The Little Man Computer

The Little Man Computer - an instructional model of
von Neuman computer architecture

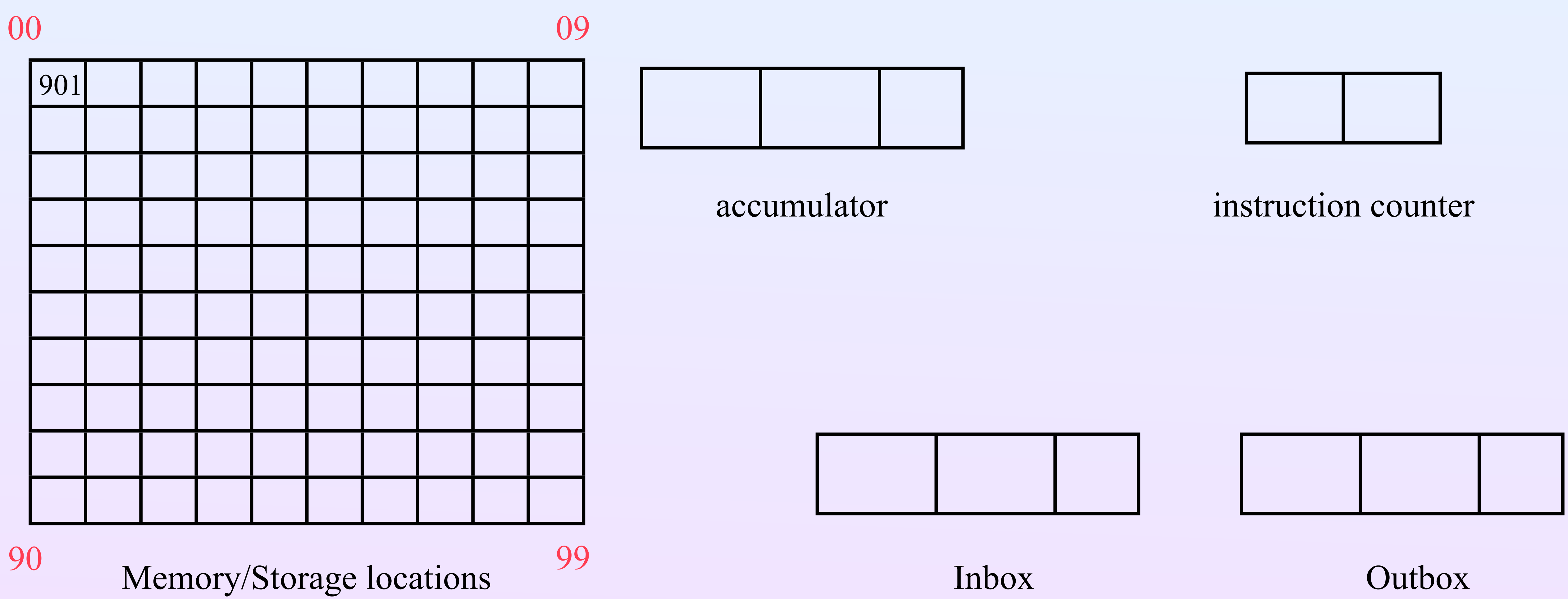
John von Neuman (1903-1957) and Alan Turing (1912-1954) each independently laid foundation for today's computers - *the stored program computer*

[von Neuman]

[Turing]

Components of Little Man

- 100 storage locations indexed 0 thru 99 - each can store a 3 digit integer
- a unique 3 digit storage location called *calculator* or *accumulator*
- a 2 digit *instruction location counter*
- an inbox that can contain a 3 digit number
- an outbox that can contain a 3 digit number



A Little Man Program

Consists of *instructions* placed in memory starting at position 00.

An instruction is a 3 digit integer

- left digit an *operation code* 0 to 9 - telling what type of action to take
- right 2 digits 0 to 99 indicate memory position

Instructions are of various sorts :

- some take information from inbox and place in the accumulator - such information called *input*
- some take information in the accumulator and place in outbox - such information called *output*
- some interact with information in accumulator

Little Man Operations

operation	mnemonic	code	description
Input	INP	901	info in inbox ⟶ accumulator
Output	OUT	902	info in accumulator ⟶ outbox
Store	STA	3xx	info in accumulator ⟶ location xx
Load	LDA	5xx	info in location xx ⟶ accumulator
Add	ADD	1xx	info in location xx added to info in accumulator
Subtract	SUB	2xx	info in location xx subtracted from info in accumulator
Branch	BRA	6xx	reset program location indicator to location xx
Branch if 0	BRZ	7xx	if info in accumulator = 000, reset program location to xx
Branch if ≥ 0	BRP	8xx	if info in accumulator ≥ 0, reset program location to xx
Halt	END	0	stop program execution
Data definition	DAT		used to define memory locations for storing data

Construction of Little Man Program

- The *source code* of a little man program is a list of mnemonic instructions.
- The collection of mnemonic instructions constitute the *assembly language* for Little Man programs
- The source code is *compiled* by a background program called the *compiler*
- The compiler places equivalent numeric 3 digit instruction codes sequentially in memory -
  - could be done manually and must be done manually without computer implementation of the Little Man

Running a Little Man program

- Once compiled (an operation external to the program), the program is started by another operation external to the program called *run*

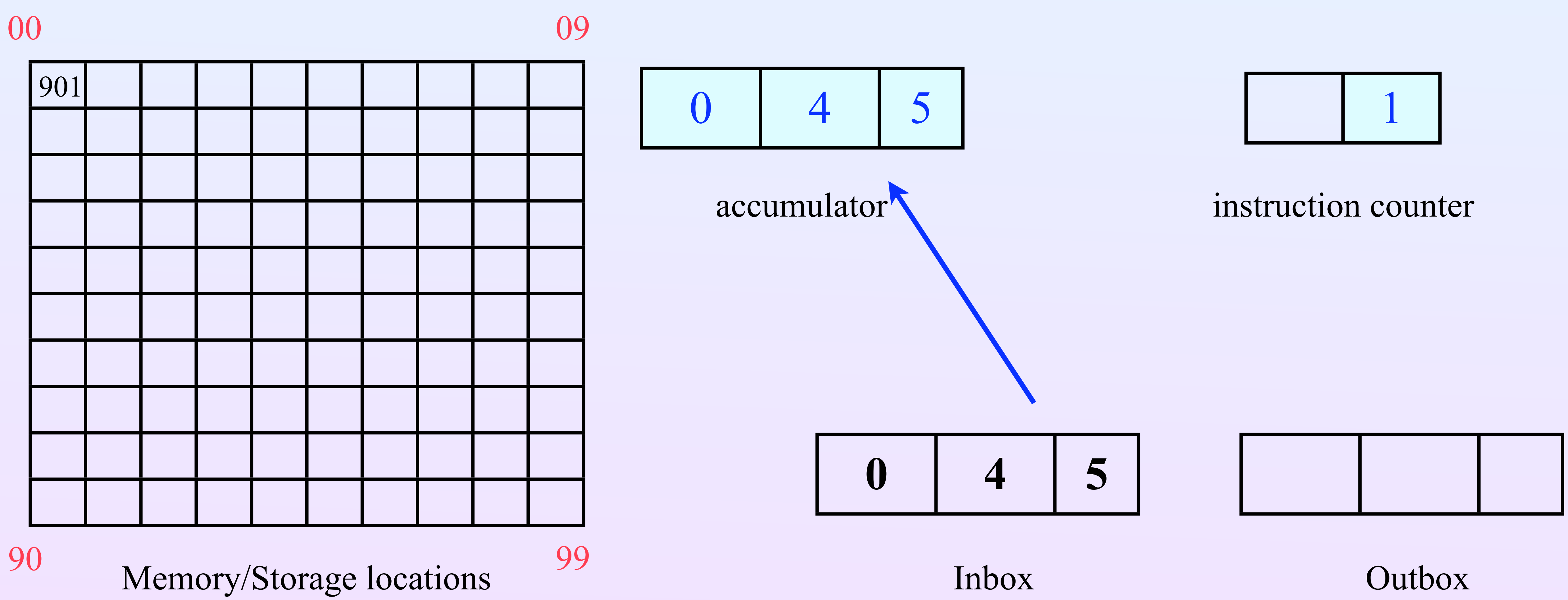
- Once started the first instruction in location 00 is *executed* and the instruction location counter is incremented from 0 to 1

  - The first Little Man instruction is often (but not always) an Input instruction -
    an outside agent puts information in the inbox for the input instruction to execute

Input operation (INP)

The Input operation here - with 901 in location 00

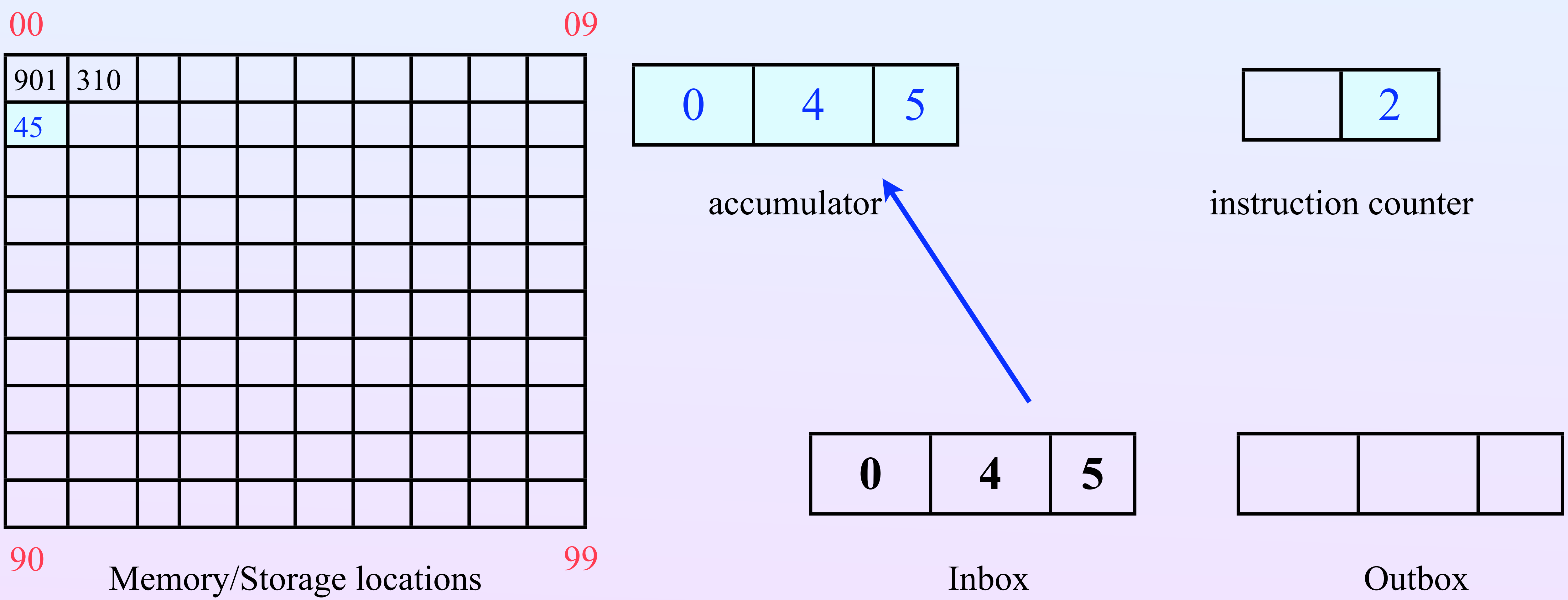
- takes 045 in the Inbox and places it in the accumulator
- increments the instruction counter from 0 to 1



Store operation (STA)

The Store operation here - with 310 in location 01

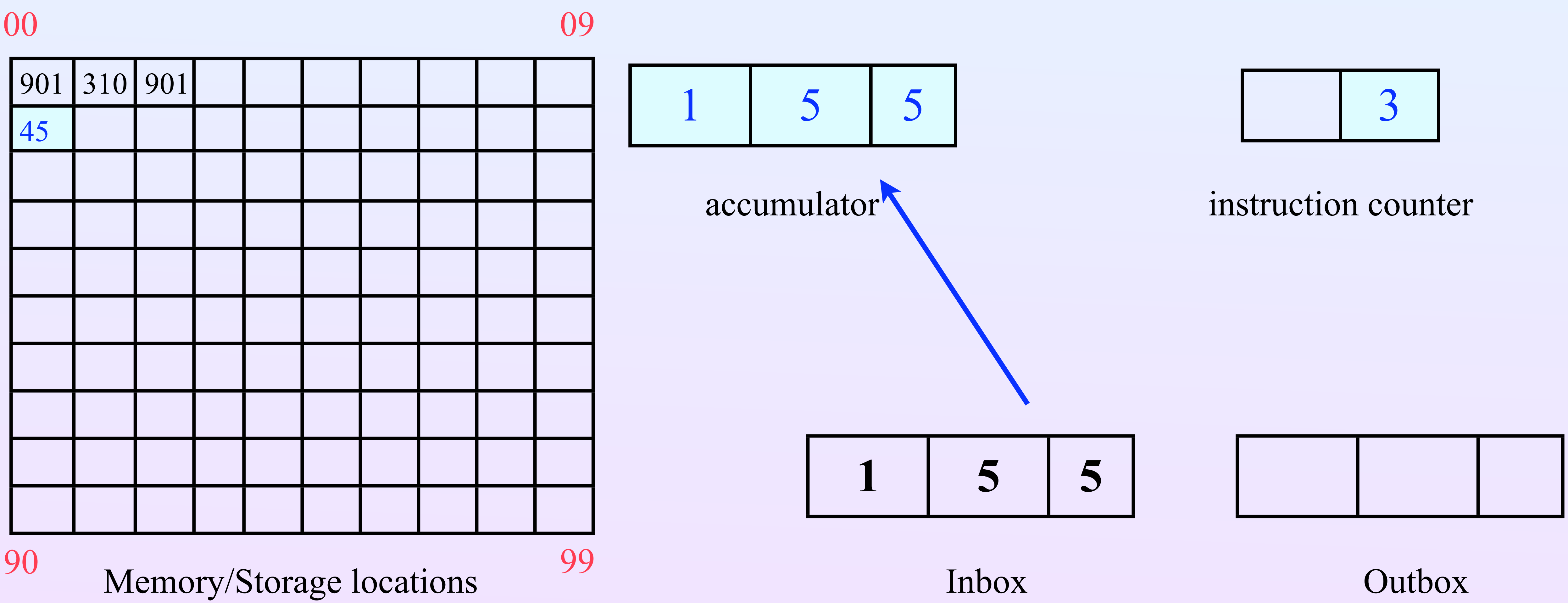
- takes 45 in the accumulator and places it in memory location 10
- increments the instruction counter from 1 to 2
- the accumulator retains its value



A second Input operation

A second input operation here - with 901 in location 02

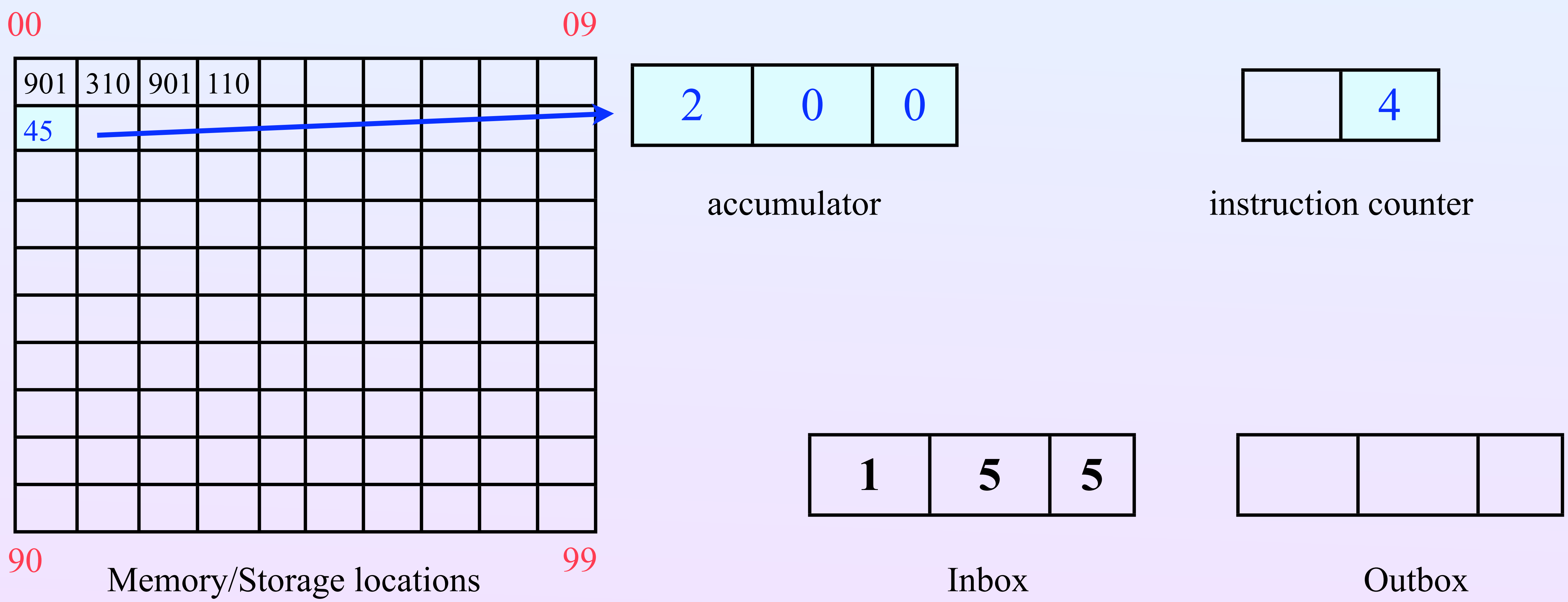
- waits for input
- takes new input 155 and put it in the accumulator
- increments the instruction counter from 2 to 3



Add operation (ADD)

An add operation is performed with a 110 in memory location 03

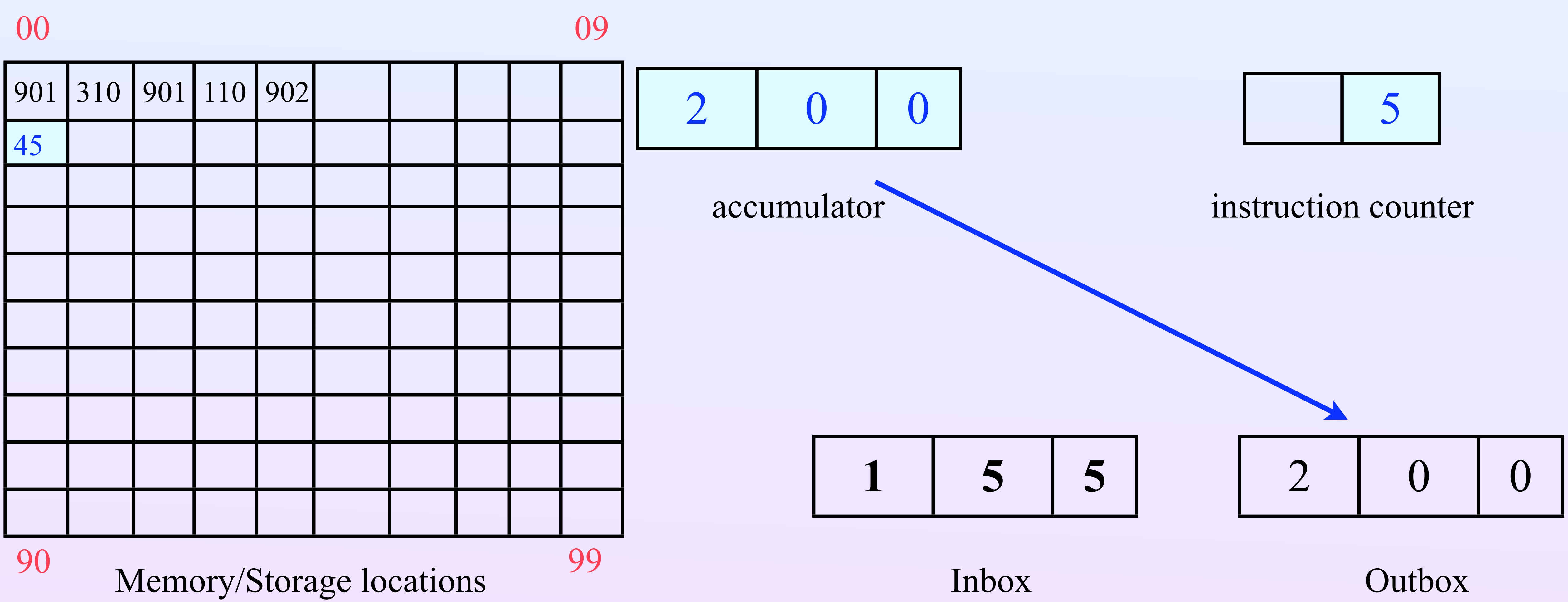
- the right 2 digits signify that the contents of location 10 are added to the accumulator giving value 200
- the instruction counter from 3 to 4



Out operation (OUT)

The Out operation is accomplished here with a 902 in location 05

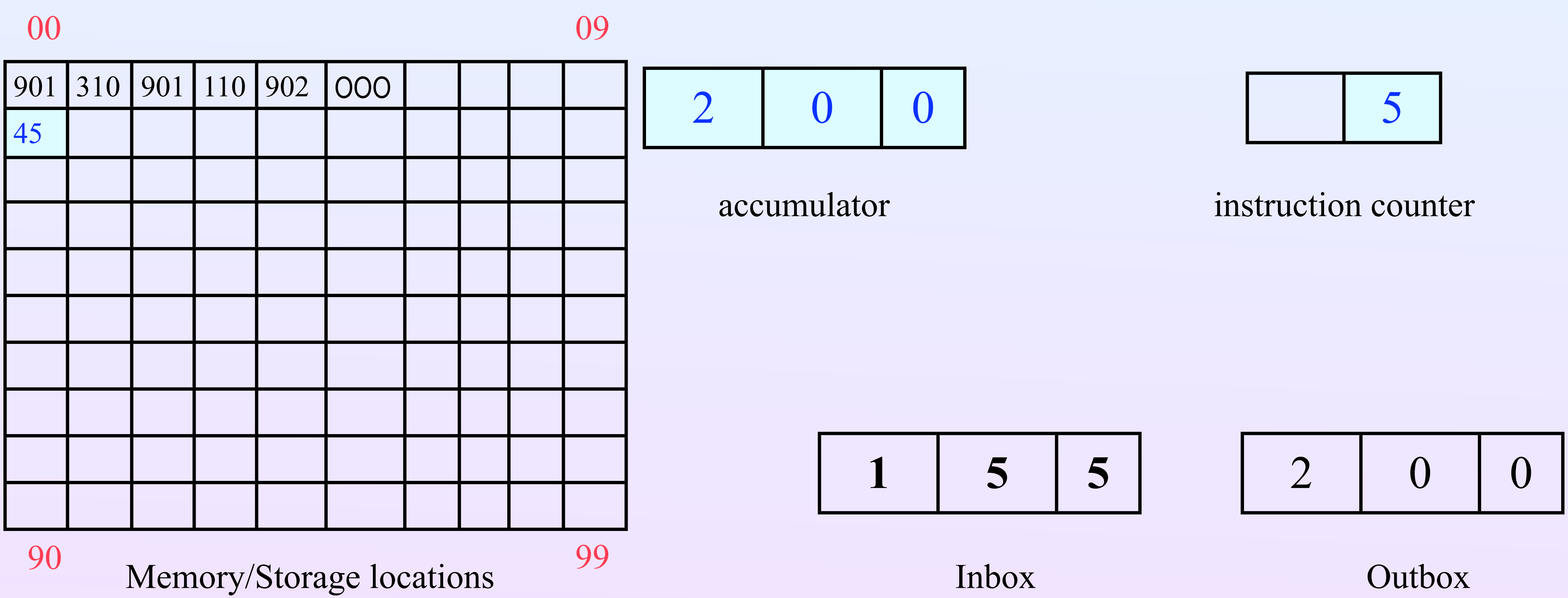
- the contents accumulator is put in the outbox - contents of accumulator not changed
- the instruction counter is incremented from 4 to 5



HALT operation

The Halt operation is accomplished here with a 000 in location 06.

It signifies the end of the program



The sequence of instructions constructed was:

901	310	910	110	902	000

constitutes a Little Man program whose function adds two numbers

We call above the *machine code* of the Little Man program.

Corresponding *assembly code*

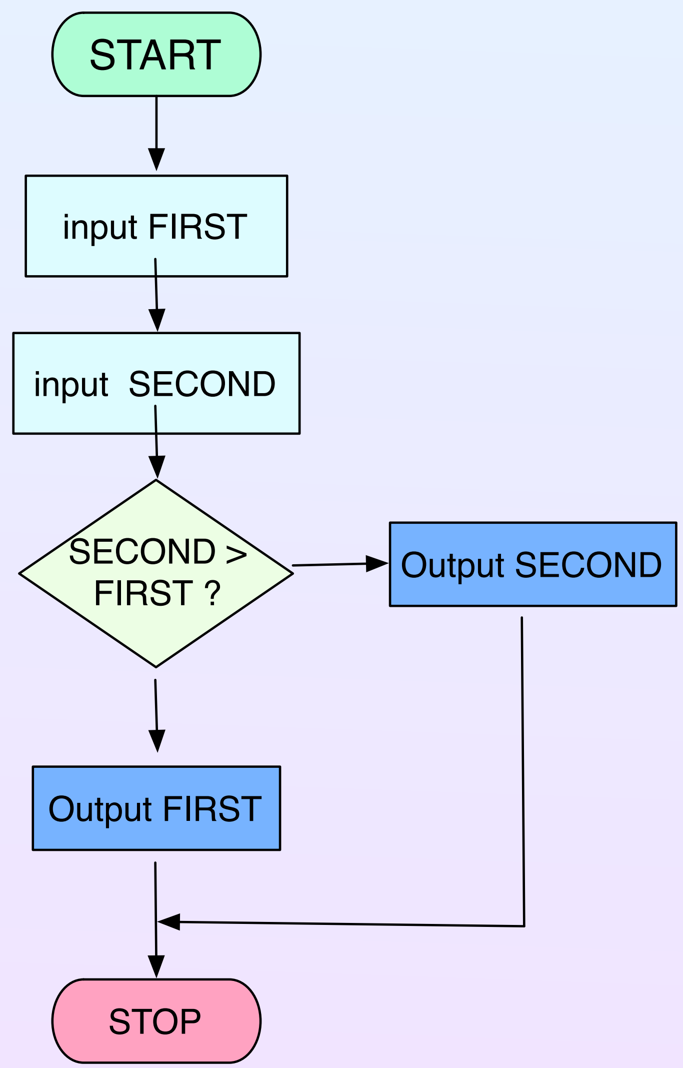
	INP	
	STA	FIRST
	INP	
	ADD	FIRST
	OUT	
	END	
FIRST	DAT	

Identifies FIRST as data - the compiler then allocates a memory position called FIRST. Our program has FIRST = 10. The "label" FIRST identifies a position in memory. It happens to be the 10th postion.

Branching and Labels

Branching instructions allow program to execute some instructions under one condition and other instructions under other conditions

Example: diagram of decision process to determine greater of 2 numbers



To implement example in source code using Little Man mnemonics - there is notion of the *label* of an instruction to be branched to or the *label* of a data storage position

The assembly code with explanation for the example is:

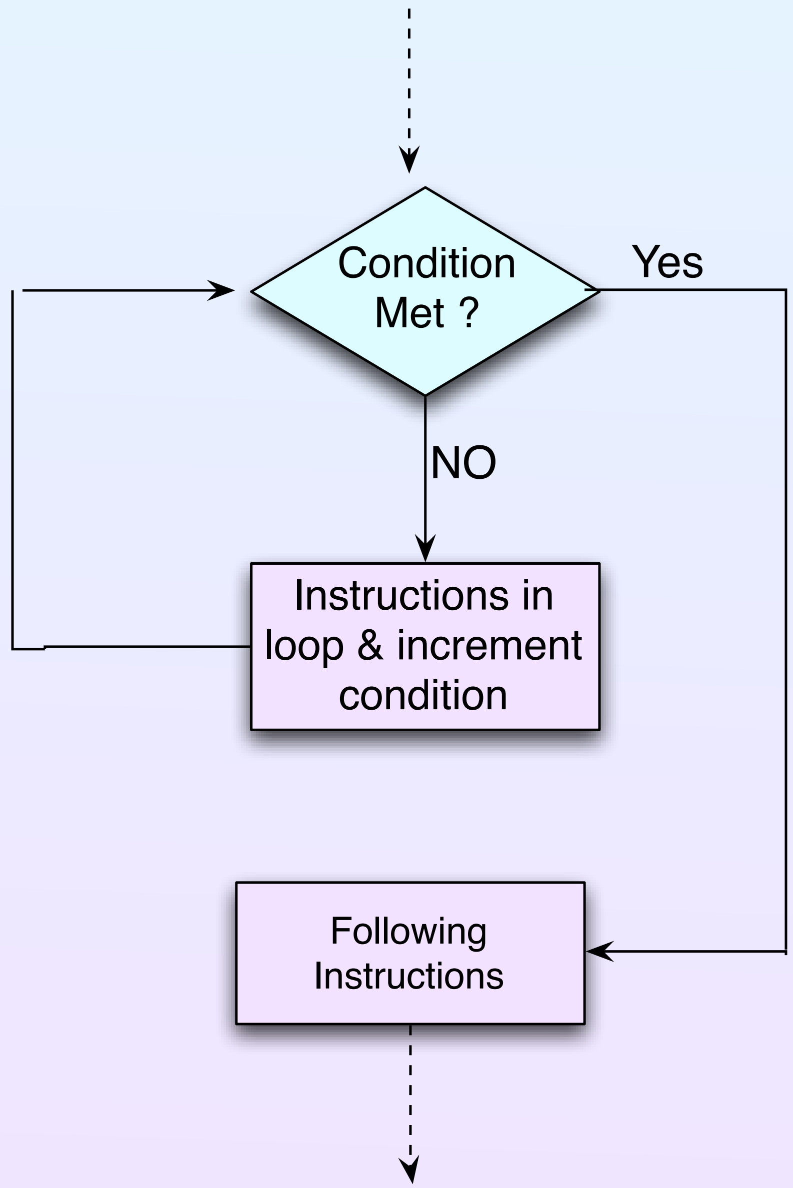
LABEL	CODE	ARGUMENT	DESCRIPTION
	INP		input data = x to accumulator
	STA	FIRST	place accumulator data x in memory location defined by FIRST
	INP		input data = y to accumulator
	STA	SECOND	place accumulator data y in location defined by SECOND
	SUB	FIRST	subtract data in First from data in accumulator i.e. y - x
	BRP	SEC_BIG	if accumulator data (y-x) ≥ 0, branch to SEC_BIG instruction
	LDA	FIRST	otherwise x < y, so load the value x in FIRST to accumulator
	OUT		output the value x
	BRA	END_PROG	go to the end instruction
SEC_BIG	LDA	SECOND	here y ≥ x - so load value y in SECOND to accumulator
	OUT		output the value y
END_PROG	HLT		end the program
FIRST	DAT		declare the data region labeled FIRST
SECOND	DAT		declare the data region labeled SECOND

The assembly code with corresponding machine code in right column

INST #	LABEL	CODE	ARGUMENT	MACHINE CODE
0		INP		901
1		STA	FIRST	312
2		INP		901
3		STA	SECOND	313
4		SUB	FIRST	212
5		BRP	SEC_BIG	809
6		LDA	FIRST	512
7		OUT		902
8		BRA	END_PROG	611
9	SEC_BIG	LDA	SECOND	513
10		OUT		902
11	END_PROG	HLT		000
12	FIRST	DAT		
13	SECOND	DAT		

Looping

A loop is a fixed group of instructions which must be executed more than once dependent on a changing condition



Looping Example - Little Man program that has output every odd number $\leq$ 99

INST #	LABEL	CODE	ARG	MACHINE CODE	
0		INP		901	input first odd number = 1
1	LOOP	SUB	NUM	210	subtract 99 from value in accumulator
2		BRP	END	807	if (value of accumulator - 99) $\geq$ 0, finished so go to END
3		LDA	ODD	508	restore value of ODD to accumulator - was destroyed in #1
4		OUT		902	output value of accumulator = value of ODD
5		ADD	TWO	109	add 2 to value of accumulator getting ODD + 2
6		BRA	LOOP	601	Branch to instruction #1 - accumulator already set for next
7	END	HALT		000	
8	TWO	DAT	2		
9	ODD	DAT	1		Note that these 3 data declarations also declare values
10	NUM	DAT	99		

A second version of previous example -

at instruction #3 subtraction gives positive numbers
except when ODD = 101

INST #	LABEL	CODE	ARG	MACHINE CODE	
0		INP		901	input first odd number = 1
1	LOOP	STA	ODD	311	put value of accumulator in ODD
2		LDA	NUM	512	put 99 in accumulator
3		SUB	ODD	211	subtract ODD from 99
4		BRP	MORE	806	if value of accumulator ≥ 0 branch to MORE
5	END	HLT			otherwise stop
6	MORE	LDA	ODD	511	put value of ODD in accumulator
7		OUT		902	output value of ODD
8		ADD	TWO	110	add 2 to value of accumulator - so it contains now ODD + 2
9		BRA	LOOP	601	Branch to LOOP
10	TWO	DAT	2		
11	ODD	DAT	1		Note that these 3 data declarations also declare values
12	NUM	DAT	99		

Playing with the Chen Little Man compiler

My colleague Stephen Chen and his associate W.C.Cudmore have constructed a simulation of a Little Man compiler.

Given Little Man assembly source code written with Little Man mnemonics, machine code is produced which can be executed

Little Man Computer simulation

Below are the two versions from the previous two slides - paste them into the simulation and run them. What is wrong with the first?

Version 1

```
INP
LOOP SUB NUM
BRP END
LDA ODD
OUT
ADD TWO
BRA LOOP
END HLT
TWO DAT 2
ODD DAT 1
NUM DAT 99
```

Version 2

```
INP
LOOP STA ODD
LDA NUM
SUB ODD
BRP MORE
END HLT
MORE LDA ODD
OUT
ADD TWO
BRA LOOP
TWO DAT 2
ODD DAT 1
NUM DAT 99
```