Design Document

Robin Chen

**Abstract**

This program reads a file with 32-bits codes, checks that all the codes are in good format, and then converts them into MIPS assembly instructions.

The program contains four classes: Opcode, BitCodeParser, Instruction, RegisterTable; one main file: BitCode.cpp; one Makefile; three tester files: inst.mach, test\_file1.mach, test\_file2.mach.

**Description:**

BitCode.cpp: This file is a main file that executes the specific codes that reads in an input binary file and call other class to do the work.

BitCodeParser: This class gets a file read in and checks the format of the codes in this input file. If the format is not correct then it will break and go back to BitCode.cpp. If the format is correct it will separate it into different operands, use the operands to form an instruction, and finally do the decoding work.

Instruction: This class stores the information of an instruction for the MIPS. It contains register fields rs, rt, rd and an immediate field as operands.

Opcode: This class stores the information of opcode value for different instructions in MIPS. It shows the valid instructions that could be used, the fields that an instruction has and the position of these fields.

RegisterTable: This class stores the information of the name and location of all the registers. Given an index of register we can get the name of that register with RegisterTable.

Input: There are 2 more input test files besides the given one, each of them has about 8 lines with codes in the. Because BitCodeParser will automatically remove the content after the 32-bits codes, the instructions after the 32-bits codes will not cause problems. The first file, test\_file1, tests and makes sure that the program handles negative values in immediate. The second file, test\_file2, lack one digit of binary in the fifth instruction and should print out a message to report the error.

**Organization**

First the BitCode file parses in a readable file, then the BitCodeParser gets the file and checks if each line of codes in the file is in correct format. If not the program will be stopped and error message will occur. If the format is correct BitCodeParser will try decode the code. For each 32-bits codes, the BitCodeParser will first read its first 6 char as opcodes, then separate the later string into different operands, according to the type of opcode. If the opcode is RTYPE, the last 6 char will not be stored as operands, but the funct field.

Then the Opcode will try read the opcode (and funct field if it has) and find the corresponding instruction. Anytime operands or opcodes in invalid format appears, the program will still stop. Then BitCodeParser continue to convert the separated operands into proper types (Register and int), and form a new Instruction variable. With an Instruction class, information is enough for a decoding work. In decode in BitCodeParser, the method will first tell the type of instruction, then call the corresponding helper method. For each index of register operand, RegisterTable will find its string-type name. Finally, the decode method will combine the opcode and each operand with proper connection and return it back to BitCode. BitCode will call cout for the MIPS instruction and then let BitCodeParser parse in the next line until end.

**Adding new Instruction**

To add a new instruction, you just need to modify the Opcode class: Add the capital letter name in enum Opcode in OpcodeTable.h, then add the instruction with small letter name in Opcode.cpp and declare its name, type, operands, operands’ positions etc.

Notice: For instruction with immediate being label, variable immLable needs to be set to be true, and for instruction with immediate added to a register, which need brackets for the register, variable immBracket needs to be set to be true.