Computer Organization Project – Documentation

# Simulator

## Purpose

This executable is used to simulate a processor’s mechanism for running user defined programs. Calling the program via CLI interface is as follows:

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

where imemin.txt contains the assembled program (see Assembler), dmemin.txt contains the initial data memory, diskin.txt is the input disk, and irq2in.txt defines at which cycles irq2 should interrupt.

Regout.txt, cycles.txt, leds.txt, display7seg.txt, diskout.txt, monitor.txt and monitor.yuv are written at the end of the program, each file’s content is described by its name.

Trace.txt, hwregtrace.txt are updated every cycle.

## Implementation

The program flow is controlled via a main loop, which contains the following logic, in the specified order:

1. Fetch and Decode an instruction
2. Update trace file
3. Instruction Execution (advance PC, or move to new PC)
4. Update hardware trace file (if necessary)
5. Handle timer logic (advance/nullify, toggle interrupt bit)
6. Handle disk logic (advance counter/reset status/read or write data)
7. Check if current cycle is IRQ2 cycle
8. Handle monitor logic
9. Poll interrupts
10. Handle led and display7seg logic
11. Advance cycle counter

At the end of this main loop, the necessary files are written, and finally all allocated memory is freed, and all the files are closed.

# Assembler

## Purpose

This executable is used to translate instructions in assembly syntax into machine code readable by a CPU (in this context, the Simulator). Calling the program via CLI interface is as follows:

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

where program.asm contains the input assembly code, imemin.txt and dmemin.txt will contain the instructions memory map and the data memory map, respectively, at the end of the program.

## Implementation

The program flow is linear, and will be described with a simple list:

1. First the program iterates over program.asm line by line, recording every label it runs into and conducting each “.word” instruction present within the code, writing to dmemin.txt the instructed values to desired addresses.
2. Using the previously mentioned label record, the program iterates over program.asm line by line again, this time each line (if applicable) is parsed into an Instruction struct with each textual field parsed into a numeric counterpart, including label values in imm1 and imm2.
3. For each Instruction instance created, the program calls encode\_instruction(…) to write the “binary” data (actually a string representation of the hexadecimal value) to imemin.txt.
4. At the end of the program all dynamically allocated memory is freed, and the program terminates.