PART1. ALGORITHM

1.how my program handle with input as well as storage data and instructions:

I use a register array the size of which is 7 to store data in the registers. I use a line array to handle the input, that is, the instructions. And I use a memory dictionary to store other data, its keys are addresses and the values are data at that address. Obviously, I store the instructions and memory separately. It's different from LC-3. It's convenient but not a good way.

2.Execute instructions:

For every line of instructions, I pass ir to the function istDic() in which program will jump to different function (ADD,AND……) according to the instruction. It return 0 when the instruction is HALT to tell the main program that it should stop. At other time, it returns 0, so the program will continue to execute instructions.

3.Then, the instruction is executed.

Many instructions（ADD, AND, JSR）have two mode. As for them, their mode should be examined first.

Besides, there are some functions to help execute the instruction and make the code more readable.

①extension:

SEXT execute extension on signed numbers while ZEXT execute extension on unsigned numbers.

②setCC

Some instructions will change the condition code. So after they are executed, program should set the condition code according to result calculated.

Condition code is important to BR instruction.

4.Stop the program and print registers.

If the istDic() function return 0, the execution should be stopped. Finally, the program are asked to print the registers from R0 to R7.

PART2. Essential Parts

High-level language: Python

1.Handle with instructions:

My algorithm to handle the input instructions and define the pc:

```
lines = []
for line in sys.stdin:
    lines.append(line[:-1])
orig = int(lines[0], 2)      //the first line is .orig (start adress), so pop.
```

```
    lines.pop(0)
    pc = orig

    Execute lines:
    while True:
        ir = lines[pc - orig]
        pc += 1
        if (istDic() == 0):
            break
```

As the codes show, for every instruction line, I use “pc-orig to fetch it in the array. And pc increments after it.

2. Select next instruction

```
def istDic():
    ist = ir[0:4]
    res = 1
    if ist == '0001':
        ADD()
    elif ist == '0101':
        AND()
(……)
    else:
        res = 0     // if it is HALT, stop the program.
    return res
```

3. Extension

```
def SEXT(s):
    sign = s[0]
    res = sign * (16 - len(s))   + s
    return res

def ZEXT(s):
    return '0' * (16 - len(s)) + s
```

I define two functions to do binary digit extension to sixteen bits.

One function which is named “SEXT” aims to do extension on signed numbers. So, for negative numbers, it adds ‘1’s to their left sides while it adds ‘0’s to their right sides for positive numbers. This function is used to deal with immediate numbers and PCoffset. It is similar to LC-3’s extension.

Another function is named “ZEXT”, which adds ‘0’s to numbers’ right sides whether it is positive or not. I used it in LEA and BR and JSR to get the address to be stored in R7. Because address is unsigned numbers. It‘s very different from ZEXT in LC-3 which is used in TRAP instruction.

4.Set CC

```
def setCC(s):
    global cc
    if s[0] == '1':
        cc = -1
    else:
        if int(s, 2) == 0:
            cc = 0
        else:
            cc = 1
```

ADD、AND、NOT、LD、LDI、LDR instructions will change condition code. So after these instructions are executed, they pass the result to setCC function and the global variable cc is renewed. BR's execution depends on it.

5.BR instruction

```
def BR():
    global pc
    if (cc == -1 and ir[4] == '1') or (cc == 0 and ir[5] == '1') or (cc == 1 and ir[6] == '1'):
        PCoffset9 = binToDec(ir[7:])
        pc += PCoffset9
```

It change the PC according to the condition code. And "global pc" is very important to all functions change the pc.

6.JSR instruction

```
def JSR():
    global pc
    temp = pc
    mode = int(ir[4], 2)
    if mode == 1:
        PCoffset11 = binToDec(ir[5:])
        pc = pc + PCoffset11
    else:
        baser = binToDec(ir[7:10])
        pc = binToDec(rgst[baser])
    rgst[7] = ZEXT(bin(temp)[2:])
```

It has two modes so the mode(ir[4]) should be examined.

7.AND instruction

```
def AND():
    dr = int(ir[4:7], 2)
    sr1 = int(ir[7:10], 2)
```

```
    mode = int(ir[10], 2)
    if mode == 0:
        sr2 = int(ir[13:], 2)
        rgst[dr] = ANDhelper(rgst[sr1], rgst[sr2])
    else:
        imm5 = SEXT(ir[11:])
        rgst[dr] = ANDhelper(rgst[sr1], imm5)
    setCC(rgst[dr])
```

Before execution, the mode should be examined. And extension should be done before AND.

PART3. Reflection

When I design my algorithm, It's nature for me to store all the data as binary digits. It turns out that it's very convenient to write code in this way. But the disadvantage is also very obvious: It takes too much time to run. SHAME ON ME. This was an important lesson for me —— I should consider more before write code.