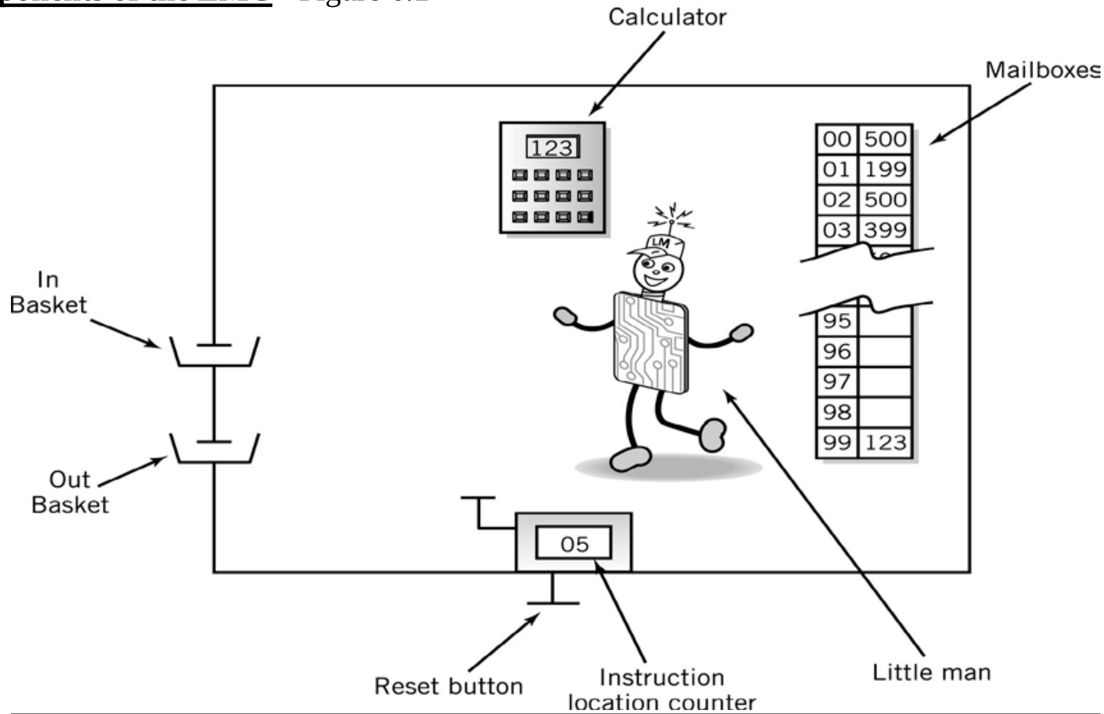
CIS-350
INFRASTRUCTURE TECHNOLOGIES

The Little Man Computer (LMC) - Chapter 6

Read chapter 6.

LMC - a hypothetical computer proposed in 1965
- its operation is very similar to a real computer

Components of the LMC - Figure 6.1



An arriving item is put in the In-basket for the Little Man (LM) to process - (read operation)

A departing item (result of calculations) is put by the LM in the Out-basket - (write operation)

The Little Man (acts as the Control Unit).

Through in-basket and out-basket the LMC communicates with the external world.

100 Mailboxes - (memory locations)
Each mailbox has:
- a 2-digit address starting with a 00 and ending with 99 (decimal) - 100 addresses
- the contents - a 3-digit #

The contents of the cell and its address are not the same.

Instruction location counter (program counter or instruction counter)
- can be increased (altered) by the LM typically by 1 or reset from outside

Calculator (ALU) - performs computations, can handle up to 3 digits

Instruction set of the LMC

To perform useful work, the LMC should be equipped with the instruction set.

7 basic instructions - are represented by digits.

(Real computer has about 60-150 instructions in the instruction set.)

An example of the instruction: 503

5	03
first digit	two other digits
represents an operation code or opcode that tells the LM what to do	represent an operand, typically the mailbox # or address. An operand identifies memory address on which the opcode acts

opcode	full name	**mnemonic**/abbreviation	
5	LOAD	**LDA**	
3	STORE	**STO**	
1	ADD	**ADD**	
2	SUBTRACT	**SUB**	
9	INPUT	**IN**	(901)
9	OUTPUT	**OUT**	(902)
0	HALT	**HLT**	

We write instructions in a mnemonic form instead of numeric codes because the instructions are more readable.

Examples (will be worked step by step on the white board in class)

503 LOAD 03 LDA 03

The computer program that the LMC executes is stored in mailboxes (memory) in the program area. The program acts on data. Data is also stored in memory (in the area different from the program). This area is called the data area.

Ex. The program below reads in 2 numbers, adds them, and outputs the result.

instru-ction #	**address** at which instruction is stored	the instruction itself,	**mnemonic form**	numeric code	
1	**00**	INPUT	**IN**	901	
2	**01**	STORE 50	**STO 50**	350	
3	**02**	INPUT	**IN**	901	**program area**
4	**03**	ADD 50	**ADD 50**	150	
5	**04**	OUTPUT	**OUT**	902	
6	**05**	HALT	**HLT**	000	
…………					
…………					
	50				**data area**

The instructions are executed sequentially starting at address 00. The LMC will execute them in the sequential order, one instruction at a time. How does the LMC know which instruction to execute? The instruction location counter will tell him. The instruction location counter stores the address of the next instruction to execute.

What you see above is assembly language

- Specific to a CPU
- 1 to 1 correspondence between assembly language instruction and binary (machine) language instruction
- *Mnemonics* (short character sequence) represent instructions
- Used when programmer needs precise control over hardware, e.g., device drivers

We will trace the execution of the program above, step by step, on the white board in class.

Extended instruction set

The LMC needs several more instructions that allow to:

- branch
- implement a loop

<u>branch</u> (*if-then*)

```
int a=2, flag;
if (a>2)
        flag=0;
else
        flag=1;
```

<u>loop</u> (*while* loop - Java/C++/C# pseudocode)

```
int a=0;
while (a<=2) {
        print a;
        a=a+1;
}
```

Extra Instructions:

- Branch on Zero - **BRZ**
- Branch on Positive - **BRP**
- Unconditional branch - **BR**

opcode	mnemonic	
7	**BRZ 70**	if the calculator **= 0** go to the instruction at address 70 and execute it else execute the next instruction
8	**BRP 70**	if the calculator **≥ 0** go to the instruction at address 70 and execute it else execute the next instruction
6	**BR 50**	go to the instruction at address 50 and execute it, unconditional branch

Ex.

650 BR 50 jump to instruction located at address 50

Like an instruction GOTO 50 in BASIC programming language.

All branch instructions change the address in the Instruction location counter.

The LMC performs instructions in <u>machine cycles</u> or (instruction cycles).

The instruction cycle consists of:

- <u>fetch cycle</u>, in which the LM finds out what instruction he is to execute, and
- <u>execute cycle</u>, in which he actually performs the work specified in the instruction

Refer to Figures 6.5a and 6.5b in the textbook on pp. 190-191 or the slides 21-25 posted on BB.

Ex. Write the Java/C++/C# program segment to find the positive difference of two #s read.

```
int a, b;
read a, b;        //or a=63; b=36;
if (a>b)
        print a-b;
else
        print b-a;
```

Ex. Write the equivalent LMC program to find positive difference of two #s read.

address	mnemonic form	numeric code	comments
00	**IN**	901	//store the contents of in-basket in the calculator
01	**STO 21**	321	//store the contents of the calculator in memory location 21
02	**IN**	901	//see above
03	**STO 22**	322	//store the contents of the calculator in memory location 22
04	**SUB 21**	221	//calculator:=calculator-contents of memory location 21
05	**BRP 08**	808	//go to address 08, if calculator ≥0; otherwise proceed to the next instruction
06	**LDA 21**	521	//load contents of memory location 21 to the calculator
07	**SUB 22**	222	//calculator:=calculator-contents of memory location 22
08	**OUT**	902	//output
09	**HLT**	000	//coffee break
... .	.....	....	....
... .	.....	....	....
21	**DAT**		
22	**DAT**		

The instructions stored at addresses 00-09 occupy the program area, whereas data stored at addresses 21-22 occupy the data area.

We will trace the execution of the program on the white board in class for the following #s entered (put in to the in-basket):

a) 63 36

b) 36 63

During the compilation process your Java/C++/C# program would be converted to a binary object code. The binary object code will include a series of simple/primitive instructions similar to the LMC program above. However, for simplicity the LMC program uses the mnemonic form and decimal notation such as STO 21 or 321 to represent operation codes and addresses.

Ex. 6.9, p. 194

Input: 3   34 17 19

A program segment in C++

```
int sum=0, count, number;
read count;                 //assume you will read in 3 for the count
while (count > 0) {
        read number;        //read in 34, 17, and 19 in this order, one at a time
        sum=sum+number;     //sum +=number;
        count=count-1;      //count--;
}
print sum;
```

The LMC program

Need to:
- implement a loop somehow
- assume that mailboxes (memory locations) 91 and 99 are initialized with 1 and 0, respectively
- assume that mailboxes 90 and 99 will store the current count and the sum, respectively

```
00    IN          //input the count 3
01    SUB 91      //decrement count by 1
02    STO 90      //current count is stored in mailbox 90
03    BRP 05      //if calculator ≥0 execute the instruction at address 05; else execute next
                    instruction
04    BR   10     //jump to address 10, done, write out results
05    IN          //input 34, 17, 19, one at a time
06    ADD 99      //add the sum stored in memory location 99 to the number just read in,
                    0+34=34, 34+17=51, 51+19=70
07    STO 99      //save the new sum in memory location 99; the contents will initially be
                    34, then 51, and finally 70
08    LDA 90      //restore the count in the calculator
09    BR   01     //jump to instruction at address 01 to decrement the count by 1
10    LDA 99      //load the sum to the calculator (should be 70)
11    OUT         //write results: 70
12    HLT         //coffee break
....
....
90    ???               //data, contents initially unknown, stores the current count 2, 1, 0,
                        -1, at end stores –1.
91    001               //data, used to decrement the count by 1
....
99    000               //data: stores the sum, initially 0, at end stores 70
```

The instructions stored at addresses 00-12 occupy the program area, whereas data stored at addresses 90-99 (specifically 90, 91, and 99) occupy the data area.

We will trace the execution of the program on the white board in class.