308223981
305131328

ISA Project Documentation

The Assembler:

The assembler reads the .asm file 2 times:

1. The first read is done to store the labels in an array of the structure Label which has two fields: The label name (.name) and the label pc (.pc). The read is done using the strtok() function contained in the C library 'string.h'. This function breaks the sentence up into smaller strings (tokens) using given delimiters (in out case: space, tab, comma, new line).

2. The second read is the main run over the .asm file, and here we use the same strtok() method to go over each line of the code. We store each word from the code into an array called words[], which at the end of each line of the file (except in the case of `.word') looks like so:

opcode	rd	rs	rt	rm	imm (קבוע)

we then decode the opcode by comparing it to an array which acts as sort of a hash function, giving each opcode its correct hexadecimal number, and printing that number onto the memin.txt file. Then we use functions that we wrote (and are located in the asm_funcs.c file, with detailed descriptions in the asm_funcs.h header file) to properly decode and print the rest of the fields of the array onto the memin.txt file.

In the case of `.word': we send the words[] array into a function named word_func (also located in asm_funcs.c and described in asm_funcs.h) and print the proper lines onto the memin.txt file.

The Simulator:

The simulator starts by copying the memin.txt file into the (currently empty) memout.txt file. Then, we begin reading the memin.txt file again. For each line, we save the 4B hexadecimal instruction in the array inst[] we print the current trace onto the trace.txt file, decode the instruction and perform the proper operations, and lastly update the counter. The instruction decoding goes as follows:

1. We decode the immediate field and convert it from hex to decimal.
2. A switch case on the opcode decides which operation to perform[1], with more complicated operations using functions[2] to do the proper task. Every type of operation will update the pc accordingly. In the event that the opcode is not recognized (i.e. an opcode belonging to a 'reserved' operation), the PC is incremented so that the simulator will continue functioning, and no operation is performed.

Once the simulator finishes reading the memin.txt file for the second time, we print the contents of the registers (contained in the regs[] array) to the regout.txt file and print the value of the counter to the count.txt file.

[1] "performing an operation" consists of 2 main parts: updating the PC and filling the contents of the registers (i.e. filling the array regs[]) and, in the case of store-word, writing onto the memory.
[2] The functions are described in detail in the code, and there is no need to provide further detail in this document