# Software Lab 2 & 3

Assembler, Linker, Loader and Simulator

Shail Rakhunde

Roll no. BT10CSE065

The architecture of this machine is very much similar to the SIC (Simplified Instructional Computer) architecture. The SIC machine is designed to illustrate some of the most basic and commonly encountered hardware features and concepts, while avoiding most of the idiosyncrasies that are found in real machine.

This document specifies the features and concepts supported four important constituents of the machine that is Assembler, Linker, Loader, and a Virtual Machine to execute the program on.

## Instruction Set:

| Mnemonic | Format | Opcode | Operation |
|---|---|---|---|
| LDA m | 3/4 | 00 | A ← (m...m+2) |
| LDX m | 3/4 | 04 | X← (m...m+2) |
| STA m | 3/4 | 0C | m...m+2 ← (A) |
| STX m | 3/4 | 10 | m...m+2 ← (X) |
| ADD m | 3/4 | 18 | A ← (A) + (m...m+2) |
| SUB m | 3/4 | 1C | A ← (A) (m...m+2) |
| MUL m | 3/4 | 20 | A ← (A) * (m...m+2) |
| DIV m | 3/4 | 24 | A ← (A) / (m...m+2) |
| COMP m | 3/4 | 28 | A ← (m...m+2) |
| JEQ m | 3/4 | 30 | PC ← m if CC set to = |
| JGT m | 3/4 | 34 | PC ← m if CC set to > |
| JLT m | 3/4 | 38 | PC ← m if CC set to < |
| JSUB m | 3/4 | 48 | L ← (PC); PC ← m< |
| RSUB | 3/4 | 4C | PC ← (L) |

## Memory:

Total size of memory is 65536 bytes. It is 8 bit or a byte addressable memory. Three consecutive bytes form a word. Memory is addressed in Big Endian Format i.e. high order bytes first. Each byte is stored as two individual chars representing the hexadecimal byte.

## Registers:

1> **A** -Accumulator, Used for arithmetic operations.
2> **X** -Index register, Used for addressing.
3> **L** -Linkage register, The Jump to subroutine (JSUB) and Return from subroutine (RSUB) instructions use it to store return address.
4> **PC** -Program Counter, Contains the address of next instruction to be fetched for execution.
5> **SW**-Status Word, Contains a condition code (CC).

## Object Code Format:

- A Header Record
  Record 1: **H**
  Record 2: Segment name
  Record 3: Initial program load address
  Record 4: Length of the segment
  Example: H^test^x1000^005D
- Text Records
  Record 1: **T**
  Record 2: Address at which the information is to be stored
  Record 3: Initial value of that address
  Example: T^1006^0C^0036
- End Record
  Record 1: **E**
  Record 2: Address at which execution is to begin.
  Example: E^1000

Each record is separated by a separator ('^' is used as separator).

## Assembler:

The design has a two pass assembler. It makes use of two data structures:

OpTable – To search particular operation and corresponding opcode.

SymTable – To store symbols, labels and corresponding location in memory.

EsTable – External symbol table is used in assembler as well, to avoid messing up of external references.

## Simulator:

The simulator simulates the SIC machine with above configuration. It works in three modes: Quiet, Trace and Step as mentioned in problem statement. It makes use of the Processor class object which very much behaves similar to a basic practical microprocessor and supports the instructions set mentioned above. The Processor class can be found in header file “processor.h”.

## Linking Loader:

It is based on following three processes:

1> **Linking**, which combines two or more separate object codes and supplies the information needed to allow references between them.

2> **Relocation**, modifies the program to load it at address different than the one specified in the program.

3> **Loading**, brings the object program into available location in main memory for execution.

The algorithm is much more complicated than absolute loader as it has to manage the external references to some symbols that may be present in another program. The program consists of only one execution sequence which may refer to some other variables and subroutines at different memory location.

The assembler provides two pseudo operations for this purpose:

**EXTDEF** - Defines symbol in current program that can be referenced by others.

```
Example:    EXTDEF      VAR   103C
```

In object code they are written in '**D**' records.

```
Example:    D^VAR^103C
```

**EXTREF** – Refer symbols from others that are used in current program.

```
Example:    EXTREF      VAR
```

In object code they are written in '**R**' records.

```
Example:    R^VAR
```

The Linking Loader uses Two Pass Implementation. In first pass, all external definitions and references are resolved with the help of EsTab (External Symbol Table). The second pass load all the opcodes and operands into the memory. The first program is loaded at starting address given, while other programs are loaded sequentially after that with a gap of 15 bytes at the end of each. All the other programs or rather their memory usage are modified according to that.

The simulator executes the very first program loaded, i.e. till the first END record, though it can refer to any memory location in entire address space to read some value or execute a subroutine.

## Forms of Output:

The application uses both forms of output i.e. Console and files. The information such as progress, error codes is usually given through console. The simulator is completely console based.

Files are extensively used by assembler. The assembler takes an input file with .txt format. It then generates an intermediate file with same name and .i file extension after pass 1, and finally output object file with same name and .o file extension.

The object file generated by assembler is the input to linking loader which loads the program into main memory. Finally simulator performs the execution with all interactions through console.