Evolution of Computing

Assignment 7

Jaidev Ramakrishna

Introduction

The computer that the simulator has been built for is the EDSAC. The source of the information that was used is the EDSAC Replica Project, run by the National Museum of Computing and sponsored by Fujitsu. This project can be found at http://www.tnmoc.org/special-projects/edsac

The project has produced a simulator that very closely simulates the EDSAC at a deep level. They have created an interface that is almost identical to the original computer, and which is capable of running the code that was produced in the 1940s with no modification.

This assignment does not make use of any of the code from the project – it is entirely original work. However, the project has also produced a manual that provides a detailed specification of the EDSAC. It is this document that has been used to create this simulator; specifically, the commands and instruction set. This document can be found at http://www.dcs.warwick.ac.uk/~edsac/Software/EdsacTG.pdf

Implementation Details

- The simulator is in the form of an interactive console. It is too complex for a menu-driven interface, so a simple programming language has been designed into it to facilitate operations. All commands are 2 letters, followed by one or two operands.
- The memory has been implemented as a list of 1024 long integers (64 bits each). This was because although the 18 bit (17 effective) numbers of the EDSAC would fit in 32 bit integers, it was possible to chain 2 together to form 35 bit double precision numbers, which would overflow from integers.
- This code requires Java 8 to successfully compile and run. It has been tested on both Windows and Linux.

Memory File Format

- Memory files can be loaded using the “lm <filename>” command at the simulator console.
- The memory file is a map of locations and their values. When a memory file is loaded, the values contained in it will be overwritten onto the existing memory.
- The EDSAC has 1024 memory locations, so addresses go from 0 to 1023. Any invalid values will be ignored.
- Not all locations need to be mentioned – those not mentioned will be unchanged.
- Locations do not need to be mentioned in order. Locations can be mentioned multiple times, but only the last instance of each location will be inserted.
- Each line contains one mapping, which is a memory location and its value, separated by a space. There should be no indentation, other characters, or trailing spaces or table.
- The “#” symbol can be used to insert single line comments. Blank lines are also ignored.
- Two examples of memory files are the attached fibonacci.mem and powers.mem.
- Memory dumps produced by the dm <filename> command will also have the same format. However, they will have all 1024 values in order and without repetition. Using this, it is actually possible to save and reload the state of the machine.

Program File Format

- Memory files can be loaded using the "lp <filename>" command at the simulator console.
- The program file is a list of instructions that are executed sequentially.
- Each line contains one instruction, which is a command followed by any arguments it might take like locations and values, separated by a space. There should be no indentation, other characters, or trailing spaces or table.
- The "#" symbol can be used to insert single line comments. Blank lines are also ignored.
- An example of a program file is the attached powers.prg.

EDSAC Commands

Command	Arguments	Notes
A	n	Add the number in storage location n into the accumulator
S	n	Subtract the number in storage location n from the accumulator
H	n	Copy the number in storage location n into the multiplier register
V	n	Multiply the number in storage location n by the number in the multiplier register and add the product into the accumulator
N	n	Multiply the number in storage location n by the number in the multiplier register and subtract the product from the accumulator
T	n	Transfer the contents of the accumulator to storage location n and clear the accumulator
U	n	Transfer the contents of the accumulator to storage location n and do not clear the accumulator
C	n	Collate [logical and] the number in storage location n with the number in the multiplier register and add the result into the accumulator
R	n	Shift the number in the accumulator n places to the right
L	n	Shift the number in the accumulator n places to the left
E	n	If the sign of the accumulator is positive, jump to location n; otherwise proceed serially
G	n	If the sign of the accumulator is negative, jump to location n; otherwise proceed serially
I	n	Read the next character from paper tape, and store it as the least significant 5 bits of location n
O	n	Print the character represented by the most significant 5 bits of storage location n
F	n	Read the last character output for verification
X		No operation
Y		Round the number in the accumulator to 34 bits
Z		Stop the machine and ring the warning bell

Limitations

- This assignment does not replicate the deep functionality of the EDSAC. It only attempts to recreate it on an instruction level. Consequently, certain functions and operations are not available, particularly those that deal with bit level operations and certain components like the tape reader, GUI display or subroutine library.
- It should be noted that some of the features present in the code are not documented here. This is because they are experimental, and not fully functional. However, a full list can be seen by running the "he" command.
- Certain commands are unnecessary due to the implementation details, but they have been included as dummy functions in order to enhance compatibility. They will, however, not produce any output.

- Very rudimentary error checking is done. Memory bounds are checked to prevent overflow, as are the number of arguments passed to functions. But it is still possible to crash the program by using alphanumeric inputs to memory locations in certain places.
- Exception handling is done thoroughly only in the case of file handling. It is advisable to place files in the same directory.

Simulator Commands

Command	Arguments	Notes
ce		Clear execution trace
cm		Clear memory map
cp		Clear program
de	<filename>	Dump execution trace to file
dm	<filename>	Dump memory to file
ee		Examine execution trace
el	<location>	Examine memory location
em	<start> <end>	Examine range of locations
ex		Exit simulator
he		Print Help
lm	<filename>	Load memory map from file
lp	<filename>	Load program from file
ls	<filename>	Load script from file
lt	<filename>	Load tape from file
pb	<button>	Push button
pr	<string>	Set Prompt
rc	<command>	Run command
rn		Run command
rs		Run script
sm	<location> <value>	Set value of location

Running the Simulator

- The code is contained within the EDSAC.java file.
- It can be compiled using "javac EDSAC.java", and run using "java EDSAC". This brings up the command shell, where the above-mentioned commands can be used.
- The "he" command is used to print help information, and "ex" is used to exit the simulator.

Example 1

We will first see how to load a memory map. The file fibonacci.mem loads the first 10 Fibonacci numbers into memory from location 100 onwards.

We first load the file into memory using "lm fibonacci.mem". We then examine memory locations 99 to 110. We see that the loading has occurred from 100 to 109.

```
EDSAC Simulator
Type he for help

EDSAC>lm fibonacci.mem

EDSAC>em 99 110
        99 : 0
        100 : 0
        101 : 1
        102 : 1
        103 : 2
        104 : 3
        105 : 5
        106 : 8
        107 : 13
        108 : 21
        109 : 34
        110 : 0

EDSAC>
```

Example 2

The earlier program was just a simple loading of values into memory. We will now use this same technique to load some initial values into memory and then perform operations in order to accomplish some task.

We have a program that prints the first 10 powers of 2. We use “lm powers.mem” to initialize two memory locations. We then load the program using “lp powers.prg”. We have now prepared the machine. We execute the program using “rn”. Once execution is completed, we can examine memory locations using “em 1 10”. We see that these locations have been loaded with the powers of 2.