Assignment 1 Documentation

Hyeonwook Kim

118010488

The MIPS Assembler

This programming project converts MIPS assembly language into binary machine code. Each line in a MIPS instruction can be classified into three categories: R type, I type and J type instructions. The program reads each line in the MIPS code, determines the type of instruction and converts accordingly. The instructions also contain comments, which are removed.

For R Type instructions, the first six digits are always 0. The remaining 26 digits are determined by the type of R instruction (alf). It has 3 registers, a shift amount and a 6 digit function.

opcode (6)	rs (5)	rt (5)	rd (5)	sa (5)	function (6)

I type instructions also have a 6 digit opcode, but they are not like R instructions and vary depending on the command. I type instructions only have 2 registers, and it is followed by a 16 bit immediate. When it comes to certain instructions, the 16 bit immediate is used as a target address field. This is encoded based on the relative distance between the address of the instruction and the target address. Multiplying the 16 bit immediate by 4 and adding the current address will result in the value of the target address.

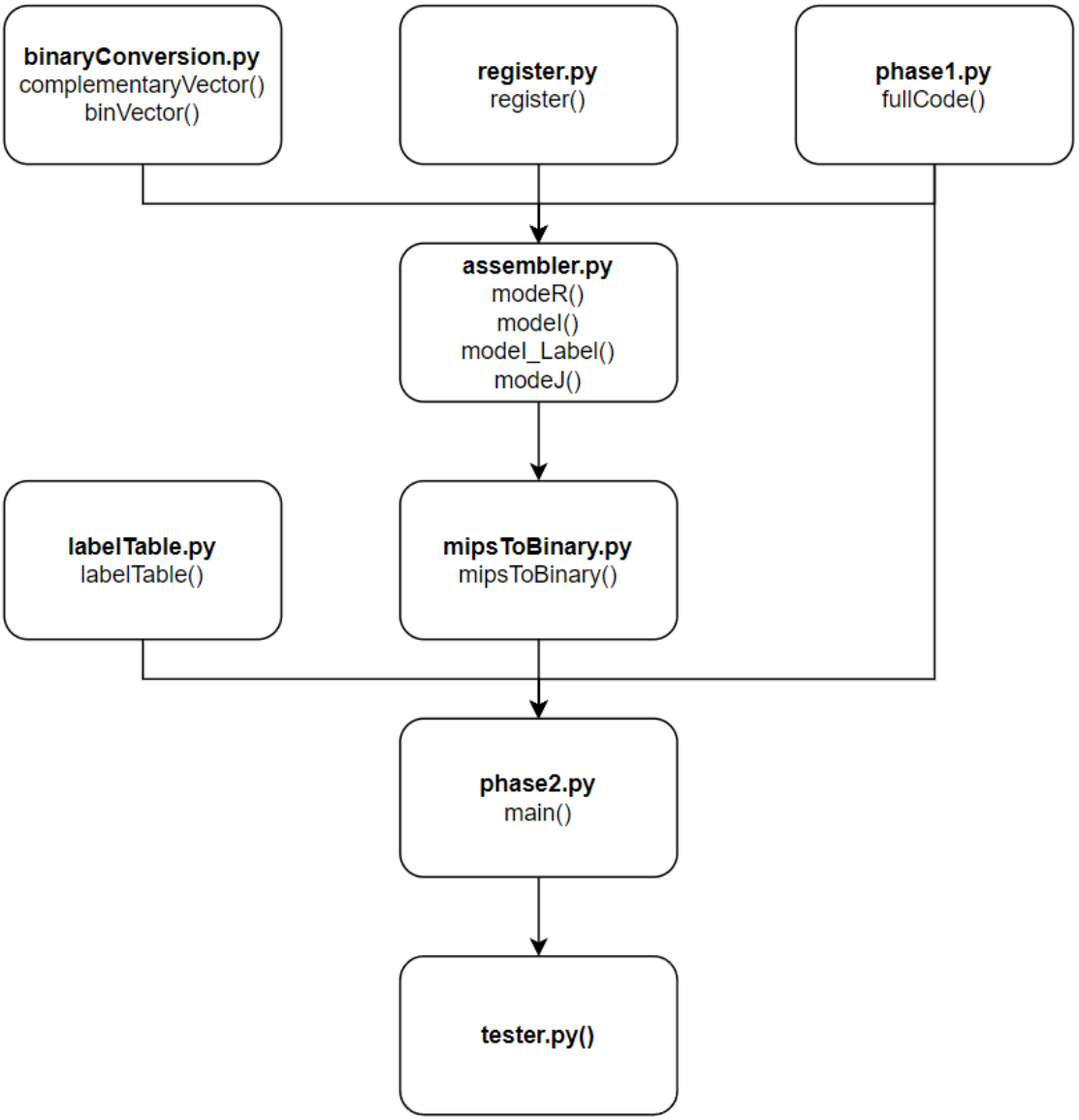
opcode (6)	rs (5)	rt (5)	immediate (16)

A J type instruction contains a 6 digit opcode and a 26 bit target address field. Unlike I type commands, J instructions encodes its target address based only on the label address. Taking the 26 bit immediate and multiplying it by 4 gives the target address.

opcode (6)	target (26)

Functions

Python file dependencies



complementaryVector( ) binaryConversion.py

Takes a list of 0's and 1's and gives the 2's complement form of the list.

binVector( ) binaryConversion.py

Takes 2 integers *value* and *n*. Converts value into its binary form with n bits and puts them in a list. if value is a negative number, *complementaryVector()* is used to give the 2's complement.

register( ) register.py

Takes a string input and reads it. Depending on the input, the output will be one of the 32 MIPS register codes written in vector form.

fullCode( ) phase1.py

Takes a file name or directory as input. The fullCode() function reads the file and discards unnecessary information such as comments or empty spaces. The output is a list where each item is a MIPS command in the file.

modeR( ) assembler.py

The modeR() function takes an R type instruction written with string format as input, and converts it into machine code. Uses *register()* to get the binary code for the registers, and uses *binVector()* for shift amount.

modeI( ) assembler.py

The modeI() function takes an I type instruction written with string format as input, and converts it into machine code. Uses *register() to get the binary code for the registers.*

modeI_label( ) assembler.py

The modeI_label() function takes an I type instruction that uses labels, and converts it into machine code. The 16 bit immediate is calculated by taking the position of the target address in the list, and subtracting it by the current address. Because of this, it also takes the current position of the command and the full code list as input.

modeR( ) assembler.py

The modeR() function takes an R type instruction and converts it into machine code. The 26 bit immediate is calculated by taking the position of the target address in the list and adding 0x400000. Because of this, it also takes the full code list as input.

labelTable( ) labeltable.py

The labelTable() function takes the full code vector as input and gives the position of all labels in the code. The output is stored in a 2xn list, where the first row contains the list of all labels, while the second row contains the list of all addresses.

mipsToBinary( ) mipsToBinary.py

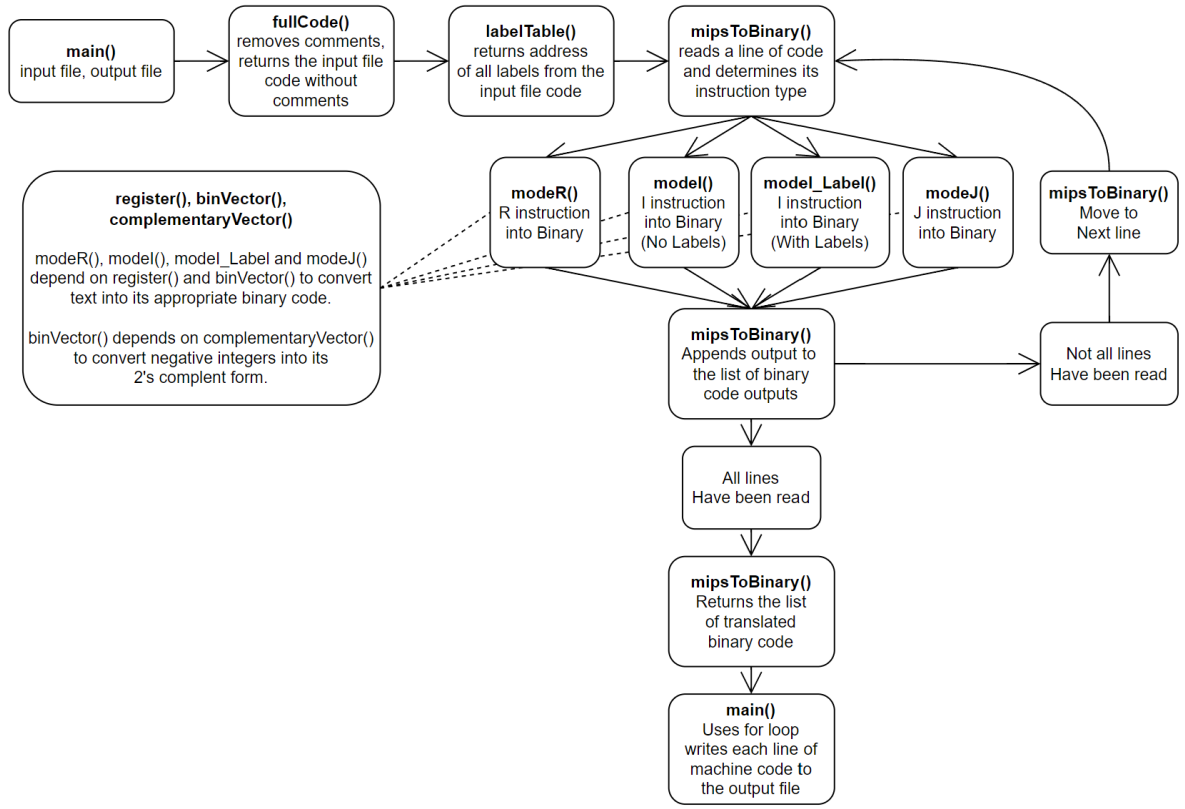
The mipsToBinary() function converts MIPS code into binary. It takes a list of MIPS assembly instructions, reads it line by line, and uses *modeR()*, *modeI()*, *modeI_label()* and *modeJ()* accordingly. The output is a list of 32 bit machine code translated from the MIPS instructions.

main( ) phase2.py

Relies on *fullCode()*, *labelTable()* and *mipsToBinary()*. The function takes an input file and output file as input. The function reads the input file, removes unnecessary text, converts relevant MIPS code to binary and writes the results to the output file. The function also prints the initial code with comments removed, the list of label addresses and the converted machine code.

Workflow

Workflow Map



General Workflow

After tester.py takes the names of the input file, output file and the expected output file, the main() function from **phase2.py** is activated. The main() function takes the input and output files as input, and it activates the fullCode(), labelTable() and mipsToBinary() functions.

The fullCode() function is taken from **phase1.py**, and takes the input file as input. Upon activation, the function reads each line one by one. For each line, a for loop is activated that removes "\n" and "\t". Then if it finds a "#", which indicates a comment, text starting from the "#" character is deleted from the string. After that, extra spaces are removed from the code, and if the resulting string is not empty, it is added to a list. The resulting output is a list that contains a list of MIPS code, with comments or unneeded information removed.

The main() function then activates labelTable(), located in **label_Table.py**. This function takes the MIPS code vector that was obtained from the fullCode() function as input. The labelTable() function looks for any item that has ":". When detected, the string is considered a label and added to a list called "label". After all the labels have been identified, the function searches for the address of each label. This is done by counting each line of code that doesn't have ":" and adding the

numbers until the for loop finds the target label. Then the number is multiplied by 4, added by 4194304 and converted into hexadecimal. The resulting number is then added to a list called "address". This process is done for all labels. Finally, the function prints the list of all labels and addresses, and returns a list that contains the list of all labels and addresses.

Afterwards, the mipsToBinary() function is activated from **mipsToBinary.py**. This function takes the vector output from fullCode() and converts each line of instruction into a 32 bit machine code. Upon activation, the function reads each line of code in the vector input. For each line, the function first looks for its operator. Once the operator is determined, the function determines what kind of instruction the MIPS code is. Depending on the type of operator, the function then activates modeR, modeI, modeI_label or modeJ, which converts the code into binary for R type, I type, I type with labels and J type instructions. All 4 functions are imported from **assembler.py**.

The modeR() function converts an R type instruction into machine code. The function first identifies its operator, and converts the instruction into machine code based on its operator type. The operators are classified into different lists, and if the operator found in the command is found in one of the lists, the function locates the inputs of the relevant registers and uses register() from **register.py** to convert them into machine code. binVector() from **binaryConversion.py** is also used to determine the binary code for the shift amount. The operand is always a 6 digit 0, since all R type instructions have an opcode of 000000. Finally, a list that contains the function binary code is called based on the operator type, and the output is stitched together and output in string form.

The modeI() function acts similarly to modeR(), but the lists for functions is replaced with lists of opcodes, since opcodes are determined by its operand and I type commands only feature 2 registers and a 16 bit immediate. The immediate is calculated by taking the number from the input code and running binVector().

modeI_label() is almost identical in its workflow as modeI() except the way it handles the 16 bit immediate. Because of this, the modeI_label function also takes the full code from the fullCode() function and the position of the current instruction as additional inputs. The function takes the current address and the target address and subtracts them to get its relative distance in the list. Then binVector() is used to output the 16 bit immediate. After calculating the opcode and registers in binary form, the 32 bit binary code is output in string form.

modeJ() converts J type instructions into binary code. The 26 bit immediate is calculated by finding the position of the target address in the full code list, and

adding 0x400000. The result is converted into binary and stitched with the opcode based on the operator, and the 32 bit binary code is output.

Using the 4 functions mentioned above, the mipsToBinary() function converts every command in the input code to its 32 bit binary equivalent. The machine code is put in a list, then output. After mipsToBinary() is complete, the main() function runs a for loop, and for every line in the output binary code, it is written to the designated output file. After printing the fully converted binary code, the output file is closed and the main() function ends.