Little Person Computer

# Assembler design

# Introduction

The Little Person Computer is an extension of the Little Man Computer which was created by Dr Stuart Madnick in 1965 as a teaching tool to introduce the concepts of how von Neumann architecture computers fetch, decode and execute instructions.

The Little Person Computer offers the following extensions:

|  |  |
| --- | --- |
| **Change** | **Rationale** |
| Renamed to Little Person Computer | It’s less sexist |
| LMC uses denary, LPC uses binary (or hex)  LMC instructions are 3 digit denary (one digit opcode and 2 digit signed operand). LPC instructions are 1 or two bytes (opcode, addressing mode and 8 bit two’s complement operand) | It’s easier to see how individual bits affect the operation of the processor. |
| LMC has 99 mailboxes, LPC has 256 addresses | LPC is an 8 bit CPU. 28 gives 256 addresses |
| LMC only uses direct addressing for memory access, arithmetic and branching but LPC can use immediate, direct, indirect or indexed addressing | LPC is designed for students studying |
| LMC has one general purpose register (the accumulator) and a program counter. In addition to these, LPC has a current instruction register, memory address register, memory data register and a stack pointer register | LPC is designed to mimic the conceptual model of a processor in OCR A Level Computer Science. The stack pointer register has been added so that indexed addressing can work effectively. |

# Registers

As indicated above, the Little Man Computer CPU only has two registers. The Little Person Computer has

|  |  |  |  |
| --- | --- | --- | --- |
| Register | Abbreviation | Description | Initial Value |
| Program Counter | PC | Stores the address of the next instruction | 0 |
| Accumulator | ACC | General purpose register used to store data loaded from the immediate access store or to store the result of a calculation | 0 |
| Current Instruction Register | CIR | Stores the instruction that is due to be executed in the current cycle | 0 |
| Memory Address Register | MAR | Determines the direct address of a load / store operation with the immediate access store | 0 |
| Memory Data Register | MDR | Buffer which stores the value being read / written from/to the immediate access store | 0 |
| Stack Pointer | SP | The base address of used for any indexed addressing reads / writes from / to the immediate access store | 0 |

# Instruction set

The little person computer is an extension of the Little Man Computer.

It retains the same instruction set but with some differences.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instruction** | **Number of bytes** | **Description** | **Machine code (hex)** | **Addressing modes [a\_m] supported** |
| ADD | 2 | Add a value stored in memory to the value in the accumulator register | 1[a\_m] [value] | # Immediate  & Direct  ~ Indirect  [] Indexed |
| SUB | 2 | Subtract a value stored in memory from the value in the accumulator register | 2[a\_m] [value] | # Immediate  & Direct  ~ Indirect  [] Indexed |
| STA | 2 | Store the value in the accumulator register to memory | 3[a\_m] [value] | & Direct  ~ Indirect  [] Indexed |
| SP | 2 | Load a value into the stack pointer register | 4 [a\_m] [value] | # Immediate  & Direct |
| LDA | 2 | Load a value from memory into the accumulator register | 5[a\_m] [value] | # Immediate  & Direct  ~ Indirect  [] Indexed |
| BRA | 2 | Unconditional branch | 6[a\_m] [value] | # Immediate  ~ Indirect  [] Indexed |
| BRZ | 2 | Branch if the value in the accumulator is zero | 7[a\_m] [value] | # Immediate  ~ Indirect  [] Indexed |
| BRP | 2 | Branch if the value in the accumulator is positive (if the MSB is 0) | 8[a\_m] [value] | # Immediate  ~ Indirect  [] Indexed |
| INP | 1 | Read a value from the input device and store it into the accumulator register | 91 |  |
| OUT | 1 | Output the value stored in the accumulator register | 92 |  |
| HLT | 1 | Halt the processor | 00 |  |
| DAT | 1 | Stores a literal data value | [data] |  |

## Addressing mode:

[a\_m] is a 4 bit sequence indicating the addressing mode used for addition, subtraction, memory access and branch instructions

[value] is an 8 bit unsigned value which specifies either the address or the immediate value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Addressing mode** | **[a\_m] hex value** | **Example mnemonic** | **Example machine code** | **Example explanation** |
| Immediate access | 0 | LDA #27 | 50 1B | Load the denary value 27 (hex 1B) into the accumulator. No memory read required. |
| [[1]](#footnote-1)Direct access | 1 | STA &27  STA 27 | 31 1B | Store the value currently in the accumulator register into the immediate access store at denary address 27 (hex address 1B) |
| Indirect access | 2 | STA ~27 | 32 1B | Store the value currently in the accumulator register into the immediate access store at the address which is currently stored as a value at denary address 27 (hex address 1B) |
| Indexed access | 3 | STA [27] | 33 1B | Store the value currently in the accumulator register into the immediate access store at the address which is the sum of the value stored in the stack pointer and the denary value 27 |

# Assembler stages:

# Lexical Analysis

The first stage of turning LPC assembly into machine code is to split up the assembly code into tokens.

The source code is split into lines.

Comments (starting with //) are stripped out of the lines.

Each line is then split into tokens wherever whitespace is encountered. Leading or trailing whitespace is ignored and contiguous whitespace is treated all as one separator.

For example:

LDA #24 // Load the value 27 into the accumulator

Is first stripped of any comments or any leading or trailing whitespace before being split into the tokens

LDA and #24

If the code had a label:

start: LDA #24

the tokens would be start:, LDA and #24

# Syntax Analysis

1. Where no symbol (#&~ or []) is specified, the addressing mode is assumed to be direct [↑](#footnote-ref-1)