



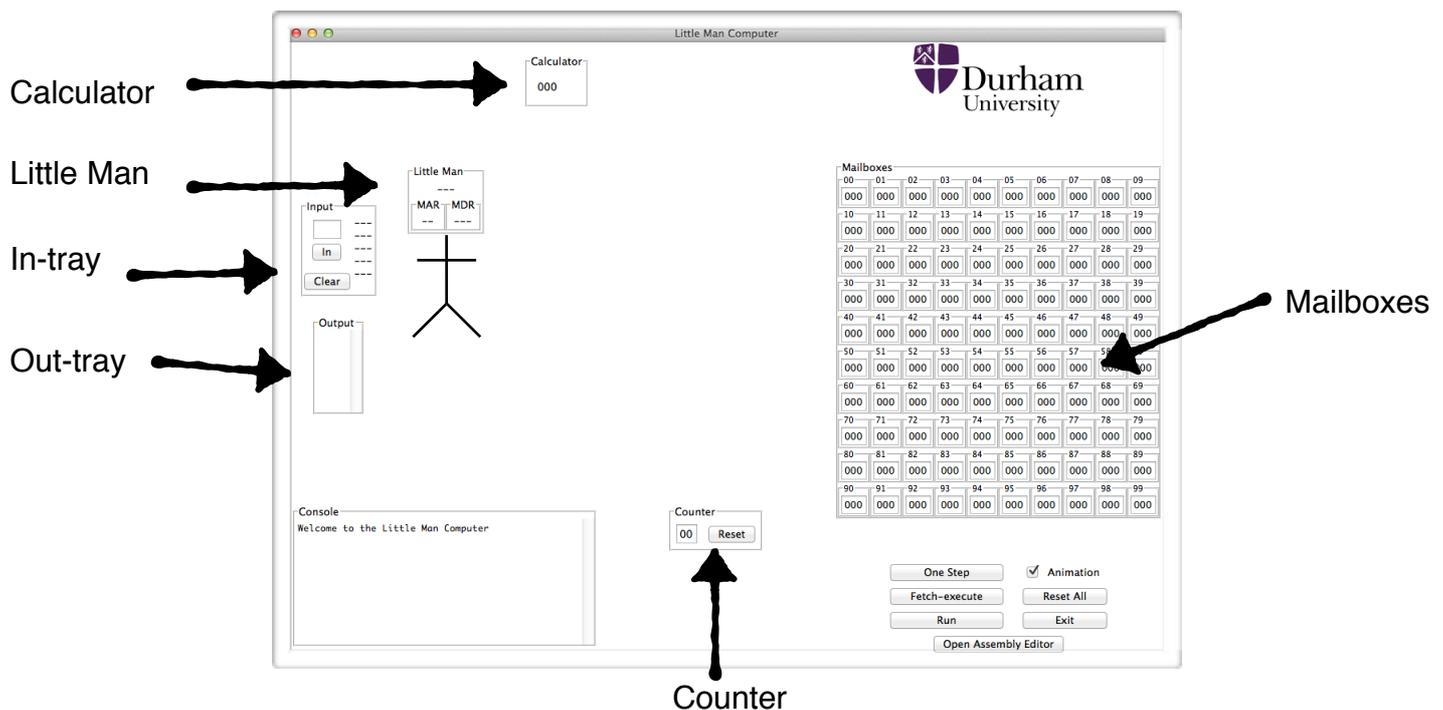
Little Man Computer

The Little Man Computer is a conceptual model of a simple CPU, introduced by Dr. Stuart Madnick of M.I.T. in 1965. Although it seems simplistic, in fact the model captures many important features of a real CPU and illustrates these in an accessible way.

The Little Man Computer

The conceit is that inside the CPU is a tiny man that runs around organising data and performing the calculations. Inside the box of the CPU are:

- 100 numbered mailboxes
- A calculator
- A 2-digit counter
- An in-tray
- An out-tray



The **mailboxes** each have a 2-digit address, so 100 mailboxes is the limit, counting from 00 to 99. Each mailbox contains a slip of paper with 3 digits on it, and can only ever contain a single slip of paper at a time.

The **calculator** that can only display 3 digits on the screen, and has buttons for numbers 0-9 and + and - . It also has an LED that lights up when a larger number is subtracted from a smaller number. This LED is known as the negative flag.

The 2-digit **counter** has two buttons: the first button increments the counter by 1, the second (external) button resets the counter to 00.

The **input** and **output trays** can each contain stack of slips of paper, each of which has 3-digits on. The user can put slips of paper with 3 digits on in the input tray, to be read when the little man next looks at his in tray. Likewise the little man can write 3-digit notes and put them in the out tray, to be read by the user.

The Fetch-Execute cycle

Or what the little man does...

1. The little man starts by looking at the counter for number, which is a mailbox number.
2. He increments the counter, so that next time he comes it will be one larger.
3. He goes to the mailbox with the number he read on the counter, and reads what is written on the slip of paper in the mailbox, i.e. 3 digits.
4. He takes the appropriate action depending on those digits
5. He then starts again...

Steps 1-3 are the **fetch**: the little man is fetching an instruction. Step 4 is the **execute**: the little man does the indicated instruction.

The instructions:

The little man will read a 3-digit message in the mailbox he is sent to, e.g. 5 8 4, which is his **instruction**. The first digit of the instruction is the operation: 5. The second and third digits give another mailbox number: 84. The operation part is also known as an “operation code” or “**op code**”, the other two numbers are an **address**. Op code 5 is **load**: the little man goes to the mailbox indicated by the address, copies what is on the slip of paper in the mailbox, then goes to the calculator and enters that number. E.g. if mailbox 84 contained 1 2 3, then when the little man reads instruction 584, he goes to mailbox 84, copies down 123, then goes to the calculator and enters 123.

Let's define some more op codes and give the little man a task.

ADD – op code 1

go to the mailbox address specified, read the 3-digit number at that address, then go to the calculator and add the number to the number already on the calculator.

SUBTRACT – op code 2

go to the mailbox address specified, read the 3-digit number at that address, then go to the calculator and subtract the number from the number already on the calculator.

Note: the mailboxes will still contain the same 3-digit messages as before, but the calculator will have changed. If the subtracted number is larger than the value already in the calculator (i.e. the subtraction ends with a negative value), then the **negative flag is set/activated** and the value in the calculator cannot be trusted to be right.

STORE – op code 3

go to the calculator and note the 3-digit number displayed, then go to mailbox address specified and enter that number on a new slip of paper.

Note: the calculator will still contain the same 3-digits as before, but the mailbox will have whatever was there before discarded.

LOAD – op code 5

go to the mailbox address specified, read the 3-digit number at that address, then go to the calculator and enter that number in.

INPUT/OUTPUT – op code 9

INPUT or READ op code 9 address 01

go to the IN tray, read the 3-digit number there, then go to the calculator and enter the number in.

Note: The slip of paper in the IN tray with the 3-digits is removed. If there are several slips in the IN tray, the little man takes the first one that was put there only, the others remain for future visits.

OUTPUT or PRINT op code 9 address 02

go to the calculator, read the 3-digit number there, then go to the OUT tray and leave a slip of paper with that number on it.

BREAK – op code 0

The little man stops and has a rest.

A program

When the Little Man Computer is started, the counter is reset to 00, and the mailboxes will already contain some values, so the little man starts by visiting mailbox 00 and executing the instruction there. What he does after that depends on the instructions!

Look at the initial mailbox contents given to the right. What will the little man do? What does this achieve?

Mailbox	Contents
00	901
01	306
02	901
03	106
04	902
05	000
06	000
07	000
08	000
09	000

Mailbox 06 is used here as a temporary storage for a value, since it is empty and unused by the program. Why didn't we use mailbox 05, the first 000 value mailbox?

Exercises

Enter values in the mailboxes so that when the Little Man starts (with counter at 00)

- He takes 3 inputs values and then outputs the sum of all of them
- He takes two input values, a and b, and outputs the difference, a - b.
- What happens if you subtract something large from something small? E.g. 350 from 100

More instructions

The instructions so far can be grouped into categories as follows:

Data movement:

LOAD – op code 5
STORE – op code 3

Arithmetic:

ADD – op code 1
SUBTRACT – op code 2

Input/Output:

INPUT – op code 901
OUTPUT – op code 902

Machine control:

BREAK – op code 0

To get the Little Man to do more powerful calculations, we need to have more control over the machine and in particular the flow of the program - i.e. which instructions he does when. We do this with **branch** instructions.

BRANCH – op code 6

set the counter to the 2-digits specified in the address.

Note: The next instruction he reads is the one in the mailbox address specified.

To have even greater control we allow two more complex branch instructions: **conditional branch** instructions.

BRANCH on ZERO – op code 7

go to the calculator and read the 3-digit number. If it is zero, set the counter to the 2-digits specified in the address, otherwise do nothing.

BRANCH on POSITIVE – op code 8

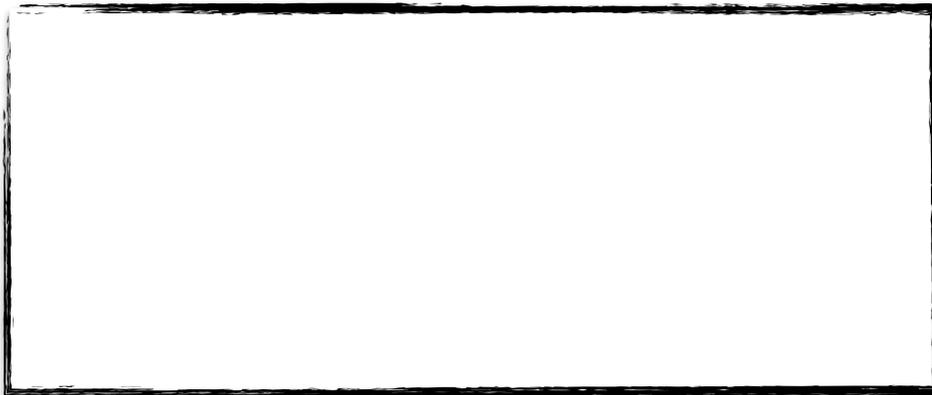
go to the calculator and check the negative flag. If it is not activated, set the counter to the 2-digits specified in the address, otherwise do nothing.

Now we can construct simple loops:

```
value = INPUT
do while value >=0
  print value
  value --
next
end
```

Mailbox	code	Description
00	901	INPUT value
01	902	OUTPUT
02	299	SUBTRACT 1
03	705	BRZ
04	601	BRANCH
05	902	OUTPUT
06	000	STOP
07	000	
08	000	
09	001	DATA - 1

And we can test conditions. What does this program do?

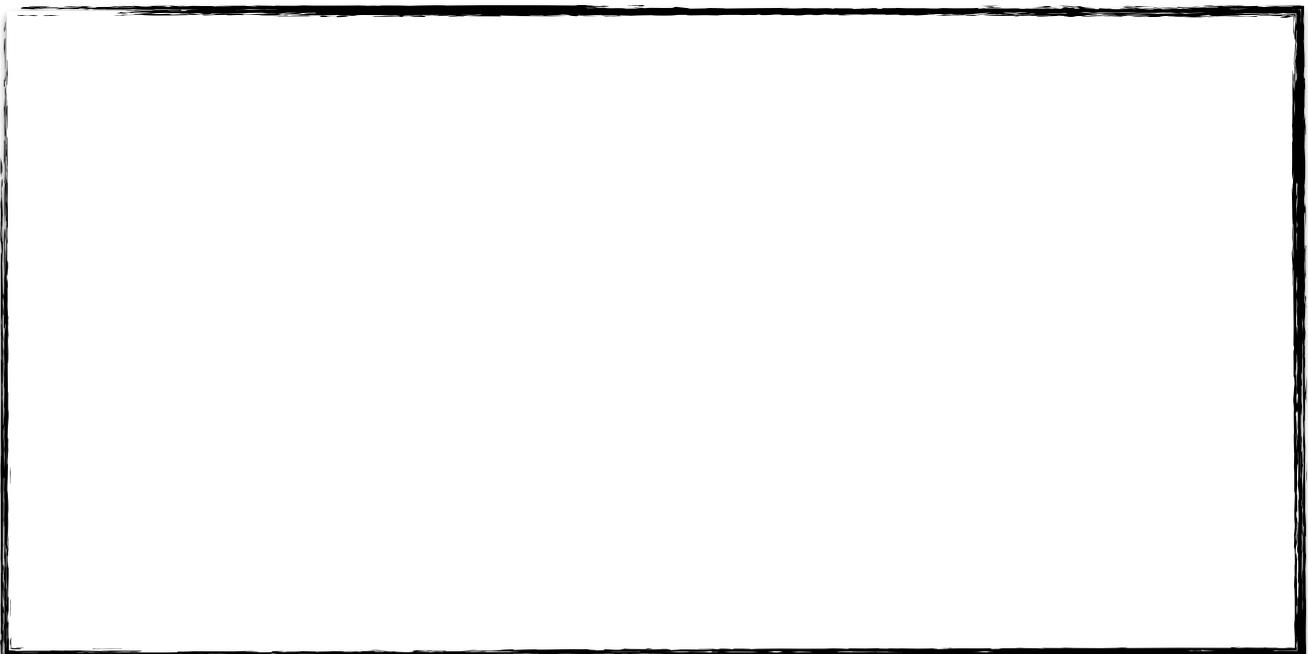


Mailbox	Contents
00	901
01	311
02	901
03	312
04	211
05	808
06	512
07	609
08	511
09	902
10	000

Exercise

Enter values in the mailboxes so that when the Little Man starts (with counter at 00)

- He takes 2 inputs values, a and b, and then outputs all integers between the two inputs.
- First assume $a < b$. Then try and get it to work for inputs either way round.



Machine code and assembly language

This style of coding is called **machine code**, since we are programming directly in the code the Little Man understands. The first step towards making things easier for the programmer is to use **assembly language** - human readable mnemonics for the machine code. Since we are now going to write our code as a list of mnemonics without reference to specific mailboxes, we shall also label lines of code that we might want to branch to. So each line of our assembly code will have:

- a label (possibly blank),
- an operation given as a mnemonic, and
- an address, given as a label (possibly blank)

Here is an example of assembly code for the program to add two numbers and output the result:

```

        IN
        STO A
        IN
        ADD A
        OUT
        HLT
A      DAT
```

Note that each line starts with a label or a tab, then has a mnemonic, then if the interaction needs an address, has another tab and an address label. We give this text to a special program called a compiler that will turn it into machine code and fill suitable mailboxes with the result. There is a compiler in the Little Man Simulator.

Mnemonic	op code	Description
ADD	1xx	Add
SUB	2xx	Subtract
STO	3xx	Store
LDA	5xx	Load
BR	6xx	Branch
BRZ	7xx	Branch on zero
BRP	8xx	Branch on positive
IN	901	Input
OUT	902	Output
HLT	000	Halt or Stop
DAT		Data storage location

Further Exercises

1. Give assembly code for your earlier programs, e.g. the one to output all numbers between two input values.
2. Create a Little Man Computer program to output the Fibonacci numbers. Look up Fibonacci numbers online if you are not familiar with them.
3. Create a Little Man Computer program to output the first ' n ' Fibonacci numbers. Your program should take the input n and then output the first n elements of the sequence. I.e. 1, 1, 2, 3, 5...
4. Create a Little Man Computer program to take two inputs a , b and compute $a \times b$.
5. Create a Little Man Computer program to an input a and compute a divided by 2.
6. Create a Little Man Computer program to take two inputs a , b and compute a divided by b .
7. Create a Little Man Computer program to take inputs until an input of 0 is received, then output the sum of the inputs.
8. Create a Little Man Computer program to take inputs until an input of 0 is received, then output the smallest of the inputs.
9. Create a Little Man Computer program to take two inputs and output the highest common factor (look up Euclid's algorithm).

These notes are based on the description found in the book: The Architecture of Computer Hardware and System Software: An Information Technology Approach, 3rd edition, by Irv Englander.