Or Bahari 204356315
Or Shahar 208712471
Daniel Kinderman 205684590

ISA Project Documentation

Assembler

Assembler: asm.exe

The assembler program has input file *.asm program and an output file memin.txt, which will be used as an input file for the simulator.

The assembler opens the file and scans the *.asm program twice: in the first scan it saves the names and addresses of all the labels found in the file, and the second scan reads each line in the file, translates it to a command in machine language according to the SIMP instructions, and saves it to memin.txt.

1) First scan – **parse_labels(FILE* asm_prog)**: takes each line in the *.asm program and does the following:
    - Converts 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's name and address are stored (according to the current PC counter) in the fields of the struct Label.
    - Calculates the matching opcode index and updates the PC accordingly.
    - We rewind back to the beginning of the file to start our next scan.
2) Second scan - **parse_instructions(FILE* asm_prog)**: takes each line in the *.asm program and does the following:
    - Converts 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 of the data argument in the address location in the Memory list (which will be later written into memin.txt).
    - else – it calculates the matching register index for rd, rs, rt, and for the immediate argument and parses 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.txt.
3) **write_to_memory(FILE* memin)** – takes each command/data from Memory and writes it as a 32-bit hexadecimal word to the corresponding line in memin.txt.

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").

Or Bahari 204356315
Or Shahar 208712471
Daniel Kinderman 205684590

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

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

OPERATION:

cpu is intiliazed in function named: ***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 processor, each command is implemented in a different function.

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.

Or Bahari 204356315
Or Shahar 208712471
Daniel Kinderman 205684590

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.