisk to appear next to the program name, signaling that the code has been modified. If you were to press 9 a second time, the mnemonic would change to "OUT". That is why the cursor does not automatically move to the next line in this case. Move the cursor down manually by pressing the bottom of the disc.

For the second line, press 3 for "STO". A Symbols Pane will appear because this instruction requires a Reference. As it is, there are no Symbols created yet that we can use as a Reference. The border color changes from black to brown while the Symbols Pane is open.




The cursor appears in front of "(Cancel)" by default. If you were to select that, the "STO" mnemonic would disappear, because the code line cannot exist with an instruction that is missing the required Reference. Instead, press the top of the disc to move the cursor to "(New Symbol)" and press the Top action key, which is labeled "Action". The "(New Symbol)" text is replaced with a green field, with a blue box after it, as seen in the above illustration to the right.

Within this green field, you are going to enter the Symbol name, "First". Disregard the blue box for now. Entering text is done the same way as on early-model mobile phones. Look at the Right controller overlay. Within the bottom row of each key is a list of all the characters available. Lower-case letters are also available but are not listed on the overlay.

To enter the first character, 'F', press the 3 key three times rapidly. A second cursor appears in the new position, while the original cursor gradually moves toward it from its original position. This shows you how long you would have to wait before entering another character if it is bound to the same key on the keypad, roughly one second.

Next, press the 4 key six times rapidly to get the lower-case 'i'. If you are not in the mood to press keys more times than necessary and you do not care about mixed-case text, then just press the 4 key three times to get capital 'I'. After that, press the 7 key seven times for 'r' or three times for 'R'.

Because the next character is on the same key, here is where you must wait for the two cursors to align. Wait about one second, and then press the 7 key eight times for 's' or four times for 'S'. Finally, press the 8 key four times for 't' or once for 'T'. Press Enter to enter the name.

Now you are back at code entry. The Symbols Pane is hidden, and the border changes to black again.


Just as with the first line, press 9 for "INP". Again, with the 900-level instructions, you must press the bottom of the disc to move to the next line. For the fourth line, press 3 for "STO" once again. Same as before, when the Symbols Pane appears, select "(New Symbol)" and this time enter the word "Second".

For the fifth line, press 5 for "LDA". The Symbols Pane opens again. This time though, because the Symbol we want already exists, you will just select it from the list of existing Symbols. Move the cursor to the row containing the existing symbol and press the Action button. If you were to create a new Symbol and enter a name that already exists within the Symbols list, the Symbols Pane will just select the existing Symbol for you, to prevent duplicate names from populating the list.



Now we are on the sixth line, 05. Press 2 for "SUB". As before, you will select an existing Symbol. Move the cursor in front of "Second" and press the Action button. Notice an Up arrow appeared to the right. When the Symbols Pane appeared, it pushed all the code lines upward to keep the tentative new line in view. After the Symbols Pane closed, the code lines remained where they are. Now is a good time to try using the "Scroll" bottom action buttons to become familiar with them. After pushing the left Scroll button to bring the top line back into view, the arrow disappears. Move the cursor to the bottom again.

This brings us to code line 06, the seventh line. Press 9 twice for "OUT" and move the cursor down to the next line. Press 0 for "COB". If you had played with the Options earlier, you might see "HLT" instead. It makes no difference. For this instruction, you must manually go to the next line once again.

The last two lines contain Labels. I recommend you create the Labels first. Press the left side of the disc to move the cursor to the green Label field and press the Action button to open the Symbols Pane.


This time, the border will be dark green, matching the color of the field you were in before you opened the Symbols Pane. Move the cursor to "First" and press the Action button. Because there needs to be an instruction for the newly created code line, "COB" or "HLT" was entered automatically. Notice also the label is right justified so it is closer to the Instruction field.

Next, you will enter the rest of the line. Normally you would press 0 twice for "DAT". Since "COB" or "HLT" is already there, you only need to press 0 once. The cursor moves to the Reference field. Enter the number you want, and press Enter. If you press Clear to cancel entering a number, the instruction will revert to "COB" or "HLT". Other implementations of Little Man Computer typically make entering a number after "DAT" optional, with the caveat that the value defaults to 0. That is not the case here. When you finish entering the number, it is resized to conform to the other Reference text and is right justified among the three available positions, to make it easier to visually compare to other data values that might be more than one digit in length.

All the code is entered now. It should look like this:

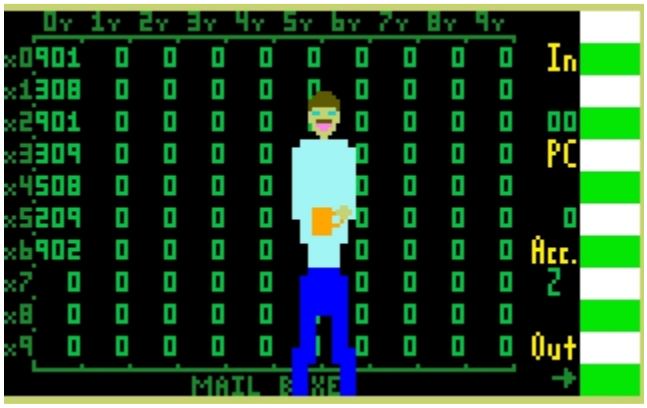


Every time the Symbols Pane was open, you saw the command at the bottom, "(Assemble Program)". As a shortcut, you can press the Action button while the cursor is in the blue Instruction field below the last existing line of code. Doing so will also invoke the Assembler.

When the Assembly Pane opens, the music stops, and the border turns blue. If there are no errors, you will get the message "Assembly successful." Press any keypad key to close the Assembly Pane. The asterisk is gone now, signaling that the code is freshly assembled.

Press 4 on the Left controller to close the Code Editor, returning you to the Simulation screen.

With the code being freshly assembled, you will be treated to a magic show, as all the mailboxes change their values. Compare these values to the ones listed on Page 7. They are the same.




To be sure, try executing the program. Remember that you are back to using the Left controller. You can try some of the execution controls if you are comfortable. Execution controls were described at the bottom of Page 7.

## Saving Your Work

Let us save our first program now. Press 3 (Left controller) to open the File Menu.

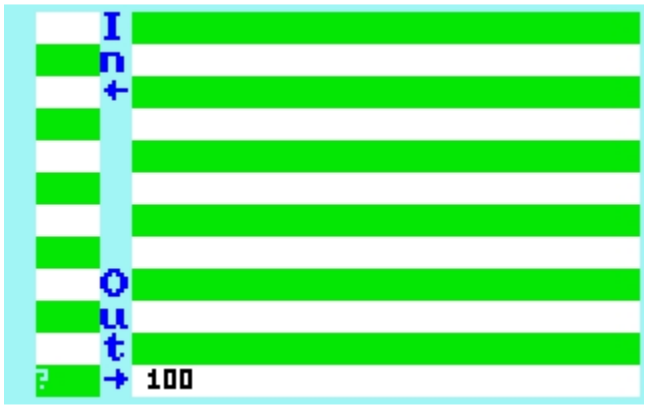

In the File Menu, press 1 to open the Save screen. The four examples in this instruction manual are already saved in the illustration above to the right. There are ten files that you can save to, numbered 0 to 9. Press the number of the file you want.

At this point, you will be queried to enter your project name. Use the same method as before when you entered Symbol names. Your project name can be up to 18 characters in length. Press Enter when you are finished. The text “Saving” will appear, followed by “Done” when finished. Press Clear to return to the File Menu and press Clear again to return to the Simulation screen.

Other options within the File Menu are to: rename your project, copy one file’s contents to another file, delete a file, and to load either some or all the components of a previously saved project.

## Tape Viewer

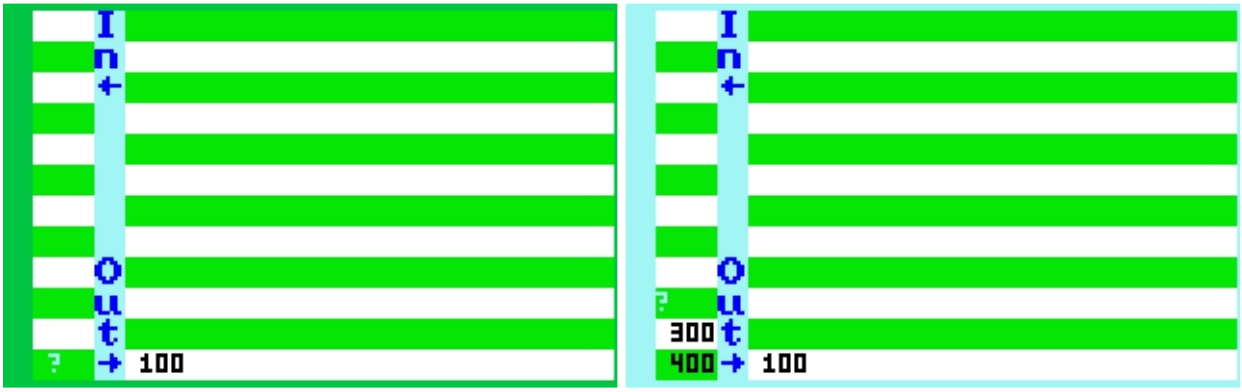
Press 5 on the Left Controller now to open the Tape Viewer.




In the above example, “100” was written to the tape because I used the values 400 and 300 when running the Subtraction program.

Only the last 12 values written to the Output tape were visible on the Simulation screen. Both tapes can carry up to 96 values. The Output tape is to the right, and the Input tape is to the left, with a divider between them. Although the Output tape is read-only, here is where you can write to the Input tape.

Notice the usual cursor is in the bottom left corner. The value that goes here is the first value that the Little Man will read from the Inbox if you execute the program again. To add a value to the Input tape, press the Insert button (you will use the Right controller again on this screen).




Notice the border changes from light blue to green to indicate waiting for numeric input. Enter a number 1 to 3 digits and press Enter. Note you cannot enter negative values; the Input tape is unsigned.

As with code lines, the cursor can be moved to one position beyond the last number, so you can insert a new value at the end. You can also Insert a new value before the end. If you enter 12 values, you can then use the Scroll buttons to pan the display and view another column of Input values. Again, up to 96 values can be on the Input tape at a time, same as with the Output tape.

In the second illustration above, the first value is 400 and the second value is 300. If the program is run, the Little Man will fetch these values and subtract 300 from 400, resulting in 100 written to the Output tape, same as what is already written there. Press 5 on the Left controller to exit this screen.

## Example 2: Countdown

This program fetches a number from the Inbox and writes that number to the Output tape, and then counts down to zero, writing all successive numbers to the Output tape along the way.

First, press 1 to go to the Reset Menu and press 1 again to reset everything. Back at the Simulation screen, press 4 to go to the Code Editor.

```
00          INP
01          OUT
02    Loop  BRZ   Quit
03          SUB   One
04          OUT
05          BRN   Loop
06    Quit  COB
07    One   DAT   1
```

The new thing we are learning here is the use of branching, both conditional and unconditional.

Press 9 to enter the "INP" mnemonic (Remember the Code Editor uses the Right controller). It does not need a Reference. Press the bottom of the disc to enter the next code line.

Press 9 twice for "OUT" and press the bottom of the disc to enter the next code line.

Press the left side of the disc to move the cursor to the Label field, press Action, select "(New Symbol)", and type "Loop" and press Enter. On the Instruction field for this new code line (02), press 7 for "BRZ". The Symbols Pane opens. Select "(New Symbol)", type "Quit", and press Enter.

In the next code line, press 2 for "SUB". The Symbols Pane opens again. Select "(New Symbol)" and type "One" and press Enter.

In the code line after that, press 9 twice for "OUT", and press the bottom of the disc.

Press 6 for "BRN". The Symbols Pane opens. Move the cursor to "Loop" which is listed as Symbol 00 and select it by pressing Action.

Press the left side of the disc to move the cursor to the Label field and press Action. Move the cursor to "Quit" which is listed as Symbol 01 and select it by pressing Action. Leave the instruction as "COB" or "HLT" because that is what we want for this line and press the bottom of the disc to go to the next line.

Once more, press the left side of the disc for the Label field (if the cursor is on another field, it will move to the Instruction field automatically when you go to a different line). Press Action, move the cursor to "One" which is listed as Symbol 02, and select it by pressing Action.

Press 0 to change the mnemonic to "DAT" and enter the number 1. Go to the next line.

Ready to assemble. From the cursor's position below the last code line, press Action. Assembly should be successful.

Before returning to the Simulation screen, I will point out what the blue field to the right of the Symbol names is for. Look at the before & after illustrations on the next page.

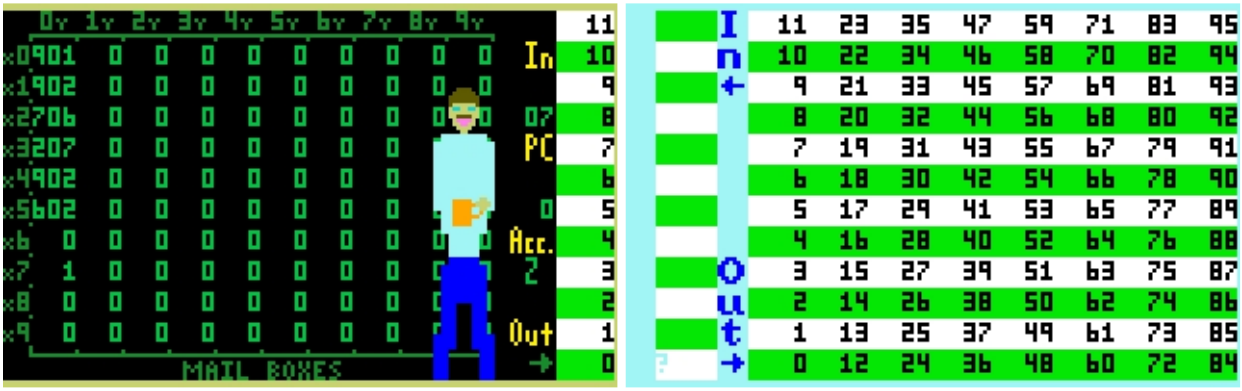

I had already renamed this project, which is why the project name “Countdown” appears in the top row. Notice the asterisk is still present in the top left corner of the first illustration, which means the code needs to be assembled. In the second illustration, opening the Symbols Pane again just after assembly displays 2-digit values in the blue field to the right of the Symbol names. These are the values of those Symbols. They represent the number of the code line where they are used as Labels. When you return to the Simulation screen and the Little Man performs magic, those values populate the last two digits in the mailboxes representing all the code lines using those Symbols as References.

For now, select “(Cancel)” and press 4 on the Left Controller to close the Code Editor and return to the Simulation Screen. After the magic performance, press 0 or Enter to put the Little Man to work. Which key you press determines which tune will play until the code execution is finished.

This program will fetch a value from the Inbox and write that value to the Output tape. Notice that nothing is ever stored to any of the mailboxes this time because we only care about one value, which is already on the Accumulator.

The four lines in the middle form an execution loop, running as many times as the number from the Inbox. Here is where branching comes into play. First, in line 02, we check if the value is already zero using the BRZ instruction (“BRanch if Zero”). If so, the line labeled “Quit” is where we branch to. The Little Man changes the Program Counter value to 06, and he will then fetch the value 0 from Mailbox 06, the code for the “COffee Break” instruction.

If the Accumulator value is higher than zero (remember the Input tape is unsigned, meaning it cannot hold negative values), then we subtract one from the Accumulator, write that value to the Output tape, and unconditionally branch to the beginning of the Loop.



If you enter a large value, you might want to hold the Fast Forward button, and when execution is finished, press 5 to see what a full Output tape looks like. Press 5 again when finished.

When you are ready, press 3 to enter the File Menu, press 1 to Save, and choose any file. If the project already had a name, you would not be asked to name it before saving. Press Clear until you return to the Simulation screen, and then press 1 to enter the Reset Menu. Press 1 again to reset everything, and then press 4 to go from the Simulation screen to the Code Editor.

## Example 3: Square

This will be our largest example, taking a number from the Inbox, calculating the square value of that number, and writing the result to the Output tape. We will explore the Code Editor in greater depth.

Two things before we get started here. First, if you ever misspell a Symbol and want to fix it, move the cursor to the Symbol, press Action, select "(New Symbol)", and type the correct name. If you are wondering how to delete the misspelled Symbol, that can only be done automatically by the Assembler. Second, if you ever need to insert a code line before the end, move the cursor up to the line after the new line you want to add, and press Insert. It will insert a code line above the cursor's position.

```
00      Start       LDA     Zero
01                  STO     Result
02                  STO     Count
03                  INP
04                  BRZ     End
05                  STO     Value
06      Loop        LDA     Result
07                  ADD     Value
08                  STO     Result
09                  LDA     Count
10                  ADD     One
11                  STO     Count
12                  SUB     Value
13                  BRZ     End_Loop
14                  BRN     Loop
15      End_Loop    LDA     Result
16                  OUT
17                  BRN     Start
18      End         COB
19      Result      DAT     0
20      Count       DAT     0
21      One         DAT     1
22      Value       DAT     0
23      Zero        DAT     0
```

By now, you should know how to get into the Code Editor, and which controller to use to write code.

00. We have a label right away, so press the left side of the disc, press Action, select "(New Symbol)", type "Start", and press Enter. Press 5 on the Instruction field for "LDA". The Symbols Pane opens. Select "(New Symbol)", type "Zero", and press Enter.
01. Press 3 for "STO", select "(New Symbol)", type "Result", and press Enter.
02. Press 3 for "STO", select "(New Symbol)", type "Count", and press Enter.
03. Press 9 for "INP" and press the bottom of the disc for the next code line.
04. Press 7 for "BRZ", select "(New Symbol)", type "End", and press Enter.
05. Press 3 for "STO", select "(New Symbol)", type "Value", and press Enter.
06. Move the cursor left, press Action, select "(New Symbol)", type "Loop", and press Enter. Press 5 on the Instruction field for "LDA". The Symbols Pane opens. At this point, you will notice an arrow inside of the Symbols Pane. Up to six Symbols can be displayed at a time, and now we have seven. This time, we are selecting an existing Symbol. Move the cursor to "Result" and press Action.
07. Press 1 for "ADD", select "Value", and press Action.
08. Press 3 for "STO", select "Result", and press Action.
09. Press 5 for "LDA", select "Count", and press Action.
10. Press 1 for "ADD", select "(New Symbol)", type "One", and press Enter.
11. Press 3 for "STO", select "Count", and press Action.
12. Press 2 for "SUB", select "Value", and press Action.
13. Press 7 for "BRZ", select "(New Symbol)" type "End_Loop", and press Enter. To get the underscore character, press 1 twice. The underscore is the only non-alphanumeric character available.
14. Press 6 for "BRN", select "Loop", and press Action.
15. Move the cursor left, press Action, select "End_Loop", and press Action. Press 5 on the Instruction field for "LDA". The Symbols Pane opens. Move the cursor up until "Result" comes into view. Select "Result" and press Action.
16. Press 9 twice for "OUT" and press the bottom of the disc for the next code line.
17. Press 6 for "BRN", move the cursor all the way up, select "Start", and press Action.
18. Move the cursor left, press Action, select "End", and press Action. The mnemonic is automatically set to "COB" or "HLT" which is what we want, so just go down to the next code line.
19. All the instructions are entered now, and all that remains are the DAT lines. It only makes sense for DAT lines to always be prefaced with a Label. Move the cursor left, press Action, move the cursor up to find "Result", select "Result", and press Action. Press 0 to change the mnemonic from "COB"|"HLT" to "DAT", type "0", and press Enter.
20. Move the cursor left, press Action, select "Count", and press Action. Press 0 for "DAT", type "0", and press Enter.
21. Move the cursor left, press Action, select "One", and press Action. Press 0 for "DAT", type "1", and press Enter.
22. Move the cursor left, press Action, select "Value", and press Action. Press 0 for "DAT", type "0", and press Enter.
23. Move the cursor left, press Action, move the cursor up to find "One", select "One", and press Action. Press 0 for "DAT", type "0", and press Enter.

With the cursor in the Instruction field beneath line 23, press Action to invoke the Assembler. If there are any errors, the Assembly Pane will show you which line has the error.

The first thing the Assembler does is check your use of Symbols. Every existing Symbol in your code must be used exactly once as a Label. If it is used more than once as a Label, the Assembler halts with an error about a duplicate Label. If a Symbol is not used as a Label, it cannot be used as a Reference either. Otherwise, the Assembler halts with an error about a Reference to a nonexistent Label. You can use the same Symbol as a Reference as many times as you like.

After ascertaining there are no errors, the Assembler rebuilds the Symbols list. Symbols are re-sorted in the order that they appear as Labels. Any Symbols not in use are purged from the list. The reason you cannot delete a Symbol yourself is that it would louse up any code lines that are using that Symbol. The list can contain up to 100 Symbols at a time.

Finally, the Assembler translates each code line into the appropriate 3-digit value for the associated mailbox. To determine the last two digits, the Assembler also generates a value for each Symbol. In the previous example, we took a moment to reopen the Symbols Pane just after assembly and see the 2-digit value that populates the blue field to the right of the Symbol names. The Symbol value is the number of the code line where that Symbol is used as a Label.

After assembly, no apparent change happened to the code. All the changes happened behind the scenes, but the Label and Reference names are the same in each code line.

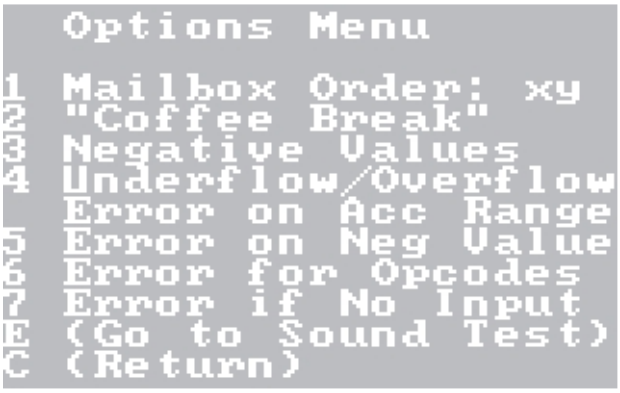
Now look at the end of the code for a moment. Little Man Computer was designed to teach the basics of computing and programming, and so there are some things we should be learning along the way. Among the DAT lines at the end, two of these are Constants: “Zero” and “One”. They are so-called because their value remains constant throughout the program. The other three – “Result”, “Count”, and “Value” – are Variables. The values in their locations vary throughout the program.

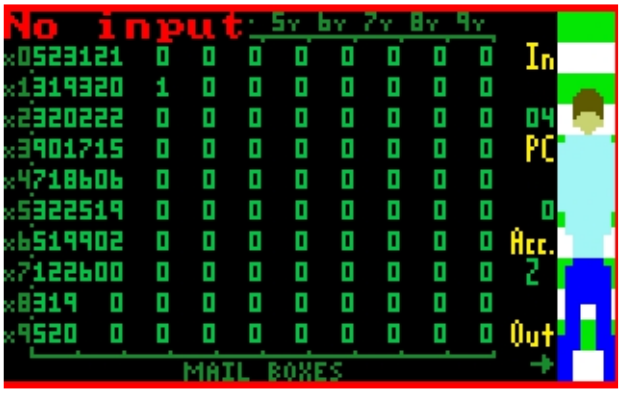
The first thing the program will do is zero out the Accumulator, because this one is designed to start over after it is finished, until it reads a zero from the Inbox. Next, that zero value from the Accumulator is used to zero out two of the variables, “Result” and “Count”. The third variable, “Value”, ends up getting its value from the Inbox unless that value is zero, in which case the program halts before that happens.

Lines 06-14 form the main program loop, which is why line 06 is prefaced with the label “Loop”. It will run as many times as the value from the Inbox. Each time it will add “Value” to “Result”, which was reset to zero earlier. “Count” is used to determine how many times the program has gone through the loop, and it is incremented each time. When the value of “Count” is equal to “Value”, the program branches to “End_Loop”. That is determined by the equation (“Count” - “Value” = 0). Remember that a value of Zero in the Accumulator decides whether the program branches to “End_Loop”, which is why the instruction is “BRanch if Zero”, or “BRZ”. Because branching happens under a condition, in this case the value of the Accumulator being zero, it is called a conditional branch. If that does not happen, we unconditionally branch back to “Loop”.

“End_Loop” starts at line 15, just after “Loop” is finished. The value of “Result” is loaded into the Accumulator and then written to the Output tape. After that, the program unconditionally branches back to “Start”, where it once again zeroes out the Accumulator and the values of “Result” and “Count”, and then fetches a value from the Inbox and checks if it is zero. If so, it branches to “End” at line 18, where it then halts so the Little Man can take another coffee break.

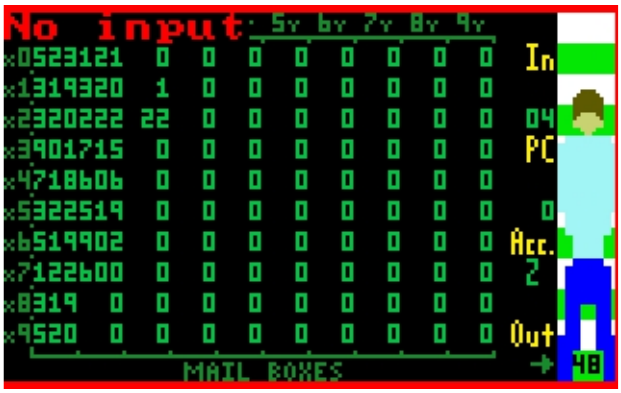
You should know how to return to the Simulation screen. Before executing the program though, let us take off our training wheels. Press 2 on the Left controller to open the Options Menu. From here, use keys 4, 5, 6, and 7 so that the program will halt with an error if: the Accumulator overflows (goes higher than 499), a mailbox receives a negative value (which can't happen in this example), a number is read as an instruction that doesn't exist (which shouldn't happen with assembled code), or if the Inbox is empty when the Little Man tries to fetch a value from there. Press Clear to return to the Simulation screen.




You should also know by now how to insert values onto the Input tape and how to execute the program. If you forget to insert values onto the Input tape before executing, the above error message is what you will see: "No input". The border turns red, the music changes, and the Little Man lowers his head. Press any keypad key, and the Little Man will return to his coffee and wait for you to fix the problem.

When you do insert a value onto the Input tape, avoid numbers higher than 22. Anything higher and the Accumulator will go over 499, resulting in an Overflow error. I suggest something small, like 3, for now.


So, you inserted a value into the Input tape and ran the program. The square value of your number was printed to the Output tape, but then you might have gotten a "No input" error message again. What went wrong? A lazy person would disregard the error because the desired output was obtained anyhow. But we are going to go the extra mile and solve the problem.

The error happened *after* the result was written to the Output tape, so it was somewhere after "End_Loop". There is our clue. Recall that this program is designed to keep running until a zero is read from the Inbox. Go back to the Tape Viewer, insert your number again, but also insert a zero in the line above it. Remember that the bottom number goes through the Inbox first.

Once you are successful in running the program without any errors, go to the File Menu (you should remember how to do so), name your project “Square”, and save to any file you choose. On to our fourth and final example.

## Example 4: Quine

Some *very* smart programmers like to challenge themselves by trying to write a type of program known as a Quine. What is a Quine? It is a program that outputs its own code. Since Little Man Computer only writes numbers onto the Output tape, all we can display are the numbers written in the mailboxes that comprise the program, not the code in the Code Editor.

This is a smaller example, and it is far beyond the scope of what would be expected from beginning programmers. However, it is included here to stretch your imagination as to what you can accomplish through programming. We will also explore the features of Little Man Computer a little further.

First, go to the Reset Menu and reset everything. Then navigate your way to the Code Editor.

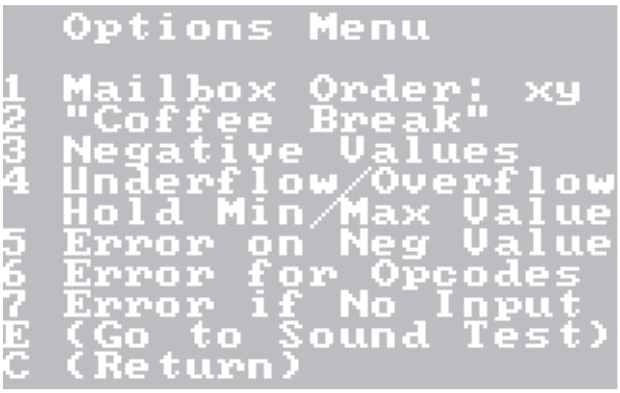
```
00      Zero    LDA     Zero
01              OUT
02              SUB     One
03              BRZ     One
04              LDA     Zero
05              ADD     One
06              STO     Zero
07              BRN     Zero
08      One     DAT     1
```

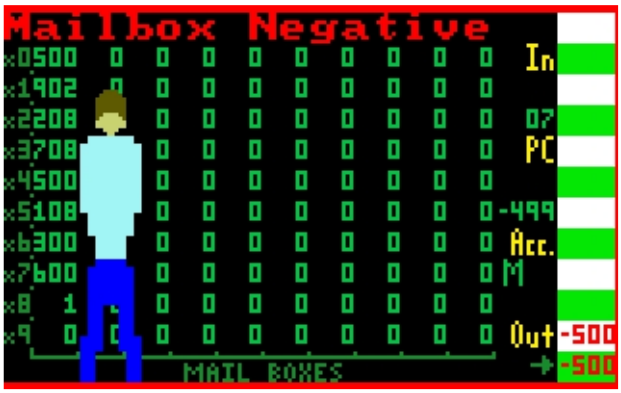
Because this is the final example, test your ability to enter this code and assemble it by yourself. Refer to the Right controller overlay as often as necessary. There are only two Symbols to create this time. After assembling the code, return to the Simulation screen and try running it.


It threw an error. It seems the Quine depends upon the exact behavior of Little Man Computer. Can we get the Output tape to mirror the first nine mailboxes? Let us give it the old college try.

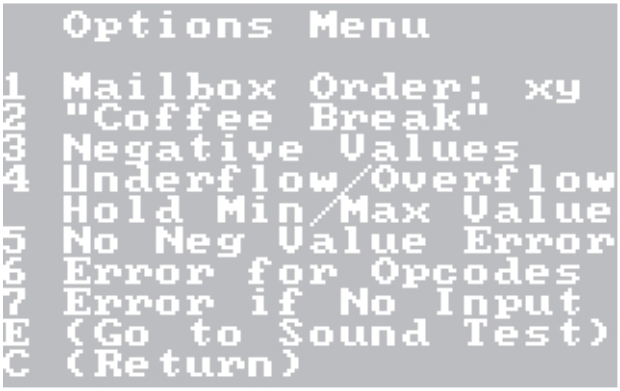
Return to the Options Menu. The error was thrown because the Accumulator went below -500, resulting in an Underflow. Option 4 defines what to do in the result of an Underflow/Overflow. Change it to “Hold Min/Max Value” for now. There is a third choice for this option, but we will try this one first.

Return to the Simulation screen and try running the program again. Now what happens?



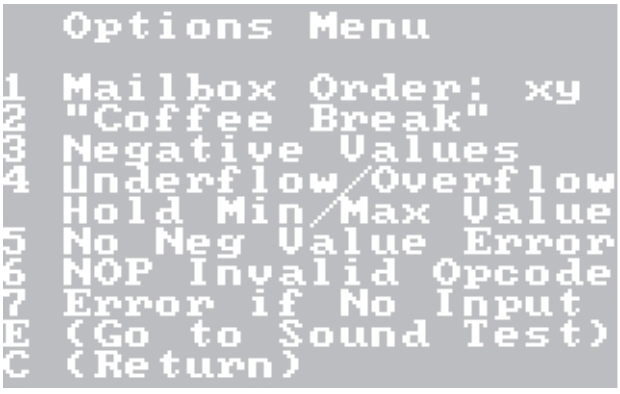

Another error is thrown because a negative value was about to be entered into a mailbox. Let us fix that as well in the Options Menu. This is Option 5. Change it to "No Neg Value Error". Return to the Simulation screen and try running the program once again.

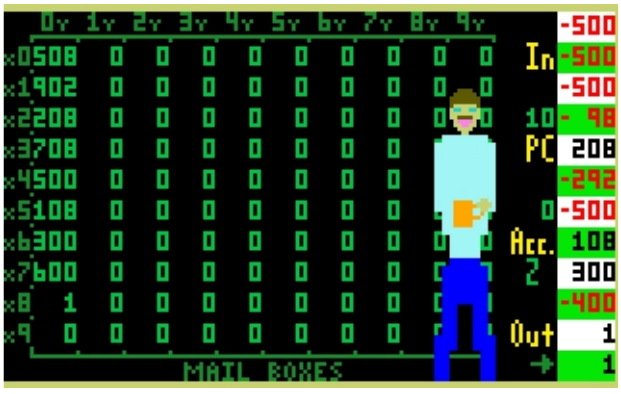




Another error. This time the Little Man read "1" as an instruction. How can that be if we assembled the code? This is another important lesson to learn. As a programmer, you need to differentiate between code and data. The "1" is in Mailbox 08, whose code line has a Label followed by "DAT 1". It turns out the Quine is double-dipping, using that "1" as data to periodically increment Mailbox 00, and counting on the Little Man to disregard the fact it is not a valid instruction.

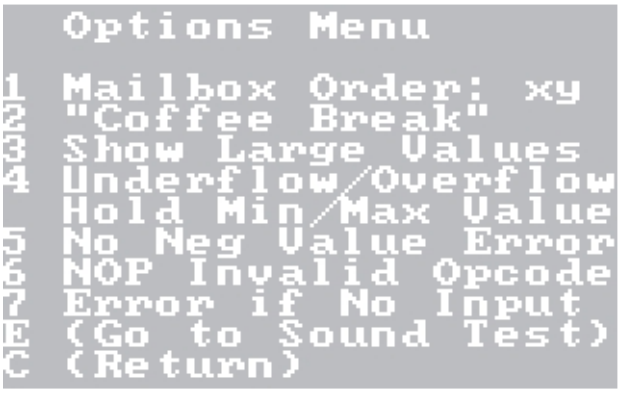
Back to the Options Menu. Option 6 can be changed back to "NOP Invalid Opcode". Return to the Simulation screen and try executing the program yet again.

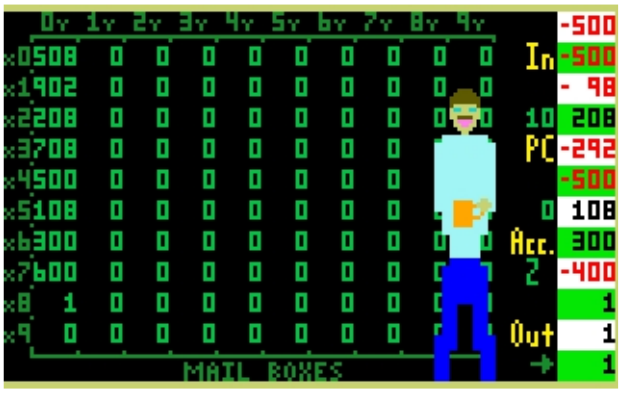


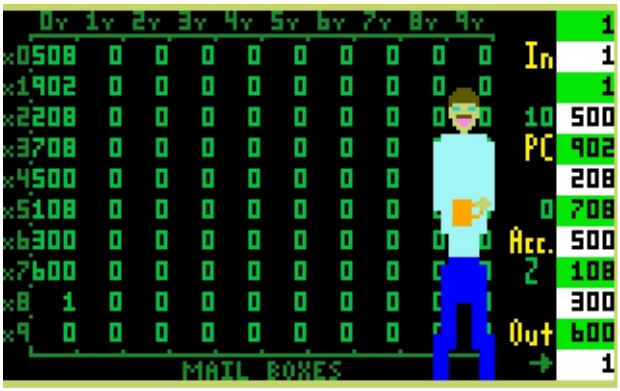
Well, the good news is the program is not throwing any more errors. Too bad the Output tape looks nothing like the mailbox values. Now is a good time to point out that negative values are printed in red on the Output tape.

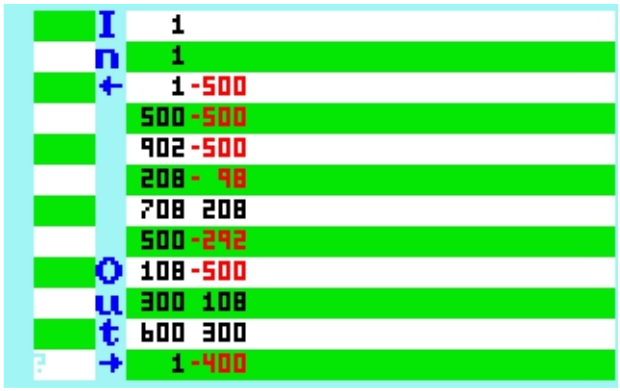
Wait, negative values? There cannot be negative values in the mailboxes, so let us change the option as well to display large values in the Accumulator. One more trip to the Options Menu. Press 3 for "Show Large Values". Return to the Simulation screen and execute the program again.




Not much happened, same as last time. Recall that Mailbox 00 is getting continually incremented. The easiest way to restore the mailbox values is to go in and out of the Code Editor. Because code exists and has not been modified since the last time it was assembled, the Little Man will do his magic performance. Now let us run the program once more.

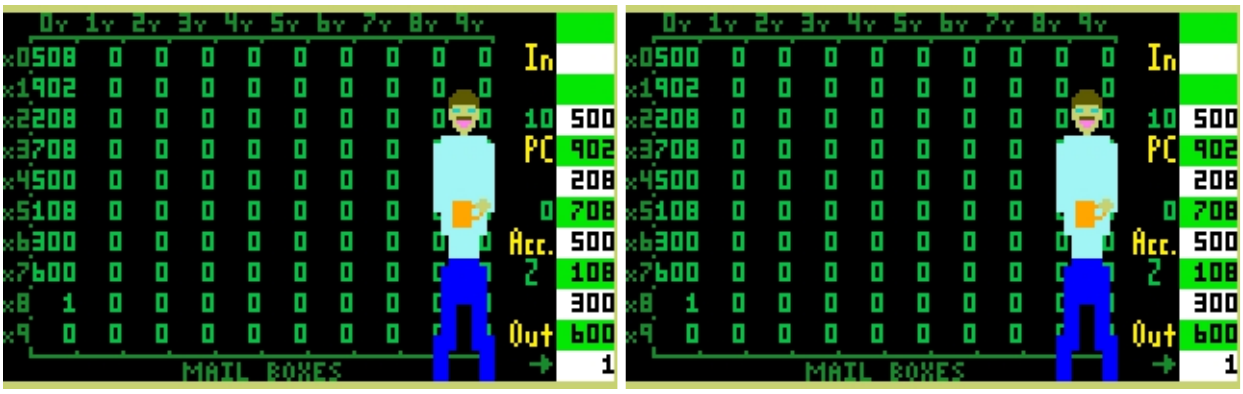



We are getting close now. Remember that Mailbox 00 keeps getting incremented while the program is running, so its value is different than it was when the first write to the Output tape happened.

We should clear the Output tape as well before running the program again. But first, go to the Tape Viewer for a moment. Observe there are both negative values and large values on the Output tape. Once something is written to the Output tape, it remains that way, even if the Option is changed between showing Negative Values and Large Values on the Accumulator.

Return to the Simulation screen, go to the Reset Menu, and select 9. Return to the Simulation screen again, open and close the Code Editor to restore the mailbox values, and run the program again.




Open and close the Code Editor one last time to restore Mailbox 00. Done!

Save this program and name it “Quine”.

## Conclusion

You now have a simulation of a theoretical computer, and a conceptual understanding of basic programming concepts. Although this is a primitive computer model, you can look up other examples of programs you can create and execute on Little Man Computer. Have fun!

## Control Reference

| | Left Controller | Right Controller |
|---|---|---|
| 1 | Go to Reset Menu | Write “ADD” mnemonic |
| 2 | Go to Options Menu | Write “SUB” mnemonic |
| 3 | Go to File Menu | Write “STO” mnemonic |
| 4 | Go to Code Editor | Write “STA” mnemonic |
| 5 | Go to Tape Viewer | Write “LDA” mnemonic |
| 6 | Set value of Program Counter | Write “BRN” mnemonic |
| 7 | Set value of a mailbox | Write “BRZ” mnemonic |
| 8 | Set value of Accumulator | Write “BRP” mnemonic |
| 9 | Write value to Output tape | Write “INP” or “OUT” mnemonic |
| Clear | Stop execution if running | Delete code line.<br>Replace Symbol |
| 0 | Start execution in Step-Through Mode<br>Change to Step-Through Mode if running | Write “COB” or “HLT” mnemonic<br>Write “DAT” mnemonic |
| Enter | Start execution, or pause if running | Enter text or a number |
| Top buttons | Hold during execution to fast-forward | If cursor in Label or Reference field, open Symbols Pane<br>If cursor in Instruction field of existing code line, insert a new line above.<br>If cursor in Instruction field of new code line, assemble code.<br>If Symbols Pane open, make selection |
| Bottom buttons | Move the Little Man | Scroll |
| Disc | Turn to adjust music volume | Move cursor |

## Glossary

**Accumulator**: a register in which arithmetic operation results are stored for immediate access.

**Assemble**: to convert written code into machine code. In Little Man Computer, "machine code" refers to the 3 digits written in each of the mailboxes.

**Assembler**: a program (distinct from a Little Man Computer program) that reads each line of written code and converts it into the 3-digit value for each mailbox that the Little Man reads as instructions.

**Assembly**: the act of converting written code into the 3-digit values for each mailbox that represents one of the code lines. Such 3-digit values when used as instructions are called Opcodes.

**Binary**: anything that involves two of some quantity. Here it is used to refer to a numbering system where each numeric character, or digit, can have only one of two values. The prefix "bi-" means "two".

**Branch**: to diverge from the main path. In programming, it is so called because the "main path" is to follow all the program's instructions in sequence.

**Character**: any written or printed letter, number, or other symbol (not to be confused with code Symbols) that has significance.

**Conditional (Branching)**: branching that happens only under a certain condition, such as whether the Accumulator's Zero flag is raised. See "branch" for more information about branching.

**Constant**: a data value that does not change throughout the execution of a program.

**Decimal**: anything that involves ten of some quantity. Here it is used to refer to the numbering system people are commonly used to, where each numeric character, or digit, can have any one of ten values. The prefix "deca-" means "ten".

**Equivalent**: equal in value, amount, function, or meaning.

**Flag**: a piece of data that can be in any one of two states – raised or not raised – at a given time, each of which conveys a specific meaning. It is supposed that the term comes from a time when a person was called a "flagman" who would stand on an airplane runway or railroad line, carrying a brightly colored *flag* and either raising or lowering it to communicate a message to the pilot or train engineer.

**Give It the Old College Try**: an idiom which means to give something your best effort even if success seems unlikely. It refers to the 1930s when college football films became popular, which portrayed a team that was losing every game and had poor morale, which suddenly through a combination of changed decision-making, strenuous effort, and luck, suddenly overcomes staggering odds and ends up winning the big game and becoming the #1 team of the season.

**Go the Extra Mile**: an idiom which means to put in more effort than other people typically would, to achieve something. It is often used in reference to the New Testament, in which Jesus teaches "If anyone forces you to go one mile, go with him two miles."

**Increment**: to increase a value by 1.

**Instruction**: a single command carried out by a computer during program execution. There are several steps involved to carry out each command.

**Instruction Set**: the complete set of all the different instructions that can be carried out by a computer's CPU (Central Processing Unit), which is what the Little Man represents.

**Label**: a Symbol used at the beginning of a code line that serves as a destination point. For example, a program may have a code line containing the word "Done" followed by the instruction to halt execution (take a coffee break). One or more other code lines would contain an instruction to branch to "Done".

**Mnemonic**: pronounced with the first 'm' silent, "nemonic", an association of any kind that helps somebody to remember something else. For example, the letters "DAT" are used to represent the word "data", letting the programmer know that a code line contains data rather than code.

**NOP**: pronounced "No Op", an abbreviation for "No Operation" which is an instruction to do nothing. Little Man Computer does not have an actual "No Op" instruction, but one of the options allows for the Little Man to disregard 3-digit codes that are read as an instruction (an Opcode) but are not defined as such, essentially treating it as "No Op". An example would be a number higher than 902.

**Opcode**: an abbreviation of "Operation Code", the official term for the 3-digit numbers that are read as instructions.

**Overflow**: to suddenly try to make something hold a quantity or value greater than what it can contain. To understand why the term "*overflow*" is used, pretend the Accumulator is a tub large enough to hold 499 gallons of water. If somebody were to pour 500 gallons of water into it, the tub would *overflow,* and the last gallon of water would spill over the top. The Accumulator has a flag labeled 'O' that can be raised to signal when it has *overflow*ed. There are three choices, specified in the Options Menu, for what to do in such a case.

**Pane**: a rectangular area on the screen that contains information. There are two *pane*s in the Code Editor that appear when necessary: one to select a Symbol from a list or create a new one, and one that displays a message during assembly.

**Program**: a collection of commands that can be carried out by a computer to perform a specific task.

**Program Counter**: a register that contains a value indicating which instruction is the next one to follow.

**Raised**: one of the states of a flag, which is communicating something because of being in that state. For example, the Accumulator has three flags. One of them is called 'Z', and it is *raised* every time the Accumulator has a value of zero, by displaying the letter Z.

**Reference**: a Symbol used after an instruction, which points to a destination. The destination is the code line which contains the same Symbol *before* the instruction. There is a "many-to-one relationship" between *Reference*s and Labels, meaning there can be one or more *Reference*s for every Label. The advantage of writing code and then having it assembled is that the programmer does not need to keep track of that destination; the Assembler will figure it out.

**Signed**: something that can have a negative value. It is so called because of the presence of a "Minus sign" whenever the value is negative.

**State**: any one of multiple conditions that something can be in. It is used to describe whether each of the Accumulator's flags is raised or not. It is also used in the Reset Menu as a collective term for the computer's Input stream, Program Counter, and Accumulator.

**Symbol**: an identifier that has a form recognizable to humans. It is the same concept as a pictorial *symbol* which conveys a certain meaning, such as a road sign. In programming, it usually takes the form of a Label and any number of References. To understand why that term is used, consider the word "Done" as an example. The Assembler does not understand what "Done" means and doesn't need to understand, beyond its ability to link "Done" as a Label together with "Done" as a Reference, which a machine can easily do.

**Take a Weight Off [Our] Shoulders**: an idiom which means to remove a mental burden from somebody, resulting in a feeling that a heavy object once slung over that person's shoulders is suddenly gone. It is used here to explain that writing code relieves us of the problem of having to keep track of which mailboxes contain data and which mailboxes contain instructions.

**Take Off Our Training Wheels**: an idiom that alludes to training wheels on a bicycle which should not otherwise be present (hence the name "bicycle", meaning "two wheels") but are temporarily used until a child has learned to stay balanced and not fall over. The "training wheels" we are referring to here are certain choices in the Options Menu which are set by default and prevent programs from halting with an error under certain conditions.

**Unconditional (Branching)**: branching that always happens regardless of any conditions. See "branch" for more information about branching.

**Underflow**: like "Overflow" but for the opposite condition – trying to make something contain a value lower than the minimum value. The Accumulator's lowest possible value is -500, so if a number is subtracted from its current value that would set it below -500, the 'O' flag is raised which is so called because it is the same flag used to signal an Overflow. There are three choices in the Options Menu for what to do if an *Underflow* or Overflow occurs.

**Unsigned**: something that can only hold positive values, opposite of "signed". Because there are no negative values, the maximum value is twice as high as for similar things that are signed. For example, the Input stream and mailboxes are *unsigned* and have a maximum value of 999. The Accumulator, on the other hand, only has a maximum value of 499 because half of its range of possible values are negative numbers. Little Man Computer was designed to teach students to be aware of interactions between things that are signed and things that are *unsigned*.

**Variable**: a data value that can change throughout the execution of a program; it varies.

**Wipe the Slate Clean**: an idiom which means to act as though something in the past never happened. It usually has an emotional connotation, meaning to forgive somebody of past wrongdoing. Here it means to reset the values of various computer components.

**Wrap Around**: to act of going from the highest possible value to the lowest possible value, or vice versa, in the event the Accumulator overflows or underflows. The idea comes from analog meters that do just that – if one can contain 6 digits for example, its highest value would be "999,999". Attempting to increment it would change its value to "000,000" rather than one million. See "overflow" and "underflow" for more information.

**Zero(es) Out**: the act of changing the value of something to zero.

## About World Book TutorVision

In the late 1980s, when the Intellivision Intellectual Property was owned by INTV Corporation, INTV made a deal with World Book Encyclopedia to create an educational video game system. It appeared in INTV catalogs and was demonstrated at the 1989 Consumer Electronics Show, which took place in Chicago. Fourteen games are known to have been developed for it, half of which were aimed at younger children ("Level 1"), and the other half of which were aimed at older children ("Level 2").

Shortly afterward, INTV filed for bankruptcy protection, and the TutorVision project died before it could be test-marketed. Since then, a few TutorVision units have been found, mostly in thrift shops in and around Chicago. At least one of the units was reverse-engineered and discovered to have five key differences from a standard Intellivision console:

- Different data in the Graphics ROM, providing for a different "system font" – compare the title screenshots at the top of Page 3,
- Four times as much Graphics RAM, allowing for 256 "tiles" to be pre-defined instead of 64,
- An additional library in the system ROM mostly containing routines to accommodate for the additional Graphics RAM, mainly to pack two characters into a single 8x8-pixel tile. This extra library is colloquially known as "REX", an abbreviation of "Revised EXecutive ROM."
- A modified version of the chip which handles graphics and collision detection,
- Additional 16-bit memory.

The last batch of Intellivision consoles made in the 1980s were labeled "INTV Super Pro System". A handful of these "Tutor Pro" units, as they are called, were found to have some TutorVision hardware in them, missing only "REX". One of the original TutorVision units that was found also included two of the games that were developed for it: *Map Mazes* and *Shapes In Space*. Those two games were graciously ROM-dumped and are available for download and use with jzIntv, the one Intellivision emulator designed to emulate the TutorVision as well.

## Developer's Notes

I recently discovered the two above game ROM images and tried them out, and I was immediately inspired to create something that would be appropriate for the TutorVision. If I wanted to develop for a console that was merely an Intellivision with additional Graphics RAM, I would have created a text adventure instead. At around the same time, I learned about Little Man Computer, and I set out right away to conceptualize how it would work on TutorVision. It takes full advantage of the additional Graphics RAM, so that all the components of the Little Man Computer can be displayed on the screen at one time, along with the animated namesake Little Man.

The disadvantage of the TutorVision is the presence of REX. It is in an address range that makes the TutorVision incompatible with the ECS, Mattel Electronics' second attempt to expand the Intellivision into a computer. Because of this incompatibility, the Computer Keyboard cannot be connected and used, and so I had to come up with a way to allow for code entry using only the two hand controllers. Worse, the program ROM is over 38.5K in size, which would have been considered massive back in the day. I had to squeeze every address range I could into the ROM, excluding the range where REX is present. When I recalled the ECS cannot be used, I gleaned all the available ranges normally set aside for ECS. It turned out to be just enough space to finish this program.

I also wanted to allow for saving one's work, which would have taken advantage of the ECS' connection to an Aquarius Data Recorder. The alternate solution was to use JLP, which allows for data to be saved to cartridge EEPROM. In addition, JLP provides for nearly 8K of additional 16-bit memory, of which Little Man Computer uses over 1.2K – also considered massive, especially since original Intellivision games only had approximately 140 bytes of 8-bit memory. Even the extra TutorVision memory is not enough.

## Thanks


- To Joseph Zbiciak, for developing JLP, for maintaining jzIntv – by far the best Intellivision (and now TutorVision) emulator available, and for helping to reverse-engineer the TutorVision.
- To Oscar Toledo G., whose IntyBASIC language made development much easier.
- To Chuck Gill for finding the two TutorVision games and allowing the ROMs to be dumped, and to David Harley and Alex Pace for dumping them.
- To Chris Neiman, Mike Thomasson, and "Rev", who each published earlier titles of mine.
- To all the people who kept me encouraged every step of the way until I regained my confidence in my development ability and created my Portable Intellivision Development Environment.
- To you, for reading through this manual and trying my implementation of Little Man Computer, making my time and effort worthwhile.


## About Midnight Blue International

It is our intention to create games in abundance for a variety of purposes: to unlock innate imaginative potential, for enjoyment in social gatherings, for rest and relaxation, and to stimulate critical thinking.

Contact info and information about other games are on our website:
www.MidnightBlueInternational.com