|  |  |  |  |
| --- | --- | --- | --- |
| **Mailbox** | **Numeric code** | **Operation** | **Comments** |
| **00** | **901** | **INBOX --> ACCUMULATOR** | **INPUT the first number, enter into calculator (erasing whatever was there)** |
| **01** | **308** | **ACCUMULATOR --> MEMORY[08]** | **STORE the calculator's current value (to prepare for the next step...)** |
| **02** | **901** | **INBOX --> ACCUMULATOR** | **INPUT the second number, enter into calculator (erasing whatever was there)** |
| **03** | **309** | **ACCUMULATOR --> MEMORY[09]** | **STORE the calculator's current value (again, to prepare for the next step...)** |
| **04** | **508** | **MEMORY[08] --> ACCUMULATOR** | **(Now that both INPUT values are STORED in Mailboxes 08 and 09...)**  **LOAD the first value back into the calculator (erasing whatever was there)** |
| **05** | **209** | **ACCUMULATOR = ACCUMULATOR - MEMORY[09]** | **SUBTRACT the second number from the calculator's current value (which was just set to the first number)** |
| **06** | **902** | **ACCUMULATOR --> OUTBOX** | **OUTPUT the calculator's result to the OUTBOX** |
| **07** | **000** | **(no operation performed)** | **HALT the LMC** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructions** | | | |
| **Numeric code** | **Mnemonic code** | **Instruction** | **Description** |
| **1xx** | **ADD** | **ADD** | **Add the value stored in mailbox xx to whatever value is currently on the accumulator (calculator).**  **Note: the contents of the mailbox are not changed, and the actions of the accumulator (calculator) are not defined for add instructions that cause sums larger than 3 digits. Similarly to SUBTRACT, one could set the negative flag on overflow.** |
| **2xx** | **SUB** | **SUBTRACT** | **Subtract the value stored in mailbox xx from whatever value is currently on the accumulator (calculator).**  **Note: the contents of the mailbox are not changed, and the actions of the accumulator are not defined for subtract instructions that cause negative results - however, a negative flag will be set so that 7xx (BRZ) and 8xx (BRP) can be used properly.** |
| **3xx** | **STA** | **STORE** | **Store the contents of the accumulator in mailbox xx (destructive).**  **Note: the contents of the accumulator (calculator) are not changed (non-destructive), but contents of mailbox are replaced regardless of what was in there (destructive)** |
| **5xx** | **LDA** | **LOAD** | **Load the value from mailbox xx (non-destructive) and enter it in the accumulator (destructive).** |
| **6xx** | **BRA** | [**BRANCH**](https://en.wikipedia.org/wiki/Branch_%28computer_science%29) **(unconditional)** | **Set the program counter to the given address (value xx). That is, value xx will be the next instruction executed.** |
| **7xx** | **BRZ** | **BRANCH IF ZERO (**[**conditional**](https://en.wikipedia.org/wiki/Conditional_%28programming%29)**)** | **If the accumulator (calculator) contains the value 000, set the program counter to the value xx. Otherwise, do nothing. Whether the negative flag is taken into account is undefined. When a SUBTRACT underflows the accumulator, this flag is set, after which the accumulator is undefined, potentially zero, causing behavior of BRZ to be undefined on underflow. Suggested behavior would be to branch if accumulator is zero and negative flag is not set.**  **Note: since the program is stored in memory, data and program instructions all have the same address/location format.** |
| **8xx** | **BRP** | **BRANCH IF POSITIVE (conditional)** | **If the accumulator (calculator) is 0 or positive, set the program counter to the value xx. Otherwise, do nothing. As LMC memory cells can only hold values between 0 and 999, this instruction depends solely on the negative flag set by an underflow on SUBTRACT and potentially on an overflow on ADD (undefined).**  **Note: since the program is stored in memory, data and program instructions all have the same address/location format.** |
| **901** | **INP** | **INPUT** | **Go to the INBOX, fetch the value from the user, and put it in the accumulator (calculator)**  **Note: this will overwrite whatever value was in the accumulator (destructive)** |
| **902** | **OUT** | **OUTPUT** | **Copy the value from the accumulator (calculator) to the OUTBOX.**  **Note: the contents of the accumulator are not changed (non-destructive).** |
| **000** | **HLT/COB** | **HALT/COFFEE BREAK** | **Stop working/end the program.** |
|  | **DAT** | **DATA** | **This is an** [**assembler**](https://en.wikipedia.org/wiki/Assembly_language) **instruction which simply loads the value into the next available mailbox. DAT can also be used in conjunction with labels to declare variables.** |