Wi TCSS 372 B programing project

Computer simulator

Group:

Benjamin Russell

Christian Boettcher

Class: Computer Architecture

Instructor: Dr. Xing Liu

Short design description

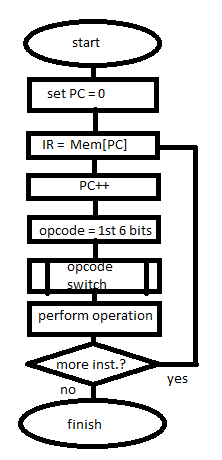
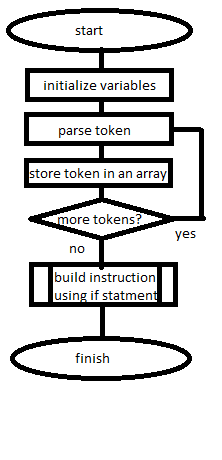
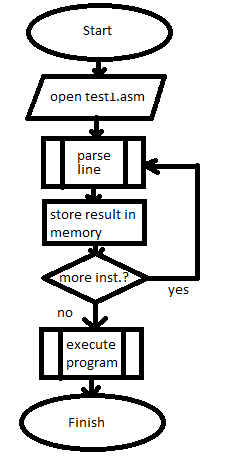
For this project, we designed a simulator able to load a MIPS instruction set and be able to execute the instructions; these instructions were to be 32-bits in length. The features that were implemented were a display of the 32 registers and their content and the content of key memory locations to our “test.asm”.

Our first phase involved converting an .asm file in the MIPS format to a machine instruction of 32-bits long. To begin with, the main function first opens the .asm file of choice and reads in the file line by line. While each line is read, the main function passes the line into a function made for parsing the line into tokens, converting them into instructions, and will return these instructions in the form of a union which will be passed into an array. Within this function, parseLine, the first element of the line, delimited by a space, is plucked off and identified as this first element is always responsible for providing the opcode and determining whether the instruction is of type R, I, or J. This involved a tedious amount of “else if” statements utilizing the strcmp function. Once the type of instruction is discovered, the program can build up the appropriate struct placed inside a union of type Instruction using a related set of functions that obtain the bits for each of those instructions fields. Once the instruction is built using the appropriate struct within our union “Instruction”, a fourth struct “inst” is used within “Instruction” which is a 32-bit unsigned int used to perform bit manipulation on which ever instruction is within each union.

In part two of the project, our group had to take our instruction set and show our simulator could execute some basic commands such as add, sub, and, or, lw, and sw. As our simulator parses the instruction lines and creates a 32-bit instruction, the main function places these instructions in a global array of type int32\_t used as memory which will be used in the execute function. Within this function, the program begins by initializing the program counter at 0 which points at the first instruction in memory and increments after each instruction is loaded into the instruction register for execution. At this point, our program uses micros to identify the opcode of the instruction which goes through a switch statement and performs the operations and stores the results in appropriate registers. In the case of an r type instruction, micros are used to discover the function code as well. Our group also used micros to discover any registers or immediate value used in each instruction. After the instructions are executed, the content of each register is printed out in the command environment as well as the memory location of a word stored in a particular spot in memory related to our “test.asm” file.

Flow-chart:

Main Parse Execute



Screen shots:

