# Advanced challenges

3. Write a program that will ask for 2 numbers and then multiply them. While this may be tricky, you should now know enough to do it!

4. How about a program that will divide two numbers and give the DIV and MOD. DIV is the whole number result of a division. MOD is the remainder.

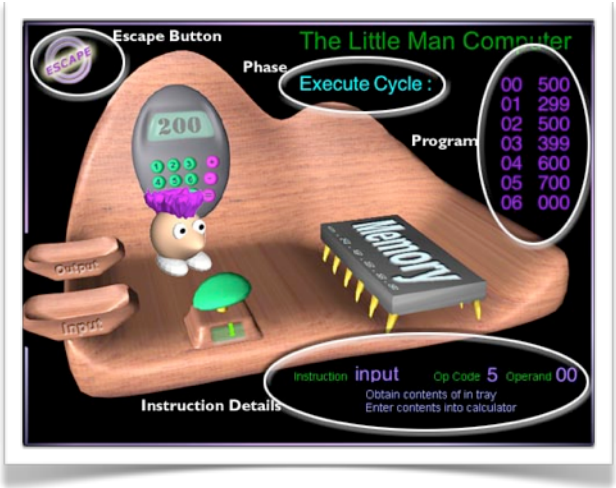
   e.g. 17 ÷ 5 = 3 remainder 2

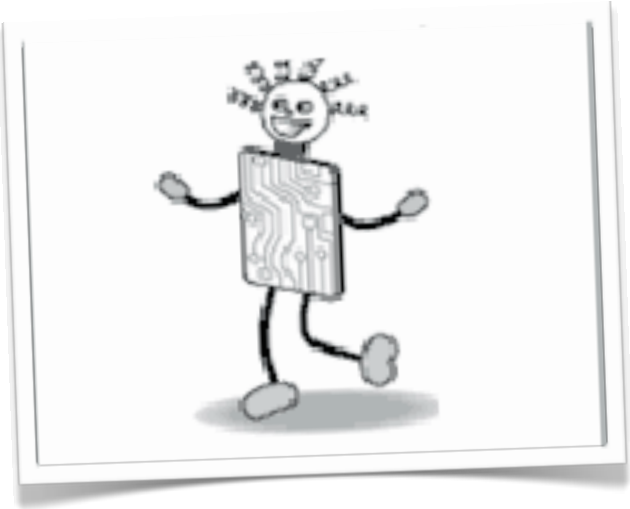
5. Try writing a program that will check if two numbers are a factor of each other. First enter a big number, then a small number. If the small number is a factor then it should divide with no remainders.

6. Try improving program 5 so that it doesn't matter which way round you enter the numbers.

# Little Man Computer

## A CAS Master Teacher CPD Session





Mark Clarkson
December 2013

# Your First Program

Some key points:

- **ALWAYS** copy your code before compiling, as you will lose it

- Remember readability:
  - LMC will ignore blank lines
  - LMC is not case sensitive, but good habits help

Message Box:

```
INP
STA numOne

INP
STA numTwo

LDA numOne
ADD numTwo

STA numThree

OUT

HLT

numOne DAT
numTwo DAT
numThree DAT
```

# Advanced challenges

2. You can declare a constant at the end of the program like this:
   `one DAT 1` (this will give the variable 'one' the value 1)
   Using this, add to your previous program to count the number of times you can successfully subtract the smaller number.

Test Data

| Inputs | Outputs |
|---|---|
| 20 , 3 | 6 |
| 16 , 4 | 4 |

Finished Code

# Advanced challenges

1. Ask the user for a big number, then a small number. Using only a BRP to loop round, keep subtracting the smaller number until you get past zero, then output the result.

Test Data

| Inputs | Outputs |
|---|---|
| 20 , 3 | -1 |
| 16 , 4 | -4 |

Finished Code

# Some ‘simple’ challenges

1. Ask the user for 3 numbers. Print them out in reverse order.

Test Data

| Inputs | Outputs |
|---|---|
| 7 , 8 , 9 | 9 , 8 , 7 |
| 8 , 16 , 32 | 32 , 16 , 8 |

Finished Code

# Some ‘simple’ challenges

2. Ask the user for 3 numbers. Add them up and print out the answer.

Test Data

| Inputs | Outputs |
|---|---|
| 7 , 8 , 9 | 24 |
| 8 , 16 , 32 | 56 |

Finished Code

# Loops!

Looping in LMC involves using one or more branches that repeats a set of instructions.

Predict what the following code will do:
(Hint: BRA means Branch Always)

```
INP
STA numOne
looptop ADD numOne
OUT
BRA looptop
HLT
numOne DAT
```

| Prediction |
| --- |
| |

Now try it out and see for yourself.

In order to make it better, we need an escape clause.

```
INP
STA bigNum
INP
STA littleNum
looptop LDA bigNum
SUB littleNum
STA bigNum
OUT
BRZ end
BRA looptop
end HLT
bigNum DAT
littleNum DAT
```

Use a **trace table** to follow this problem through.

Try it with 20 and 4 as the inputs

The BRZ is a conditional escape from the loop

`If the answer is zero, escape, otherwise keep looping.`

This is just like a WHILE loop.

# Intermediate challenges

3. Ask the user for 2 numbers, print out the result of the biggest number minus the smallest number

Test Data

| Inputs | Outputs |
|---|---|
| 7 , 3 | 4 |
| 5 , 12 | 7 |

Finished Code

# Some ‘simple’ challenges

3. Ask for 2 numbers.
   Print out the first - the second.
   Then the second - the first.

Test Data

| Inputs | Outputs |
|---|---|
| 7 , 3 | 4 , -4 |
| 5 , 12 | -7 , 7 |

Finished Code

# Phase 2 - branching

Branching allows you to take a program down 2 different paths.

There are 3 types of branch:

| Code | Meaning |
|---|---|
| BRZ | Branch if zero |
| BRP | Branch if positive (or zero) |
| BRA | Branch always (used for looping) |

# Intermediate challenges

2. Ask the user for 2 numbers. Print out biggest, then the smallest.

Test Data

| Inputs | Outputs |
|---|---|
| 12 , 15 | 15 , 12 |
| 7 , 2 | 7 , 2 |

Finished Code

# Intermediate challenges

1. Ask the user for 2 numbers. If they are the same then double the number and print it out. If they are different then print them both out individually.

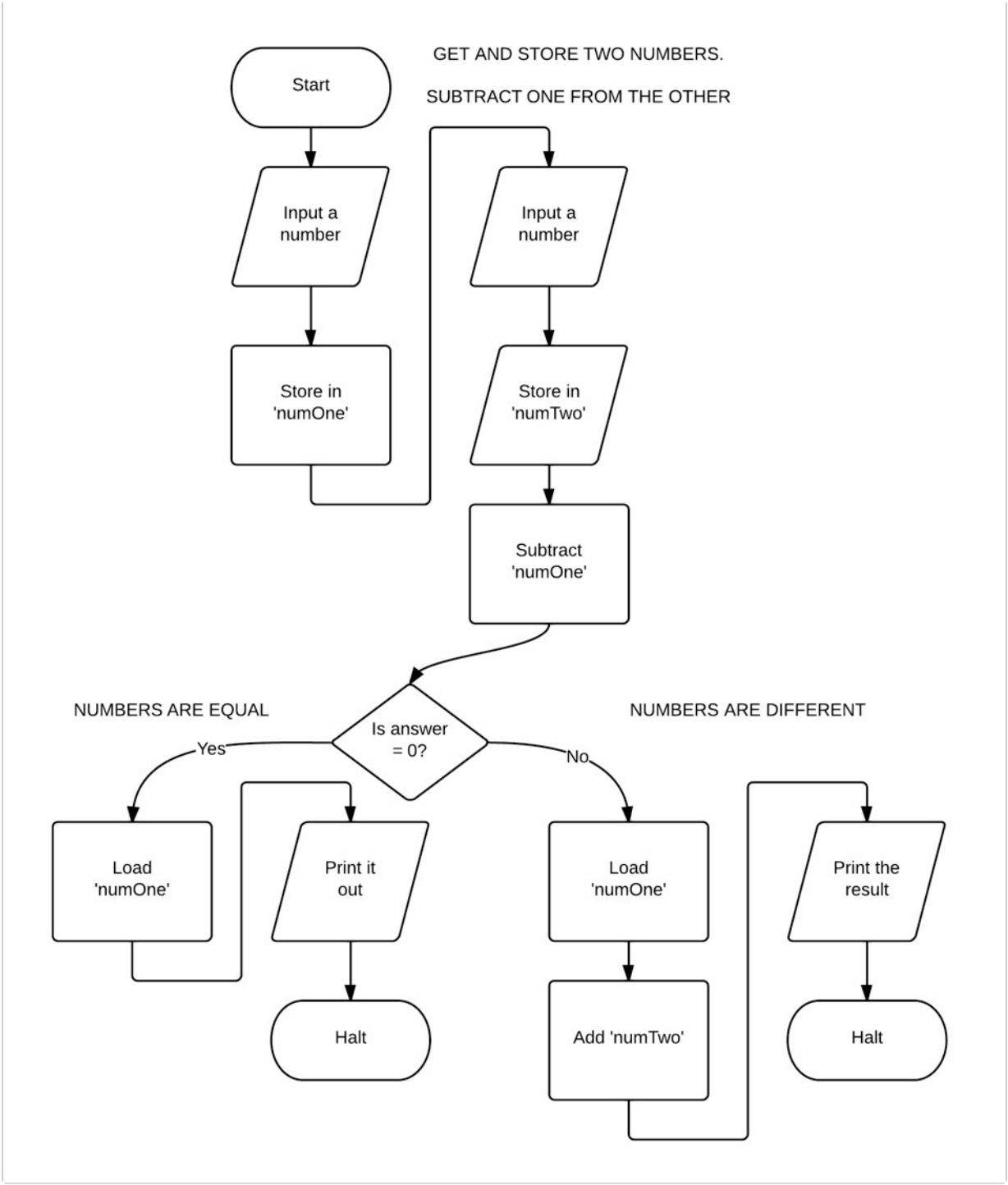
Test Data

| Inputs | Outputs |
|---|---|
| 15 , 15 | 30 |
| 12 , 9 | 12 , 9 |

Finished Code

# Branching Example

Branching allows you to take a program down 2 different paths.



# If Statements...

Human logic works like this:

```
If the two numbers are the same then print one of them out.
Otherwise, add them together and print the result.
```

We work through the positive result first, then the negative one. In LMC it doesn't work like that.

```
If the two number are the same then jump to 'same'.
Otherwise, add them together and print the result.
Same: Load the first number and print it out.
```

The easiest workflow is like this:

```
INP
STA numOne
INP
STA numTwo
SUB numOne
```

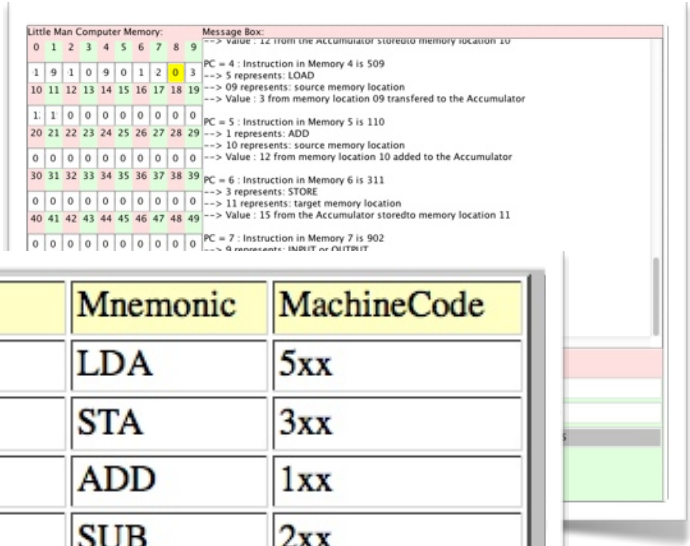
First, write your opening instructions.

In this case, input and store two numbers and then subtract them.

# ...continued

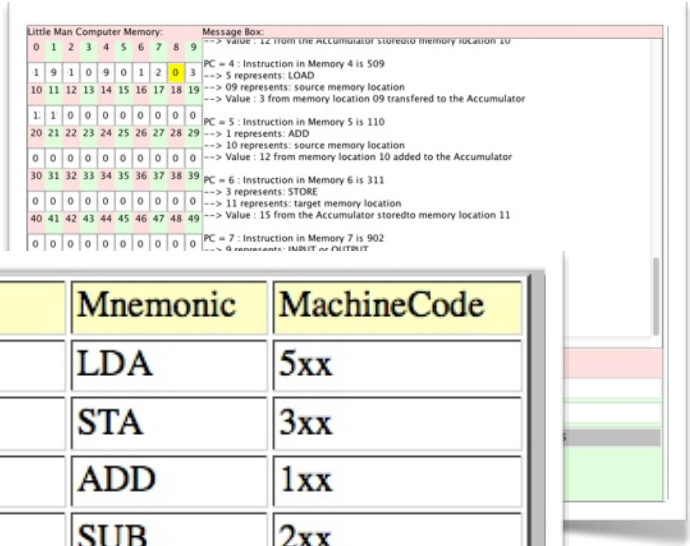
| if answer IS zero... | ...otherwise... | & declarations |
|---|---|---|
| INP<br>STA numOne<br>INP<br>STA numTwo<br>SUB numOne<br>BRZ same<br><br>same LDA numOne<br>OUT<br>HLT | INP<br>STA numOne<br>INP<br>STA numTwo<br>SUB numOne<br>BRZ same<br><br>LDA numOne<br>ADD numTwo<br>OUT<br>HLT<br><br>same LDA numOne<br>OUT<br>HLT | INP<br>STA numOne<br>INP<br>STA numTwo<br>SUB numOne<br>BRZ same<br><br>LDA numOne<br>ADD numTwo<br>OUT<br>HLT<br><br>same LDA numOne<br>OUT<br>HLT<br><br>numOne DAT<br>numTwo DAT |
| Add the branch if zero to the label 'same'.<br><br>Leave some empty space.<br><br>Then write the "same" instructions | Next, add in the "otherwise" instructions.<br><br>Remembering to include a HLT | Finally, add the DAT declarations at the very end (only once) |

# LMC Instruction Set




| Instruction | Mnemonic | MachineCode |
|---|---|---|
| Load | LDA | 5xx |
| Store | STA | 3xx |
| Add | ADD | 1xx |
| Subtract | SUB | 2xx |
| Input | INP | 901 |
| Output | OUT | 902 |
| End | HLT | 000 |
| Branch if zero | BRZ | 7xx |
| Branch if zero or positive | BRP | 8xx |
| Branch always | BRA | 6xx |
| Data storage | DAT | |

# LMC Instruction Set




| Instruction | Mnemonic | MachineCode |
|---|---|---|
| Load | LDA | 5xx |
| Store | STA | 3xx |
| Add | ADD | 1xx |
| Subtract | SUB | 2xx |
| Input | INP | 901 |
| Output | OUT | 902 |
| End | HLT | 000 |
| Branch if zero | BRZ | 7xx |
| Branch if zero or positive | BRP | 8xx |
| Branch always | BRA | 6xx |
| Data storage | DAT | |