# Lab6: LC-3 Excutor

## 1 pseudocode

```
In the main function:

    Set intCode[65537][16] as memory array.
        REG[8] as registers array.
        CC = z

    Read input into Code[0] as ORIG. Set i as PC.
    Read the remaining input lines into Code[line] and convert them to binary in
intCod
    Call decodeWords(intCode){
        while(not HALT){
            switch(opcode){
                    ADD, AND, NOT ...
            }
        }
    }

    Print the final values of registers

the functions in LC-3 as following:

Def judgeCC(num):
    if num == 0:
        CC = 0
    else if 1 <= num <= 32767:
        CC = 1
    else if 32768 <= num <= 65535:
        CC = -1

Def function ADD(int intCode[][16]):
    imm5 = (intCode[i][10] == 1) ? binary_to_decimal_5(intCode) : 0
    SR2 = (intCode[i][10] == 0) ? intCode[i][13]*4 + intCode[i][14]*2 +
intCode[i][15]*1 : 0
    REG[DR] = REG[SR1] + REG[SR2](or imm5)
    judge_CC(REG[DR])
    i++

Def function AND(int intCode[][16]):
    imm5 = (intCode[i][10] == 1) ? binary_to_decimal_5(intCode) : 0
    SR2 = (intCode[i][10] == 0) ? intCode[i][13]*4 + intCode[i][14]*2 +
intCode[i][15]*1 : 0
    REG[DR] = (intCode[i][10] == 1) ? (REG[SR1] & imm5) : (REG[SR1] & REG[SR2])
    judge_CC(REG[DR])
    i++

Def function NOT(int intCode[][16]):
    for p in range(15, -1, -1):
        bin[k] = (REG[DR] % 2 == 0) ? 1 : 0
```

```
        REG[DR] = REG[DR] // 2
    for p in range(16):
        bin[k] = (bin[p] == 1) ? 0 : 1
        REG[DR] += bin[p] * 2^p
    judge_CC(REG[DR])
    i++

Def function LEA(int intCode[][16]):
    REG[DR] = i + binary_to_decimal(intCode) + 1
    i++

Def function ST(int intCode[][16]):
    store_adr = i + binary_to_decimal(intCode) + 1
    store_num = REG[SR]
    for p in range(15, -1, -1):
        intCode[store_adr][p] = store_num % 2
        store_num /= 2
    i++

Def function STI(int intCode[][16]):
    store_adr = i + binary_to_decimal(intCode) + 1
    store_adr_adr = binary_to_decimal_16(intCode, store_adr)
    store_num = REG[SR]
    for p in range(15, -1, -1):
        intCode[store_adr_adr][p] = store_num % 2
        store_num /= 2
    i++

Def function STR(int intCode[][16]):
    store_add = REG[BaseR] + binary_to_decimal_6(intCode)
    for p in range(15, -1, -1):
        intCode[store_add][p] = REG[SR] % 2
        REG[SR] /= REG[SR]
    i++

Def function LD(int intCode[][16]):
    read_from = i + binary_to_decimal(intCode) + 1
    for j in range(15, -1, -1):
        REG[DR] += intCode[read_from][j] * 2^j
    judge_CC(REG[DR])
    i++

Def function LDI(int intCode[][16]):
    address = i + binary_to_decimal(intCode) + 1
    address_data = binary_to_decimal_16(intCode, address)
    REG[DR] = binary_to_decimal_16(intCode, address_data)
    judge_CC(REG[DR])
    i++

Def function LDR(int intCode[][16]):
    store_add = REG[BaseR] + binary_to_decimal_6(intCode)
    REG[DR] = binary_to_decimal_16(intCode, store_add)
    judge_CC(REG[DR])
    i++
```

```
Def function JSR(int intCode[][16]):
    REG[7] = i + 1
    if intCode[i][4] == 1:
        i = i + binary_to_decimal_11(intCode) + 1
    else:
        i = REG[BaseR]

Def function JMP(int intCode[][16]):
    i = REG[BaseR]

Def function BR(int intCode[][16]):
    jmp_to = i + binary_to_decimal(intCode) + 1
    if (intCode[i][4] == 1 and CC < 0) or
       (intCode[i][5] == 1 and CC == 0) or
       (intCode[i][6] == 1 and CC > 0) or
       (intCode[i][4] == 1 CC < 0 and intCode[i][5] == 1 CC == 0 ) or
       (intCode[i][4] == 1 CC < 0 and intCode[i][6] == 1 CC > 0 ) or
       (intCode[i][5] == 1 CC == 0and intCode[i][6] == 1 CC >0 ):
        i = jmp_to
    else:
        i++
```

- The main structure of this code is each branch of switch, which performs different operations under different OPCODEs. For convenience, the arrays of registers and memory are global variable arrays.

## 2 Essential Parts

- ```
  void judge_CC(int num){
      if (num == 0){
          CC = 0;
      }
      else if (num >= 1 && num <= 32767){
          CC = 1;
      }
      else if ((num >= 32768 && num <= 65535)){
          CC = -1;
      }
  }
  ```

  Used to determine the value of CC, where -1, 0, and 1 represent n, z, and p respectively.

- ```
  int binary_to_decimal(int intCode[][16]){
      int decimal = 0;
      if (intCode[i][7] == 0){
          for (int j = 15, k = 0; j >= 8; j--, k++){
              decimal += intCode[i][j] * pow(2, k);
          }
      }
      else{
          for (int j = 15, k = 0; j >= 7; j--, k++){
              if (intCode[i][j] == 0){
                  intCode[i][j] = 1;
  ```

```
            }
            else{
                intCode[i][j] = 0;
            }
            decimal += intCode[i][j] * pow(2, k);
        }
        decimal++;
        decimal = -decimal;
    }
    return decimal;
}
```

Used to convert the PCoffset in the memory to an integer. The complement of the two's complement is the source code. Therefore, if the first bit is 1, invert it and add 1 to get the absolute value. Just add the sign.

- ```
  REG[DR] = REG[SR1] + imm5;
  if (REG[DR] < 0){
  REG[DR] += 65536;
      }
  else if (REG[DR] >= 65536){
  REG[DR] -= 65536;
      }
  ```

I am not using unsigned short data, so I need to convert negative numbers or numbers exceeding 65535 into 0 to 65535.

- ```
  scanf("%s", Code[0]);
  for (int l = 0; l < 16; l++){
  intCode[0][l] = Code[0][l] - '0';
      }
  ORIG = binary_to_decimal_16(intCode, 0);
  line = ORIG;
  while (scanf("%s", Code[line]) != EOF){
  for (int l = 0; l < 16; l++){
      intCode[line][l] = Code[line][l] - '0';
          }
      line++;
      }
  ```

First use the SCANF function to enter the starting address of the program, set the PC as the starting address, and read the program.

Since %s reads characters, you need to subtract '0' .