ISA Project Documentation

**Assembler**

The assembler gets an input file – the .asm program, and an output file – memin, which will be used as an input file to the simulator.

The assembler opens the files and goes over the asm program twice: in the first run it save the names and addresses of all the label found in the file, and in the second run it reads each line in the file, translates it to a command in machine language according to the SIMP instructions, and saves it to the memin file.

* First run – parse\_labels(FILE\* asm\_prog): takes each line in the asm program and does the following:
  + Turns the line from a string to a list of arguments and removes comments.
  + Checks if the line has a label (if colons are present).
  + If there is a line it stores it's name and address (according to the current PC counter) in the fields of the struct Label.
  + Calculates the matching opcode index and updates the PC accordingly.
  + When it finishes going over the file it rewinds it so it we can read it again.
* Second run - parse\_instructions(FILE\* asm\_prog): takes each line in the asm program and does the following:
  + Turns the line from a string to a list of arguments and removes comments.
  + Calculates the matching opcode index.
  + If the opcode is ".word" it stores the value in the data argument in the address value's place in the global list Memory (which will be later written into memin).
  + If else – it calculates the matching register index for rd, rs, rt, and for the immediate argument – it gets the matching label address or the immediate number.
  + Calculates the SIMP command from the arguments and stores it in Memory.
  + Updates the last "line" in Memory which has any data that should be written to memin.
* write\_to\_memory(FILE\* memin) – takes each command/data from Memory and writes it as a 32-bit hex word to the corresponding line in memin.

**Simulator**

Simulator: sim.exe  
main file: main.c. main function: main(int argc, char\*\* argv).

the project is divided to modules.  
**cpu module:**

DATA:

each opcode/register/IOregister is defined as an enum with its correct “SIMP” index.

instruction is a struct containing rd,rs,rt,imm and opcode values, already parsed from memory (“memin.txt”).

cpu structure holds the **current** PC counter, parsed instruction, registers, IOregisters, memory and irq status.

operation is a pointer to array of functions. every command operator is a function defined in operators module.

OPERATION:

cpu intiliazed in *sim\_init*().

each clock cycle, the cpu fetches an instruction using fetch\_address(cpu).

each (legal) instruction will be executed by executeInstruction(cpu). otherwise, “ignore and continue”.

**operators module:**

OPERATION:

contains all the commands defined in SIMP.

**filesManager module:**

Data:

irq2 structure contains all the data required to handle irq2 interrupt.  
Operation:

Parses memin.txt and writes memout.txt

Parses diskin.txt and writes diskout.txt

write cycles.txt

initializing irq2 structure.

**main module:**

Operation:

write\_trace(cpu,trace\_file\_desc) writes a trace line for trace.txt.

main(int argc, char\*\* argv) ‘pseudo code’:  
  
1. initializes cpu

2. Parses memin.txt, diskin.txt, initializing irq2.txt and starts parsing. Note that irq2.txt is being parsed during the entire program run, as answered in the course forum.

3. open outfiles

4. for every clock cycle until HALT command:

\* check and handle interrupt

\* fetch correct address

\* write trace

\* handles leds.txt , display.txt and hwregout.txt if IN or OUT commands

\* update irq0status and irq2status (note that irq0status is updated inside executeInstruction(cpu)

\* execute instruction

\* handle disk request

5. write out files and close files

6. free all dynamic allocated memory