Tommy Prusak
CMPE 220-02
Lab 5

Description of program:

This program takes a .asm file as an input and outputs the MIPS instructions within this .asm file in binary and hex (machine code). The way we do this is first, we ignore all lines in the .asm file until we reach the .text directive. After that, we are going line by line and splitting it up to have all components separate (instruction, value, register, etc.). Then we first find the correct instruction in binary by having an already-made array of instructions with the same index as another array that has the corresponding binary value of that instruction. I have split up the arrays by whether it's an R type or I type instruction, and whether it's a logical instruction, arithmetic instruction, branch instruction, etc. This way we know exactly how we have to construct the final 32-bit word after finding the correct instruction. Then, if we have addresses in the line, I have a function that converts the address to the correct binary value, with 5 bits. Then if we have a constant value, I have a function that converts that to binary, while making sure its 16 bits. We find all of these different pieces in the current line we are looking at and then based off of what type it is and what type of instruction it is, I correctly concatenate all the different binary values we found so it's correct machine code. I print out that value and then lastly print out the hexadecimal equivalent to it.

Analysis of program:

I think that my program could definitely be more efficient and more readable. I did split up big sections of code into functions, but my main function is still somewhat large and cluttered. It is also somewhat hard to read without the comments. Also, due to the nature of how I have 7 different arrays containing instructions, going through and doing calculations based off those and the corresponding arrays of the instructions in binary creates a lot of if statements. Another thing too is that I have a lot of flags I use that are global, which makes things pretty messy inside the main function and other functions for keeping track of everything. When it comes to the smaller functions, they are very readable and efficient, but the bigger ones are not great. Another thing is that I re-use a lot of the same code that could be put into functions. Overall, it is quite messy but works as intended and covers all of the different instructions correctly.

Inputs files and outputs:

lab5.asm

```
1  #Tommy Prusak
2  #CMPE220-02
3
4  .data
5  .word
6  .text
7  add $t0, $t1, $t2
8  addi $t0, $t1, 30352
9  and $t0, $t1, $t2
10 andi $t0, $t1, 19
11 beq $t0, $t1, 19
12 lw $t1, 8($t2)
13 jr $t1
```

```
R-format: 00000001001010100100000000100000
0x00000000: 0x012A4020


I-format: 00100001001010000111011010010000
0x00000004: 0x21287690


R-format: 00000001011010100010000000100100
0x00000008: 0x016A2024


I-format: 00110001110010000000000000010011
0x0000000C: 0x31C80013


I-format: 00010001000110100000000000010011
0x00000010: 0x111A0013


I-format: 10001101010010010000000000001000
0x00000014: 0x8D490008


R-format: 00000001001000000000000000001000
0x00000018: 0x01200008
```

lab5-2.asm

```
1  #Tommy Prusak
2  #CMPE220-02
3
4  .data
5  .word
6  .text
7  sltiu $s0, $zero, 4
8  bne $s0, $s1, $s2
9  addi $s1, $s1, 1
10 sub $s1, $s1, $s0
11 jr $ra
12 sltu $t6, $s3, $a2
13 sw $t5, 7($t2)
14 nor $s1, $t4, $t0
15
```

```
I-format: 00101100000100000000000000000100
0x00000000: 0x2C100004


I-format: 00010110000100010000000000000000
0x00000004: 0x16110000


I-format: 00100010001100010000000000000001
0x00000008: 0x22310001


R-format: 00000010001100001000100000100100
0x0000000C: 0x02308824


R-format: 00000011111000000000000000001000
0x00000010: 0x03E00008


R-format: 00000010011001100111000000101011
0x00000014: 0x0266702B


I-format: 10101101010011010000000000000111
0x00000018: 0xAD4D0007


R-format: 00000001100010001000100000100111
0x0000001C: 0x01888827
```