# SIMP Processor Simulator Documentation

Table of Contents

[SIMP Processor Simulator Documentation 1](#_Toc188895476)

[Submitted By 2](#_Toc188895477)

[Project Overview 2](#_Toc188895478)

[Project Directory Structure 2](#_Toc188895479)

[Usage 2](#_Toc188895480)

[The assembler - asm 3](#_Toc188895481)

[The simulator - sim 4](#_Toc188895482)

[Programs 6](#_Toc188895483)

[mulmat 6](#_Toc188895484)

[binom 8](#_Toc188895485)

[circle 9](#_Toc188895486)

[disktest 10](#_Toc188895487)

[fib 11](#_Toc188895488)

## Submitted By

Roy Sarafov - 209619477

Yoav Dychtwald - 209518299

## Project Overview

The SIMP Processor Simulator is an educational tool that simulates a custom processor. It allows users to write assembly code, assemble it into machine code, and run instructions through a software-based CPU simulator.   
Every file, both .c and .asm files are well documented within the code.   
In addition to the inner documentation, the following file will explain a bit more about the purposes of each file/function.

You can find our repository code by pressing this [link](https://github.com/roy-sarafov/simp_processor_simulator).   
\* If you opened the repository link, notice there is a difference between the ***repository Directory Structure*** and the ***Directory Structure requested in the assignment***.   
This file describes the structure as requested in the assignment.

## Project Directory Structure

The repository contains the following main directories:

1. asm – contains the assembler source code
2. sim – contains the simulator source code
3. circle – contains the circle assembly program and its input + output files
4. fib – contains the fib assembly program and its input + output files
5. mulmat – contains the mulmat assembly program and its input + output files
6. disktest – contains the disktest assembly program and its input + output files
7. binom – contains the binom assembly program and its input + output files

## Usage

To run the assembler, you should write the following command in the command line:

asm.exe program.asm imemin.txt dmemin.txt

To run the simulator, you should write the following command in the command line:

sim.exe imemin.txt dmemin.txt diskin.txt irq2in.txt dmemout.txt regout.txt trace.txt hwregtrace.txt cycles.txt leds.txt display7seg.txt diskout.txt monitor.txt monitor.yuv

## The assembler - asm

This module includes a single main.c file which assembles a given assembly program into machine code, according to the given ISA.  
Key functions and purposes:

|  |  |
| --- | --- |
| Function | Description |
| getRegisterNumber | maps register names and I/O register names to their corresponding numerical values, which are used during the assembly process to encode instructions. |
| getOpcodeNumber | maps assembly instruction mnemonics (e.g., add, sub, lw) to their corresponding opcode numbers, which are used to generate machine code during the assembly process. |
| getLabelAddress | searches for a given label in the labelTable and returns its corresponding memory address. This function is crucial for resolving symbolic labels to numerical addresses during the second pass of the assembly process. |
| firstPass | The firstPass function performs an initial scan of the input assembly file to identify and store labels with their respective line addresses, which are later used for resolving symbolic references during the second pass of assembly. |
| secondPass | processes the assembly file a second time to generate the final machine code instructions and initialize data memory. It converts assembly instructions into binary representations and writes them to output files for instruction memory and data memory. |
| main | serves as the entry point of the assembler program. It orchestrates the overall assembly process by performing the first and second passes over the input assembly file to generate machine code and data memory output. |

## The simulator - sim

This module includes source files and header files of different purposes to run instructions through a software-based CPU simulator.  
Key source files and purposes:

|  |  |
| --- | --- |
| File | Description |
| main.c | • This file serves as the core of the simulator, orchestrating the execution of the fetch-decode-execute cycle.  • It initializes key components such as registers, memory, I/O, disk, and monitor.  • Manages input and output operations by reading instruction and data memory files and generating output logs.  • Provides logging functions for tracking the state of registers, LEDs, and the 7-segment display.  • Ensures proper cleanup by closing files and writing final simulation results upon completion.  • Implements error handling for file operations and argument validation to ensure correct simulator execution. |
| instruction\_fetch.c | Handles fetching of instructions from memory based on the program counter (PC) value. |
| instruction\_decode.c | Implements the decoding of fetched instructions to extract opcode and operand details. |
| execution.c | Defines functions to execute decoded instructions by performing operations based on the instruction type and updating the system state. |
| interrupts.c | Manages interrupt handling in the simulator, including handling various interrupt requests and managing their priorities. |
| disk.c | Implements disk-related operations, including reading from and writing to simulated disk sectors. |
| registers.c | Manages CPU registers, providing functions to initialize, read, and write register values during simulation. |
| memory.c | Handles instruction and data memory operations, including loading memory from files and reading/writing memory locations. |
| io.c | Defines and manages input/output operations such as reading and writing to I/O registers, handling LED and 7-segment display updates. |
| monitor.c | Provides functions for handling the simulated monitor, including pixel rendering and display memory management. |

## Programs

### mulmat

This directory holds a program **mulmat.asm** that performs a multiplication of two 4x4 matrices (each entry in the result matrix is a scalar multiplication of the corresponding row and column of the two matrices). The values of the first matrix are given in addresses 0x100 to 0x10F, the second matrix in 0x110 to 0x11F, and the result matrix should be written to 0x120 to 0x12F. You can assume that there is no overflow in the computation.  
Each matrix is ordered in memory in an increasing row order, and each row from left to right.

Matrix A we chose:

Matrix B we chose:

Matrix C – Result needed:

You can also the find the input and output files of running this program through our assembler and simulator.

Key files and purposes:

|  |  |
| --- | --- |
| File | Description |
| mulmat.asm | Assembly source code. Loops over every index of the result matrix and calculates the values according to the index. Writes the output to the correct offset in the memory. |
| imemin.txt | Instruction memory input generated from the assembler. |
| dmemin.txt | Data memory input generated in accordance with the “. word” commands. We initiated 2 matrices. Their values can be seen above. |
| diskin.txt | Disk input file – initiated to 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| irq2in.txt | IRQ2 input file |
| dmemout.txt | Data memory output after simulator. You can view the result matrix in the correct offset with the correct values as needed. |
| regout.txt | Register values after execution |
| trace.txt | Execution trace |
| hwregtrace.txt | Hardware register trace |
| cycles.txt | Number of execution cycles |
| leds.txt | LED output trace |
| display7seg.txt | 7-segment display output |
| diskout.txt | Disk output file – there is no disk commands here, so in is copied to out as is: 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| monitor.txt | Monitor output |
| monitor.yuv | Monitor graphical output in YUV format |

### binom

This directory holds a program **binom.asm** that calculates the Newton binom coefficient recursively according to the following algorithm. At the start of the run n is given at address 0x100, k is given at address 0x101, and the result should be written to address 0x102. It can be assumed that n is small enough such that overflow doesn’t happen.

We chose: n=6=0x6 and k=4=0x4. The result binom is 15 = 0xF.

You can also the find the input and output files of running this program through our assembler and simulator.

Key files and purposes:

|  |  |
| --- | --- |
| File | Description |
| binom.asm | Assembly source code. Recursion implementation for the calculation of binom. |
| imemin.txt | Instruction memory input generated from the assembler. |
| dmemin.txt | Data memory input generated in accordance with the “. word” commands. We initiated n=6 and k=4 in their matching addresses. |
| diskin.txt | Disk input file – initiated to 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| irq2in.txt | IRQ2 input file |
| dmemout.txt | Data memory output after simulator. You can view the result binom in the correct offset with the correct values as needed. |
| regout.txt | Register values after execution |
| trace.txt | Execution trace |
| hwregtrace.txt | Hardware register trace |
| cycles.txt | Number of execution cycles |
| leds.txt | LED output trace |
| display7seg.txt | 7-segment display output |
| diskout.txt | Disk output file – there is no disk commands here, so in is copied to out as is: 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| monitor.txt | Monitor output |
| monitor.yuv | Monitor graphical output in YUV format |

### circle

This directory holds a program **circle.asm** that draws a circle on the screen. The circle should be placed at the center of the screen, in the white color, and should be a full circle (all the pixels on the edge and inside the circle should be white). The radius of the circle is given at address 0x100. It can be assumed that the circle fits in the screen.  
We chose radius=30.

You can also the find the input and output files of running this program through our assembler and simulator.

Key files and purposes:

|  |  |
| --- | --- |
| File | Description |
| circle.asm | Assembly source code. Implements a circle by referring to the screen as an XY graph, every loop calculating the distance from center and asking if the distance is lower then wanted radius. If yes – paints white. If no – continues. |
| imemin.txt | Instruction memory input generated from the assembler. |
| dmemin.txt | Data memory input generated in accordance with the “. word” commands. We initiated radius value in its matching address. |
| diskin.txt | Disk input file – initiated to 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| irq2in.txt | IRQ2 input file |
| dmemout.txt | Data memory output after simulator. |
| regout.txt | Register values after execution |
| trace.txt | Execution trace |
| hwregtrace.txt | Hardware register trace |
| cycles.txt | Number of execution cycles |
| leds.txt | LED output trace |
| display7seg.txt | 7-segment display output |
| diskout.txt | Disk output file – there is no disk commands here, so in is copied to out as is: 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| monitor.txt | Monitor output – You can see some lines written as FF which is white color. |
| monitor.yuv | Monitor graphical output in YUV format – You can open the file, and see the circle located in the center of the screen, in the matching radius size. |

### disktest

This directory holds a program **disktest.asm**, that moves the contents of the first 8 sectors in the disk drive (sectors 0 to 7) one sector forward such that at the end of the run, sector 1 will contain the contents of the original sector 0 and so on till sector 8 that will contain the contents of the original sector 7.

We chose to initiate the disk in a way that every sector has 128 lines of 8 hexa chars that match their sector index. Meaning sector 0 will have 128 lines of ‘00000000’.  
Sector 1 will have 128 lines of ‘11111111’ and so on.  
You can see that at the end of this run, every sector moved forword and now in sector 1 there is ‘0000000’ and in sector 2 there is ‘11111111’ and so on.

You can also the find the input and output files of running this program through our assembler and simulator.

Key files and purposes:

|  |  |
| --- | --- |
| File | Description |
| disktest.asm | Assembly source code. Reads the sectors from disk to memory while waiting for the disk status to be available. Then writing the memory back to the disk in the matching sectors as requested. |
| imemin.txt | Instruction memory input generated from the assembler. |
| dmemin.txt | Data memory input. |
| diskin.txt | Disk input file – initiated to 8 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index. |
| irq2in.txt | IRQ2 input file |
| dmemout.txt | Data memory output after simulator. You can view the data from the disk in the memory since we loaded the disk to memory before writing it back to disk. |
| regout.txt | Register values after execution |
| trace.txt | Execution trace |
| hwregtrace.txt | Hardware register trace |
| cycles.txt | Number of execution cycles |
| leds.txt | LED output trace |
| display7seg.txt | 7-segment display output |
| diskout.txt | Disk output file – you can see the new sector created, and every sector moved one sector Forword. Now, there are 9 sectors. Each sector has 128 lines of 8 hexadecimal chars matching their sector index - 1. |
| monitor.txt | Monitor output |
| monitor.yuv | Monitor graphical output in YUV format |

### fib

This directory holds a program **fib.asm**. this program was given by you for us to test our code.

This program is mainly for us to compare to the output files we were given in advance.

Our output files and the given output files are identical as expected.

You can also the find the input and output files of running this program through our assembler and simulator.

Key files and purposes:

|  |  |
| --- | --- |
| File | Description |
| fib.asm | Assembly source code for fibonachi. |
| imemin.txt | Instruction memory input (generated) |
| dmemin.txt | Data memory input (generated) |
| diskin.txt | Disk input file |
| irq2in.txt | IRQ2 input file |
| dmemout.txt | Data memory output after execution |
| regout.txt | Register values after execution |
| trace.txt | Execution trace |
| hwregtrace.txt | Hardware register trace |
| cycles.txt | Number of execution cycles |
| leds.txt | LED output trace |
| display7seg.txt | 7-segment display output |
| diskout.txt | Disk output file |
| monitor.txt | Monitor output |
| monitor.yuv | Monitor graphical output in YUV format |