Description of the Program:

The purpose of this program was to make a low-level MIPS assembler. Essentially, we were to take MIPS instructions and convert them to binary and print all of this out. Since binary is the language of the machine and MIPS instructions are just an abstraction from that the two can be directly translated.

Algorithm Analysis:

The way I translated the MIPS instructions was simple. I first read a line of the text file using fgets method. I then take that string and use strtok to split up the individual parts. So, the input add $t0 $t1 $t1, gets split up and stored into a 2D string array and looks like this:

{ {'a', 'd', 'd'}, {'$', 't', '0'}, {'$', 't', '1'}, {'$', 't', '1'} } //this is for R-Format

I then use these parts to determine the format (R or I), then I use the next 3 sections to determine the decimal values of rs, rt, and rd (if needed). I then convert these numbers to binary and print them out. I also had to make a special exception for my bne command, and calculate the target address based on the current PC + 4 + the offset read from the text file.

Input:

```
add $t0 $t1 $t0
add $t7 $t6 $t4
addi $t1 $t2 10
sub $t1 $t2 $t1
sub $t5 $t0 $t2
andi $t3 $t2 1
sll &t1 $t5 4
addi $t6 $t3 42
bne $t1 $t5 8
sll $t7 $t1 9
```

Output:

```
0x00000000: add $t0 $t1 $t0
    (R) 000000 01001 01000 01000 00000 100000
0x00000004: add $t7 $t6 $t4
    (R) 000000 01110 01100 01111 00000 100000
0x00000008: addi $t1 $t2 10
    (I) 001000 01010 01001 0000000000001010
0x0000000c: sub $t1 $t2 $t1
    (R) 000000 01010 01001 01001 00000 100010
0x00000010: sub $t5 $t0 $t2
    (R) 000000 01000 01010 01101 00000 100010
0x00000014: andi $t3 $t2 1
    (I) 001100 01010 01011 0000000000000001
0x00000018: sll &t1 $t5 4
    (R) 000000 00000 01101 01001 00100 000000
0x0000001c: addi $t6 $t3 42
    (I) 001000 01011 01110 0000000000101010
0x00000020: bne $t1 $t5 8
    (I) 000101 01001 01101 0000000000001000 (Branch Address: 0x0000002c)
0x00000024: sll $t7 $t1 9
    (R) 000000 00000 01001 01111 01001 000000
```

```
#include <stdio.h>
#include <string.h>

int PC = 0;
int main(void) {
  FILE* input = fopen("mips.txt", "r");

  char line[50];
  char* parts[4];

  while(fgets(line, 50, input)) {
    printf("0x%08x: ", PC);
    printf("%s", line);

    split(line, parts);
    translate(parts);
    PC +=4;

  }
}
split(char* line, char** parts) {
  char* temp = strtok(line, " ");

  int i = 0;
  while(temp != NULL) {
    parts[i] = temp;
    temp = strtok(NULL, " ");
    i++;
  }
}

translate(char **parts) {
  //parts[0] = function, parts[1] = dest, parts[2] = first src, parts[3] = sec
src/imm
  int opcode;
  int rd;
  int rs;
  int rt;
  int shamt;
  int funct;
  int immediate;
  int isBranch = 0;

  //I-format
  if(strcmp(parts[0], "addi") == 0) {
    opcode = 8;
    rd = 8 + atoi(&parts[1][2]);
    rs = 8 + atoi(&parts[2][2]);
    immediate = atoi(parts[3]);
    printIFormat(opcode, rs, rd, immediate, isBranch);
```

```
  }
  else if(strcmp(parts[0], "andi") == 0) {
    opcode = 12;
    rs = 8 + atoi(&parts[2][2]);
    rd = 8 + atoi(&parts[1][2]);
    immediate = atoi(parts[3]);
    printIFormat(opcode, rs, rd, immediate, isBranch);
  }
  else if(strcmp(parts[0], "bne") == 0) {
    isBranch = 1;
    opcode = 5;
    rs = 8 + atoi(&parts[1][2]);
    rd = 8 + atoi(&parts[2][2]);
    immediate = atoi(parts[3]);
    printIFormat(opcode, rs, rd, immediate, isBranch);
  }
  //R-Format
  else if(strcmp(parts[0], "add") == 0) {
    opcode = 0;
    rs = 8 + atoi(&parts[1][2]);
    rt = 8 + atoi(&parts[2][2]);
    rd = 8 + atoi(&parts[3][2]);
    shamt = 0;
    funct = 32;

    printRFormat(opcode, rs, rt, rd, shamt, funct);
  }
  else if(strcmp(parts[0], "sub") == 0) {
    opcode = 0;
    rs = 8 + atoi(&parts[1][2]);
    rt = 8 + atoi(&parts[2][2]);
    rd = 8 + atoi(&parts[3][2]);
    shamt = 0;
    funct = 34;

    printRFormat(opcode, rs, rt, rd, shamt, funct);
  }
  else if(strcmp(parts[0], "sll") == 0) {
    opcode = 0;
    rs = 8 + atoi(&parts[1][2]);
    rt = 0;
    rd = 8 + atoi(&parts[2][2]);
    shamt = atoi(parts[3]);
    funct = 0;

    printRFormat(opcode, rs, rt, rd, shamt, funct);
  }
}

printIFormat(int opcode, int rs, int rd, int immediate, int isBranch) {
```

```
  int forBranch = immediate;

  char* opcodeBinary = malloc(6 * sizeof(char));
  for(int i = 5; i >= 0; i--) {
    opcodeBinary[i] = (opcode % 2) + '0';
    opcode /= 2;
  }
  printf("\t(I) %s ", opcodeBinary);

  char* rsBinary = malloc(5 * sizeof(char));
    for(int i = 4; i >= 0; i--) {
      rsBinary[i] = (rs % 2) + '0';
      rs /= 2;
    }
    printf("%s ", rsBinary);

  char* rdBinary = malloc(5 * sizeof(char));
  for(int i = 4; i >= 0; i--) {
    rdBinary[i] = (rd % 2) + '0';
    rd /= 2;
  }
  printf("%s ", rdBinary);

  char* immediateBinary = malloc(16 * sizeof(char));
  for(int i = 15; i >= 0; i--) {
    immediateBinary[i] = (immediate % 2) + '0';
    immediate /= 2;
  }

  if(isBranch == 1) {
    int newPC = PC + 4 + forBranch;
    printf("%s ", immediateBinary);
    printf("(Branch Address: 0x%08x)\n", newPC);
  } else {
     printf("%s\n", immediateBinary);
  }
}

printRFormat(int opcode, int rd, int rs, int rt, int shamt, int funct) {
  char* opcodeBinary = malloc(6 * sizeof(char));
  for(int i = 5; i >= 0; i--) {
    opcodeBinary[i] = (opcode % 2) + '0';
    opcode /= 2;
  }
  printf("\t(R) %s ", opcodeBinary);

  char* rsBinary = malloc(5 * sizeof(char));
    for(int i = 4; i >= 0; i--) {
      rsBinary[i] = (rs % 2) + '0';
```

```
      rs /= 2;
    }
    printf("%s ", rsBinary);

  char* rtBinary = malloc(5 * sizeof(char));
  for(int i = 4; i >= 0; i--) {
    rtBinary[i] = (rt % 2) + '0';
    rt /= 2;
  }
  printf("%s ", rtBinary);

  char* rdBinary = malloc(5 * sizeof(char));
  for(int i = 4; i >= 0; i--) {
    rdBinary[i] = (rd % 2) + '0';
    rd /= 2;
  }
  printf("%s ", rdBinary);

  char* shamtBinary = malloc(5 * sizeof(char));
  for(int i = 4; i >= 0; i--) {
    shamtBinary[i] = (shamt % 2) + '0';
    shamt /= 2;
  }
  printf("%s ", shamtBinary);

 char* functBinary = malloc(6 * sizeof(char));
  for(int i = 5; i >= 0; i--) {
    functBinary[i] = (funct % 2) + '0';
    funct /= 2;
  }
  printf("%s\n", functBinary);
}
```