Alex Lundin

CS/CE 3340

Fall 2016

Final Report

Notes about operation:

When I output the immediate value for I type, I did not use a sign extension. So it can only handle positive values for the immediate section of the instruction.

When I output the pseudoaddress for j type instructions it is the string of binary 0 and 1’s.

1.) Problem statement

This program demonstrates how instructions are formatted in mips.

Having a good grasp of opcodes and function codes is very important to assembly.

Knowing how to implement a disassembler removes the obscurity of how the computer handles 0’s and 1’s. It’s a lot clearer now.

2.) Approach to solution

I started this assignment by learning how to read text files.

Next, I researched how mips handles newlines in text files, this was important for the main loop

Finally I built sub functions to convert binary strings to integer decimals, integers to strings (itoa) and one to write strings to text files (fputs).

This really made me appreciate the high level languages, I didn’t realize how tedious simple things in those languages are like converting integers to strings.

3.) Solution Description

These paragraphs will give a brief overview of the method used in the Final program. The comments at the top of the program describe the register use, in detail for the main loop. The comments also give a more granular description of the three labels itype, rtype and jtype. These three labels format the output for each instruction type with computedgoto functions.

First the main loop reads the entire contents of a text file called “machinecode.text” to a buffer called allinstructions.

Next the main loop jumps to allinstructionsetup which saves the address of this buffer, allinstructions, to $s0 and $s1. The register use follows: $s0 is a untouched saved copy of the starting address and $s1 is used by the loop to move through the buffers contents. Finally, allinstructionsetup loads 0 into $s2, which will be used later to move past new line characters in the buffer.

Now the next several functions are setup and getter functions. First, they setup the current instruction then peel off the corresponding bits to the opcode, rs, rt, rd, sa and function buffers. They all use the same register convention and are pretty straight forward.

Next, the endgetters function converts the opcode with the binarytodecimal function and uses this value in coordination with the computedgoto to find the correct opcode label. There is a more detailed description of the goto in the comments. Each of the 64 opcodes has its own label.

Next, the computedgoto will choose what to do depending on the opcode. For rtypes, it will not write the opcode string and jump to the rtype label which writes the rest. For itypes, it will setup register $a1 with the opcode string then jump to the itype label to write the rest. For jtypes it will setup register $a1 with the opcode string then jump to the jtype label to write the rest. There is also error handling for invalid opcodes, they will write the invalid string then exitgoto.

Each of the type labels (rtype, itype and jtype) follows the same structure. In order to keep the code clean, there are sublabels in each of the types depending on what needs to be done. There are setups, getters and writers. The following description will be an overview of what this section of code accomplishes (rtype, itype and jtype). Also there are subfuctions used such as decimaltobinary, itoa and fputs(for writing strings), computedgoto2 (register names) and computedgoto3 (function codes).

First, they get any remaining information they might need, like the function code for rtypes, immediate values for itypes and psuedoaddresses for jtypes. They do this with the same setup and getter labels as the main loop. Then they convert any function or register values with binary to decimal. Next, they use the converted function value (rtype only) with computedgoto3 to write the function code. After that, they use the converted rs, rt and rd values (rtype and itype only) with computedgoto2, to write the correct register name. Finaly they output the immediate and psuedoaddresses (itype and jtype only). Now the immediate address had to be converted from a string to a integer with the itoa function. This value does not account for negative numbers, I did not use a sign extend function. So the output will be an incorrect integer for any negative immediate values in the instruction. The pseudoaddress will be output as a binary string of 0 and 1’s.

Finally, each of the type labels (iwrite, rwrite and jwrite) will call the exitwriter label which starts over at currentinstructionsetup.

Screenshots



