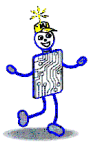


# CHAPTER 6: The Little Man Computer

## The Architecture of Computer Hardware, Systems Software & Networking:
### An Information Technology Approach

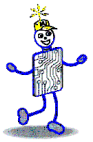
**5th Edition, Irv Englander**

**John Wiley and Sons ©2013**

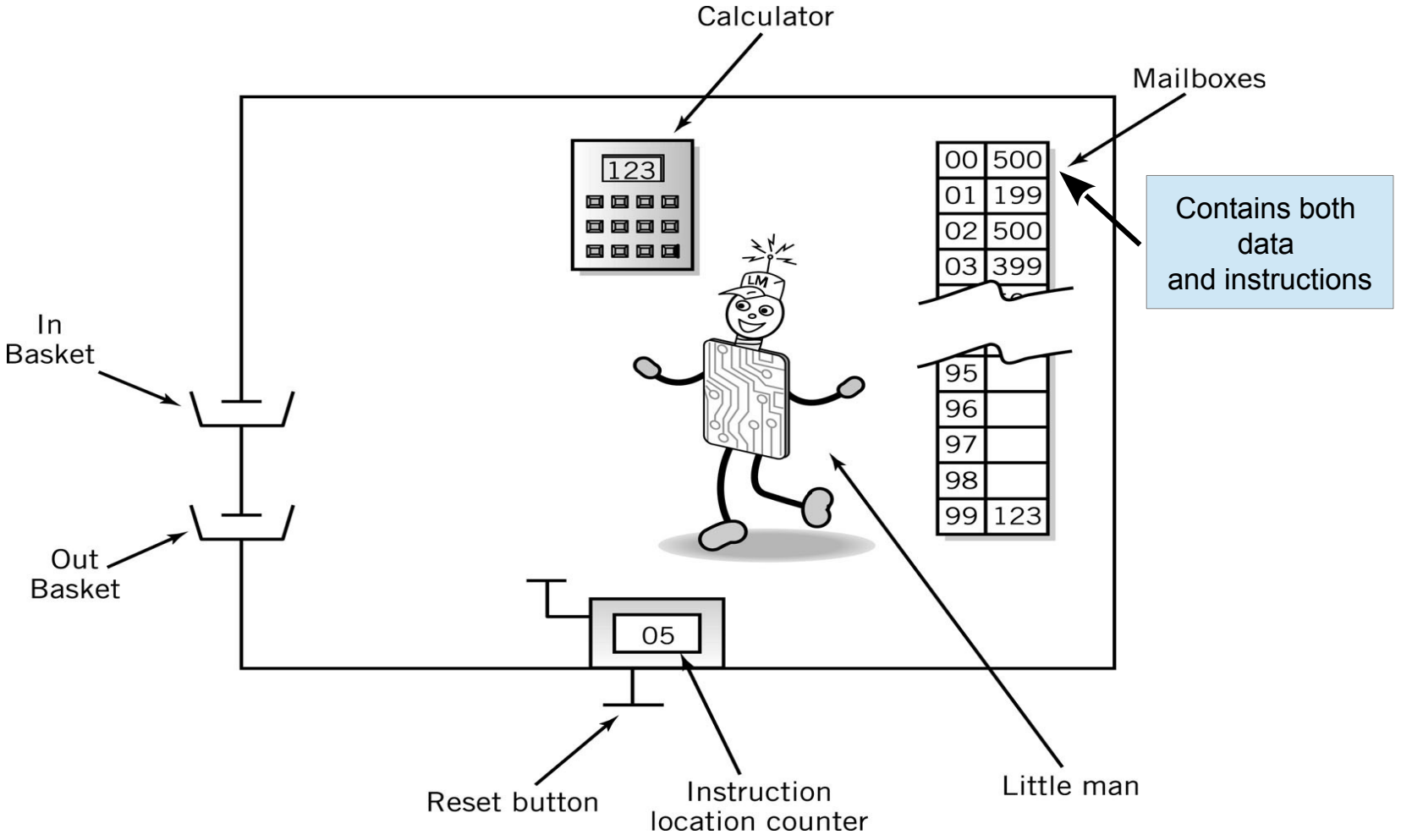
PowerPoint slides authored by Angela Clark, University of South Alabama

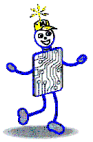
PowerPoint slides for the 4th edition were authored by Wilson Wong, Bentley University

PowerPoint slides modified by Gianluca Amato, Univ. di Chieti-Pescara



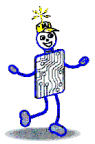
# The Little Man Computer





# CPU and Memory

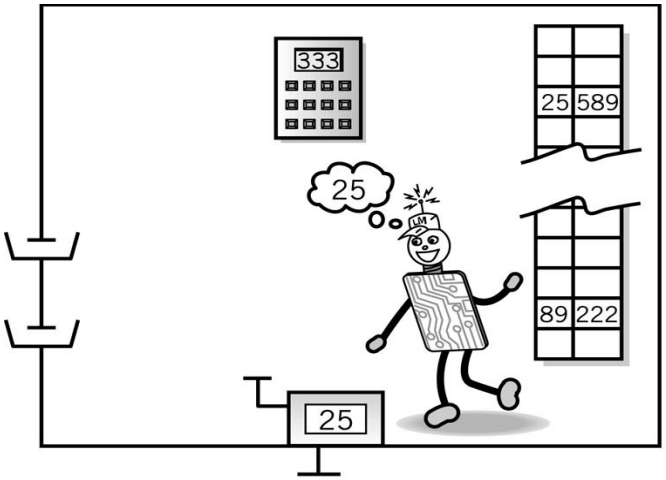
- Every instruction executed by the CPU requires memory access
- Primary memory holds program instructions and data
- Secondary storage is used for long term storage
  - Data is moved from secondary storage to primary memory for CPU execution

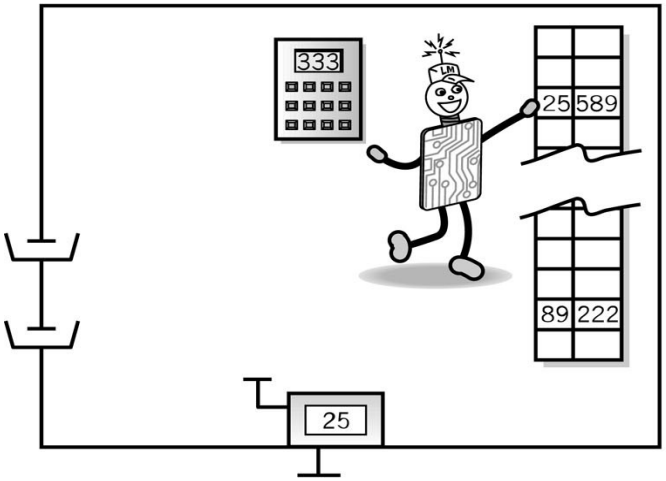


# Working of LMC

- *Instruction Cycle*
  - *Fetch*: Little Man finds out what instruction he is to execute
  - *Execute*: Little Man performs the work
    - depends from the instruction read

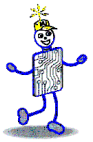
# Fetch Portion of Fetch and Execute Cycle



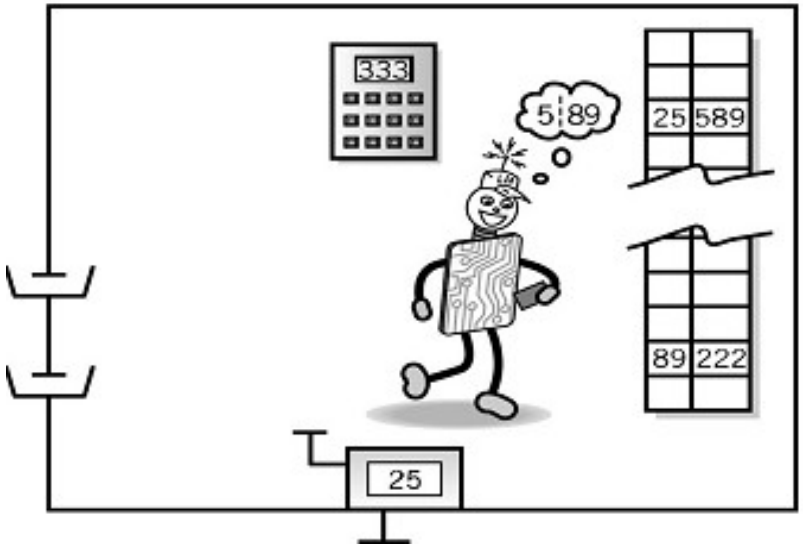




1. Little Man reads the address from the location counter

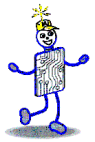
2. He walks over to the mailbox that corresponds to the location counter



# Fetch, cont.




3. And reads the number on the slip of paper (he puts the slip back in case he needs to read it again later)



# Coding of Instructions

- Op code
  - In LMC, represented by a single digit
  - Operation code
- Operand
  - In LMC, represented by two digits following the op code
  - Address of a memory location

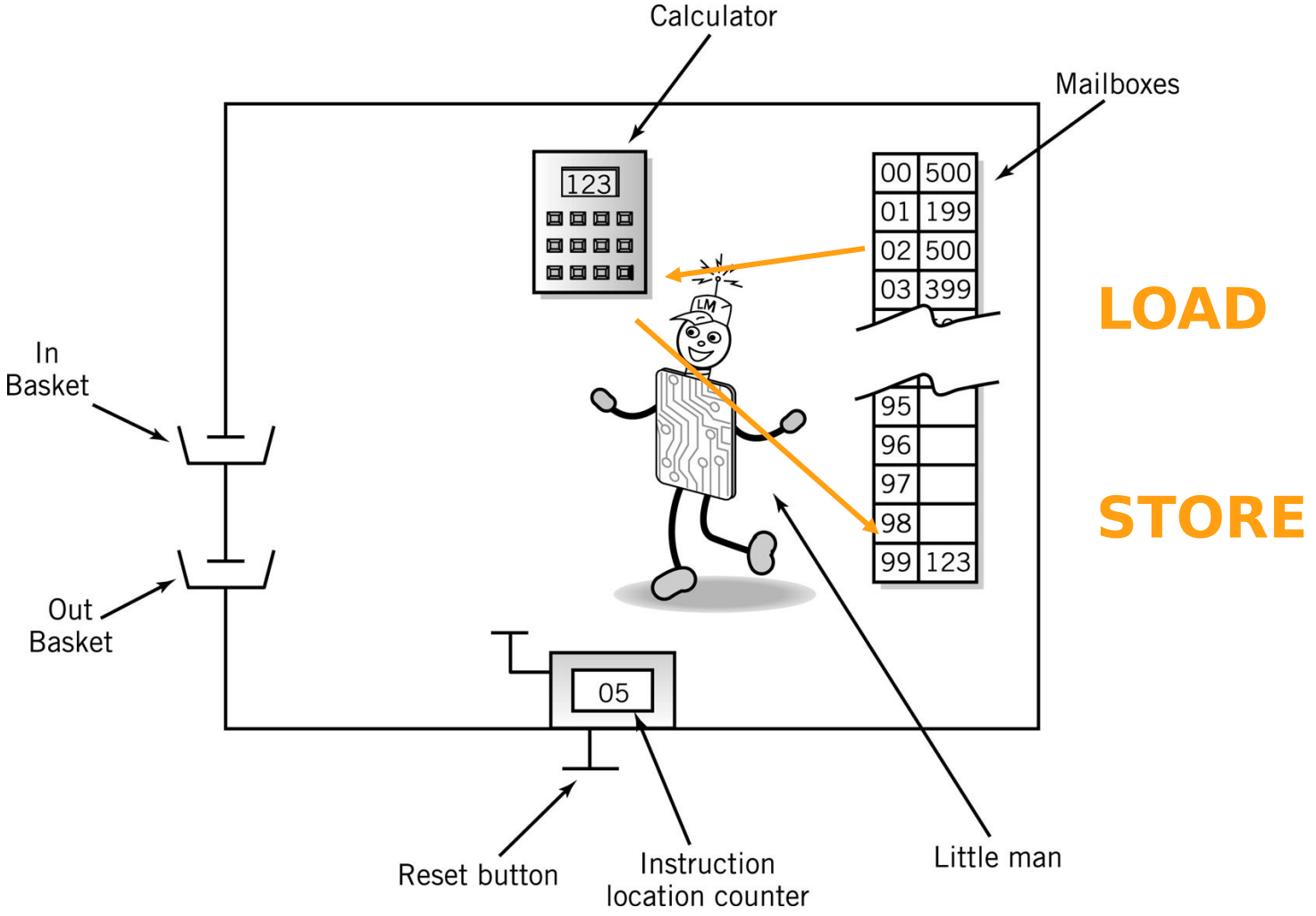
Instruction

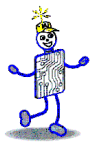
| Op code | Operand |
|---|---|
| 3 | 25 |

# Instruction Set

| | | |
|---|---|---|
| Arithmetic | 1xx | ADD |
| | 2xx | SUBTRACT |
| Data Movement | 3xx | STORE |
| | 5xx | LOAD |
| Input/Output | 901 | INPUT |
| | 902 | OUTPUT |
| Machine Control | 000 | HALT |

# LMC Internal Data Movement



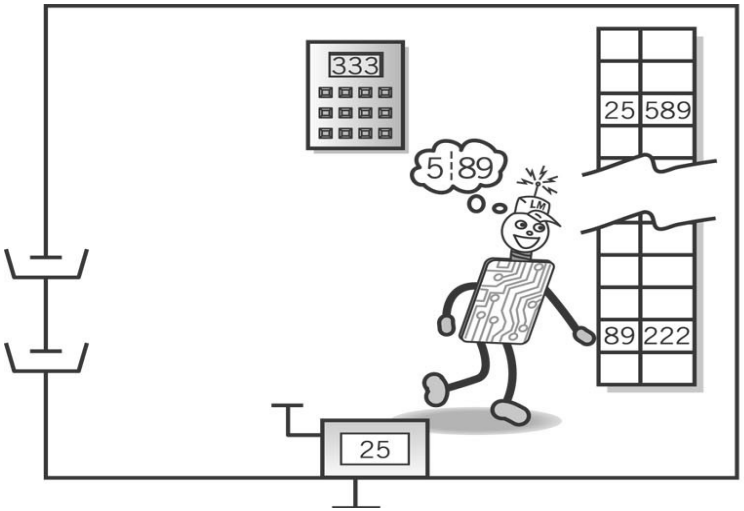


# Internal Data Movement

- Between mailbox and calculator

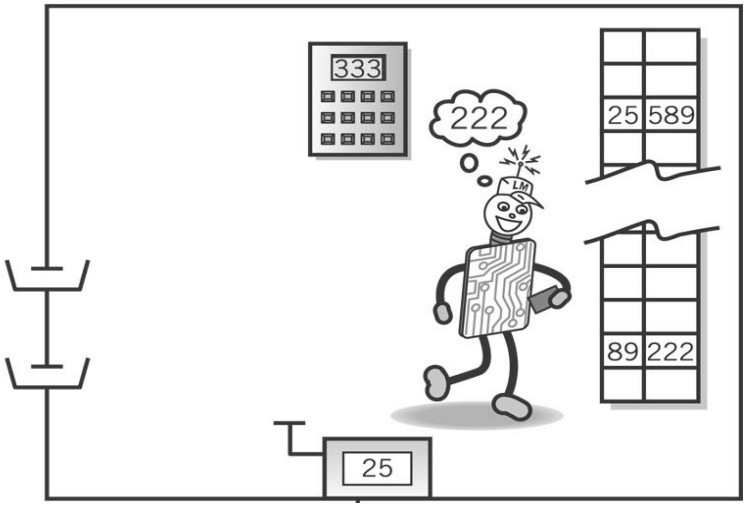
| | Content | |
|---|---|---|
| | Op Code | Operand (address) |
| STA (store) | 3 | xx |
| LDA (load) | 5 | xx |

# Execute Portion

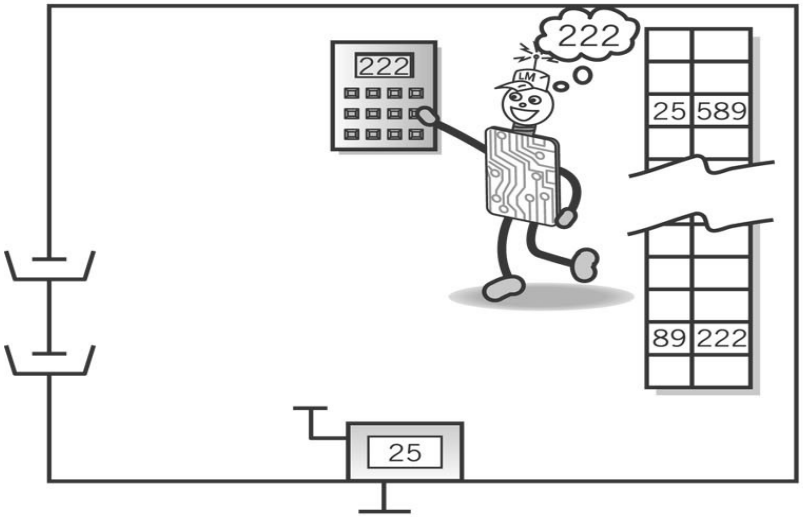



1. The Little Man goes to the mailbox address specified in the instruction he just fetched.

2. He reads the number in that mailbox (he remembers to replace it in case he needs it later).

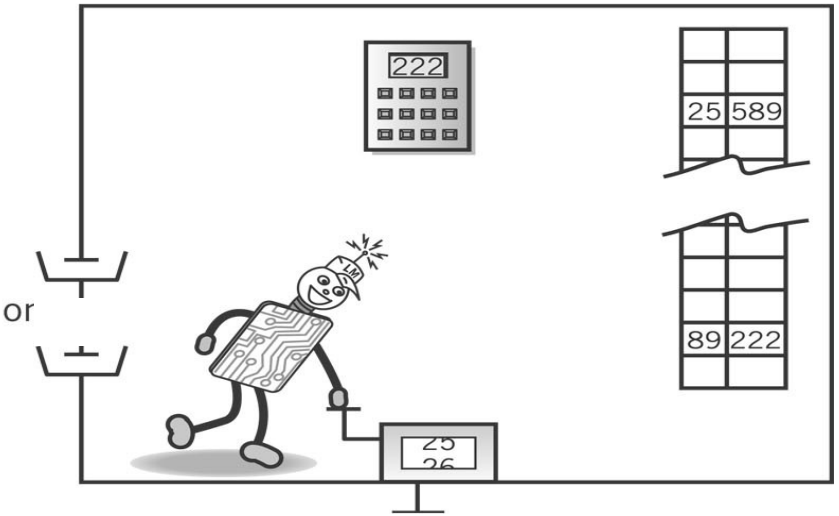


# Execute, cont.

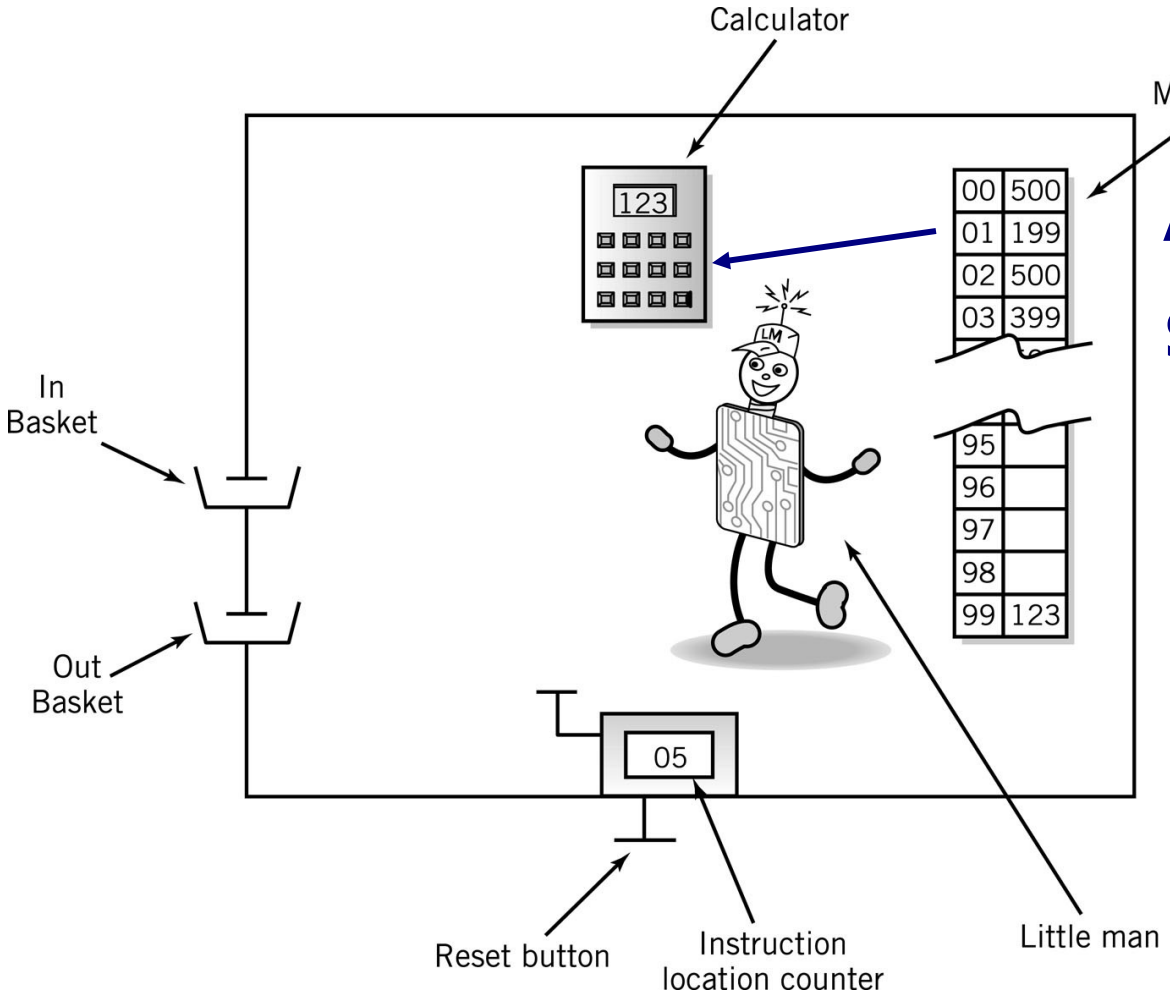



3. He walks over to the calculator and punches the number in.

4. He walks over to the location counter and clicks it, which gets him ready to fetch the next instruction.

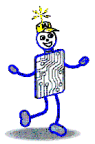


# LMC Arithmetic Instructions




**ADD**

**SUBTRACT**

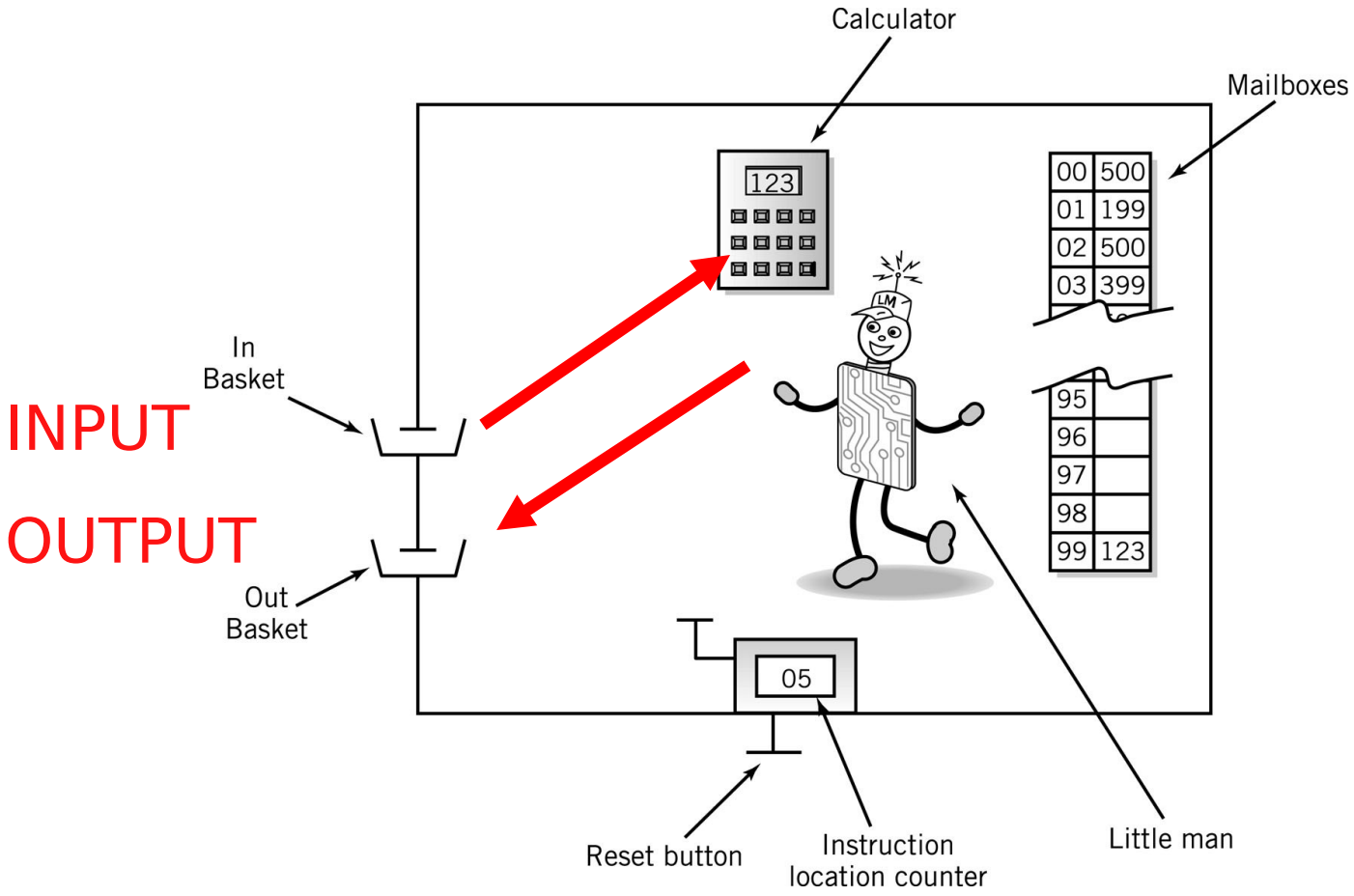


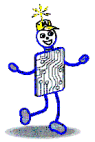
# Arithmetic Instructions

- Read mailbox
- Perform operation in the calculator

| | Content | |
|---|---|---|
| | Op Code | Operand (address) |
| ADD (add) | 1 | xx |
| SUB (subtract) | 2 | xx |

# LMC Input/Output

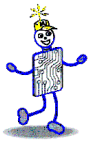




# Input/Output

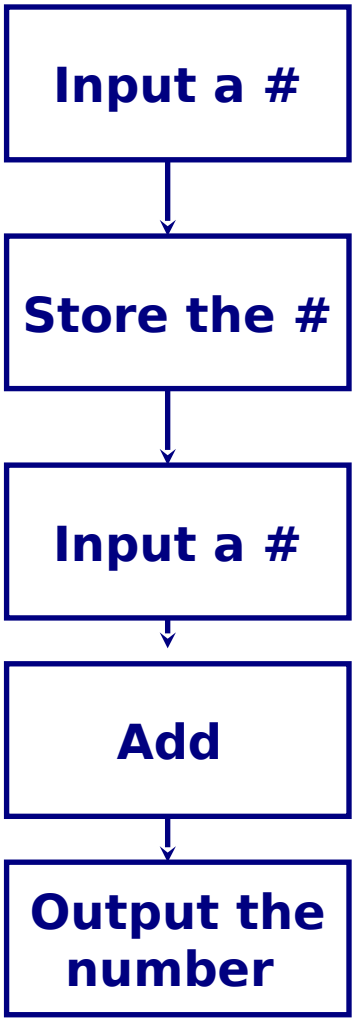
- Move data between calculator and in/out baskets

| | Content | |
|---|---|---|
| | Op Code | Operand (address) |
| INP (input) | 9 | 01 |
| OUT (output) | 9 | 02 |



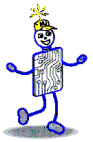
# Simple Program: Add 2 Numbers

- Assume data is stored in mailboxes with addresses >90
- Write instructions



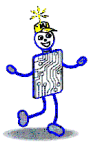
# Program to Add 2 Numbers

| Mailbox | Code | Instruction Description |
|---|---|---|
| 00 | 901 | ;input 1st Number |
| 01 | 399 | ;store data |
| 02 | 901 | ;input 2nd Number |
| 03 | 199 | ;add 1st # to 2nd # |
| 04 | 902 | ;output result |
| 05 | 000 | ;stop |
| 99 | 000 | ;data |



# Assembly Language

- *Mnemonics* (short character sequence) represent instructions
- 1 to 1 correspondence between assembly language instruction and binary (machine) language instruction
- specific to a CPU



# Program to Add 2 Numbers: Using Mnemonics

| Mailbox | Mnemonic | Instruction Description |
|---|---|---|
| 00 | IN | ;input 1st Number |
| 01 | STO 99 | ;store data |
| 02 | IN | ;input 2nd Number |
| 03 | ADD 99 | ;add 1st # to 2nd # |
| 04 | OUT | ;output result |
| 05 | HLT | ;stop |
| 99 | 000 | ;data |

# Program Control

- Branching (executing an instruction out of sequence)
  - Changes the address in the counter

BRA (Jump)

BRZ (Branch on 0)

BRP (Branch on +)

| | Content | |
|---|---|---|
| | Op Code | Operand (address) |
| BRA (Jump) | 6 | xx |
| BRZ (Branch on 0) | 7 | xx |
| BRP (Branch on +) | 8 | xx |

# Revised Instruction Set

| | | |
|---|---|---|
| Arithmetic | 1xx | ADD |
| | 2xx | SUBTRACT |
| Data Movement | 3xx | STORE |
| | 5xx | LOAD |
| BRA | 6xx | JUMP |
| BRZ | 7xx | BRANCH ON 0 |
| BRP | 8xx | BRANCH ON + |
| Input/Output | 901 | INPUT |
| | 902 | OUTPUT |
| Machine Control | 000 | HALT |

# Find Positive Difference of 2 Numbers

| | | | |
|---|---|---|---|
| 00 | INP | 901 | |
| 01 | STA 10 | 310 | |
| 02 | INP | 901 | |
| 03 | STA 11 | 311 | |
| 04 | SUB 10 | 210 | |
| 05 | BRP 08 | 808 | ;test |
| 06 | LDA 10 | 510 | ;if negative, reverse order |
| 07 | SUB 11 | 211 | |
| 08 | OUT | 902 | ;print result and |
| 09 | HLT | 000 | ;stop |
| 10 | --- | 000 | ;used for data |
| 11 | --- | 000 | ;used for data |

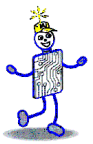
# Execution trace

```
        INP
        STA :tmp
        INP
        SUB :tmp
        BRZ :salta
salta:  LDA :one
        OUT
        HALT
tmp:    000
one:    001
```

| PC prima | PC dopo | A | tmp | one |
|---|---|---|---|---|
| -- | 0 | 0 | 0 | 1 |
| 0 | 1 | 23 | 0 | 1 |
| 1 | 2 | 23 | 23 | 1 |
| 2 | 3 | 12 | 23 | 1 |
| 3 | 4 | -9 | 23 | 1 |
| 4 | 5 | -9 | 23 | 1 |
| 5 | 6 | 1 | 23 | 1 |
| 6 | 7 | 1 | 23 | 1 |
| 7 | -- | 1 | 23 | 1 |

Inputs: 23 12

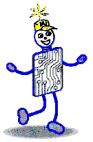
Outputs: 1



# Program with a cycle

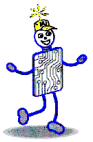
Given an input *n*, returns 0 + 1 + 2+ …. *n*

```
            INP
ciclo:      STA :count
            LDA :somma
            ADD :count
            STA :somma
            LDA :count
            SUB :one
            BRZ :fine
            BRA :ciclo
fine:       LDA :somma
            OUT
            HALT
somma:      000
count:      000
one:        001
```



# von Neumann Architecture (1945)

- Stored program concept
- Memory is addressed linearly
- Memory is addressed without regard to content



# Copyright 2013 John Wiley & Sons