Computer Organization: MIPS Assembler

Anna Tang
3180300155

Introduction

This program is a command line-based MIPS assembler and disassembler, coded in Python. No executable is generated, and everything runs on the terminal. Please refer to the readme.txt as well.

Usage

The user must first specify whether they want to assemble or disassemble a file in the command-line prompt. It then takes in an input file and an output file as the arguments. The input file must be in your CURRENT directory for the assembler to detect input. This assembler also supports all file formats (e.g. .txt, .coe, .asm, .hex).

In the case where an input error, an error prompt will appear and terminate the program. If the file successfully assembles or disassembles, the program will terminate. For the following example, user input is bolded to differentiate from program prompts.

Successful Run:

```
C:\Users\Anna\Desktop\Comp Org\Assembler> python assembler.py
Welcome to the MIPS assembler and disassembler.
Type A to assemble and D to disassemble: a
Input the name of the file to be assembled. For example, test.asm: t.asm
Input the name of the file you want the hex code to be outputted in.
If the file does not exist in your directory, a new one will be created:
h.hex
Assembly successful. Please check h.hex for the assembled instructions.
```

Error:

```
C:\Users\Anna\Desktop\Comp Org\Assembler> python assembler.py
Welcome to the MIPS assembler and disassembler.
Type A to assemble and D to disassemble: d
Input the name of the file to be disassembled. For example, test.hex:
file.hex
Oops! That file does not exist. Please double check the file name and try
again.
```

Input Specifications

The input file does not support multiline labels. Please ensure that labels are on the same lines as the instructions. Any white space in the input file will be removed in the output file. Comments are denoted by the '#' character, and the end of an instruction is denoted by a semicolon (';'). For intermediate values, they are to be in decimal format.

Not Supported:

```
addi $a0, $zero, 1;
j next;
next:
j skip1;
add $a0, $a0, $a0;
skip1:
j skip2;
add $a0, $a0, $a0;
add $a0, $a0, $a0;
skip2:
j skip3;
loop:
add $a0, $a0, $a0;
add $a0, $a0, $a0;
add $a0, $a0, $a0;
skip3:
j loop;
```

Supported:

```
# mutiline
# commennts
# are
# ok!

addi $a0, $zero, 1;
j next;
next: j skip1; #next is a label
add $a0, $a0, $a0;
skip1: j skip2;
add $a0, $a0, $a0;
add $a0, $a0, $a0;
skip2: j skip3;
loop: add $a0, $a0, $a0;
add $a0, $a0, $a0;
add $a0, $a0, $a0;
skip3: j loop;
```

Output Specifications

The machine code generated by the assembler will be in hexadecimal format. If the program is in assembler mode, the generated file can be fed back into the disassembler. The disassembler prints out the PC values as comments, next to its respective instruction. Branch and jump addresses are printed in hexadecimal, and intermediate/offset values are preserved in decimal. For reference, the PC addresses start at 0x00000000. Outputs of the example above are shown below.

Assembler:

```
20040001
08000006
08000008
00842020
0800000b
00842020
00842020
0800000f
00842020
00842020
00842020
0800000c
```

Disassembler:

```
addi $a0,$zero,1; #0x00000000
j 0x00000018; #0x00000004
j 0x00000020; #0x00000008
add $a0,$a0,$a0; #0x0000000C
j 0x0000002C; #0x00000010
add $a0,$a0,$a0; #0x00000014
add $a0,$a0,$a0; #0x00000018
j 0x0000003C; #0x0000001C
add $a0,$a0,$a0; #0x00000020
add $a0,$a0,$a0; #0x00000024
add $a0,$a0,$a0; #0x00000028
j 0x00000030; #0x0000002C
```

Supported Instructions for Assembler and Disassembler

- `add`
- `sub`
- `and`
- `or`

- `addi`
- `andi`
- `ori`
- `lw`
- `sw`
- `srl`
- `sll`
- `lui`
- `slt`
- `slti`
- `beq`
- `bne`
- `j`
- `jal`
- `jr`
- `nor`
- `xor`
- All 32 registers are supported.