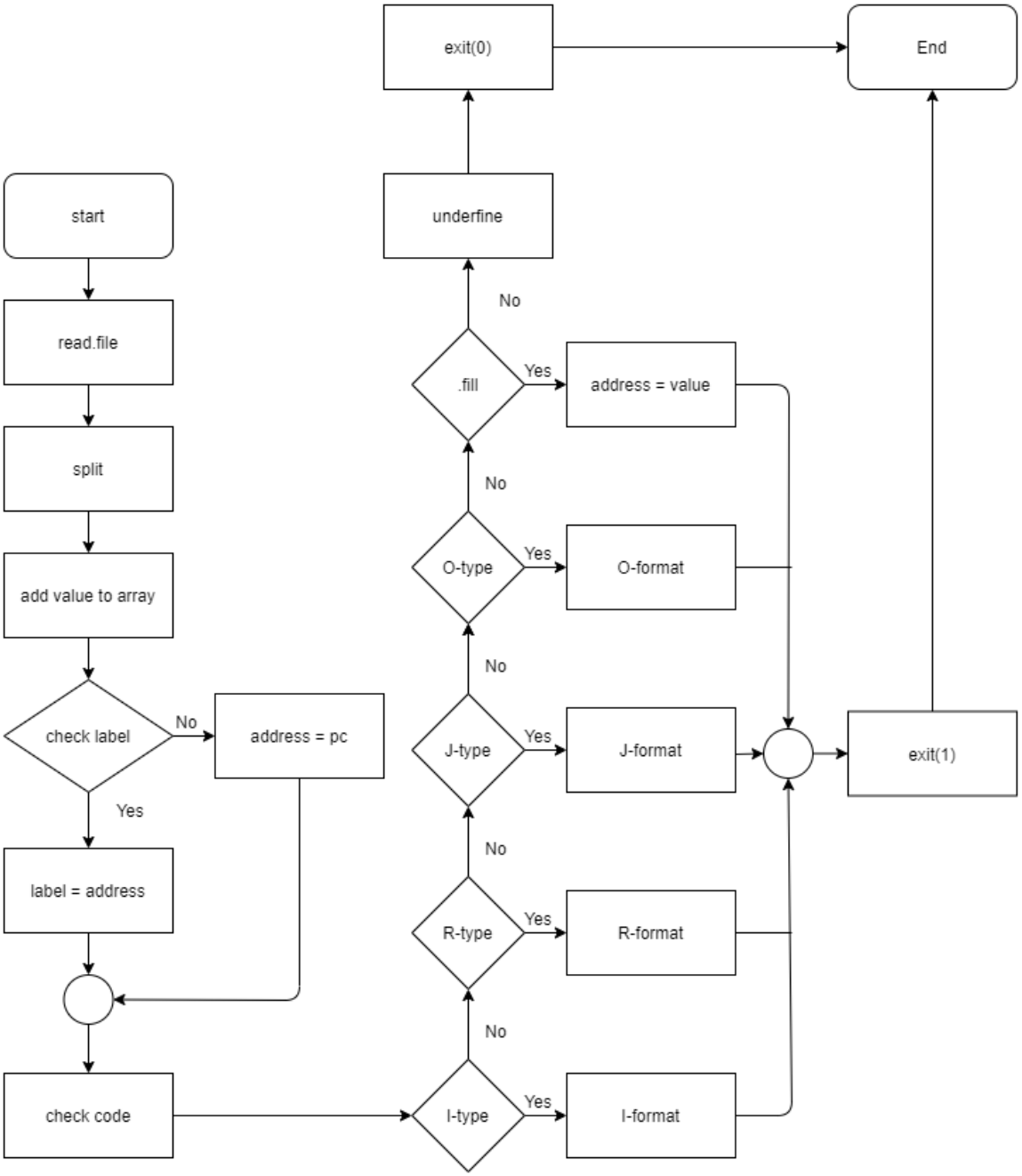
ส่วนที่ 1 Assembly to Machine Code Converter and Simulator

1.1 Assembly to Machine Code Converter



Flowchart การทำงานของโปรแกรมแปลง Assembly Code เป็น Machine Code

1.1.1 หลักการทำงาน (Pseudocode)

1.Read file อ่าน Assembly Code จากไฟล์ Text

2.ทำการ Split Row ส่วนต่างๆในหนึ่ง Row

3.หลังจาก Split เสร็จ นำไปเช็คว่ามี label หรือไม่

- หากมีให้ label = Address

- หากไม่มีให้ Address = PC

4.Check Code เพื่อทำการแยกคำสั่งว่าเป็น type ไหน

เป็น I-type ไหม

ถ้าเป็น

I-format

instructions (lw, sw, beq)

Bits 24-22 opcode

Bits 21-19 reg A (rs)

Bits 18-16 reg B (rt)

Bits 15-0 offsetField (เลข16-bit และเป็น 2’s complement โดยอยู่ในช่วง –32768 ถึง 32767)

จบการทำงาน

ถ้าไม่ ดูว่าเป็น R-type ไหม

ถ้าเป็น

R-format

instructions (add, nand)

Bits 24-22 opcode

Bits 21-19 reg A (rs)

Bits 18-16 res B (rt)

Bits 15-3 ไม่ใช้ (ควรตั้งไว้ที่ 0)

Bits 2-0 destReg (rd)

จบการทำงาน

ถ้าไม่ ดูว่าเป็น J-type ไหม

ถ้าเป็น

J-format

instructions (jalr)

Bits 24-22 opcode

Bits 21-19 reg A (rs)

Bits 18-16 reg B (rd)

Bits 15-0 ไม่ใช้ (ควรตั้งไว้ที่ 0)

จบการทำงาน

ถ้าไม่ ดูว่าเป็น O-type ไหม

ถ้าเป็น

O-format

instructions (halt, noop)

Bits 24-22 opcode

Bits 21-0 ไม่ใช้ (ควรตั้งไว้ที่ 0) หากเป็น I – Type ให้มาตรวจสอบ format โดยดูจาก Opcode ณ ตำแหน่งบิตที่ 24,23,22

จบการทำงาน

ถ้าไม่ ดูว่าเป็น .fill ไหม

ถ้าเป็น

เพิ่มค่าให้กับ Address

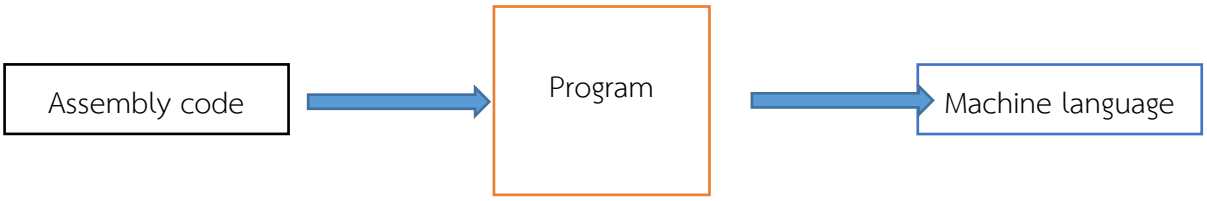
ถ้าไม่

แสดงว่า underfine

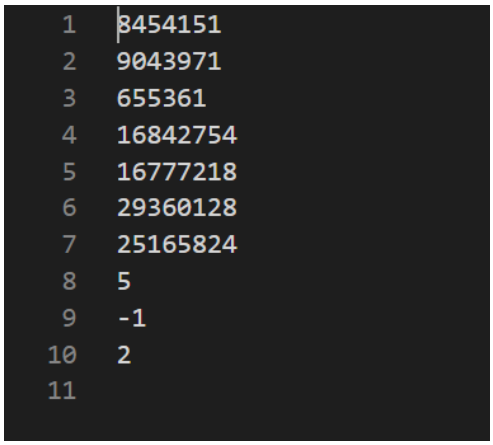
จบการทำงาน

1.1.2 ผลการทดลองและการวิเคราะห์ผลการทดลอง

โปรแกรมแปลง Assembly code เป็น Machine Language




```
        lw      0       1       five
        lw      1       2       3
start   add     1       2       1
        beq     0       1       2
        beq     0       0       start
        noop
done    halt
five    .fill   5
neg1    .fill   -1
stAddr  .fill   start
```

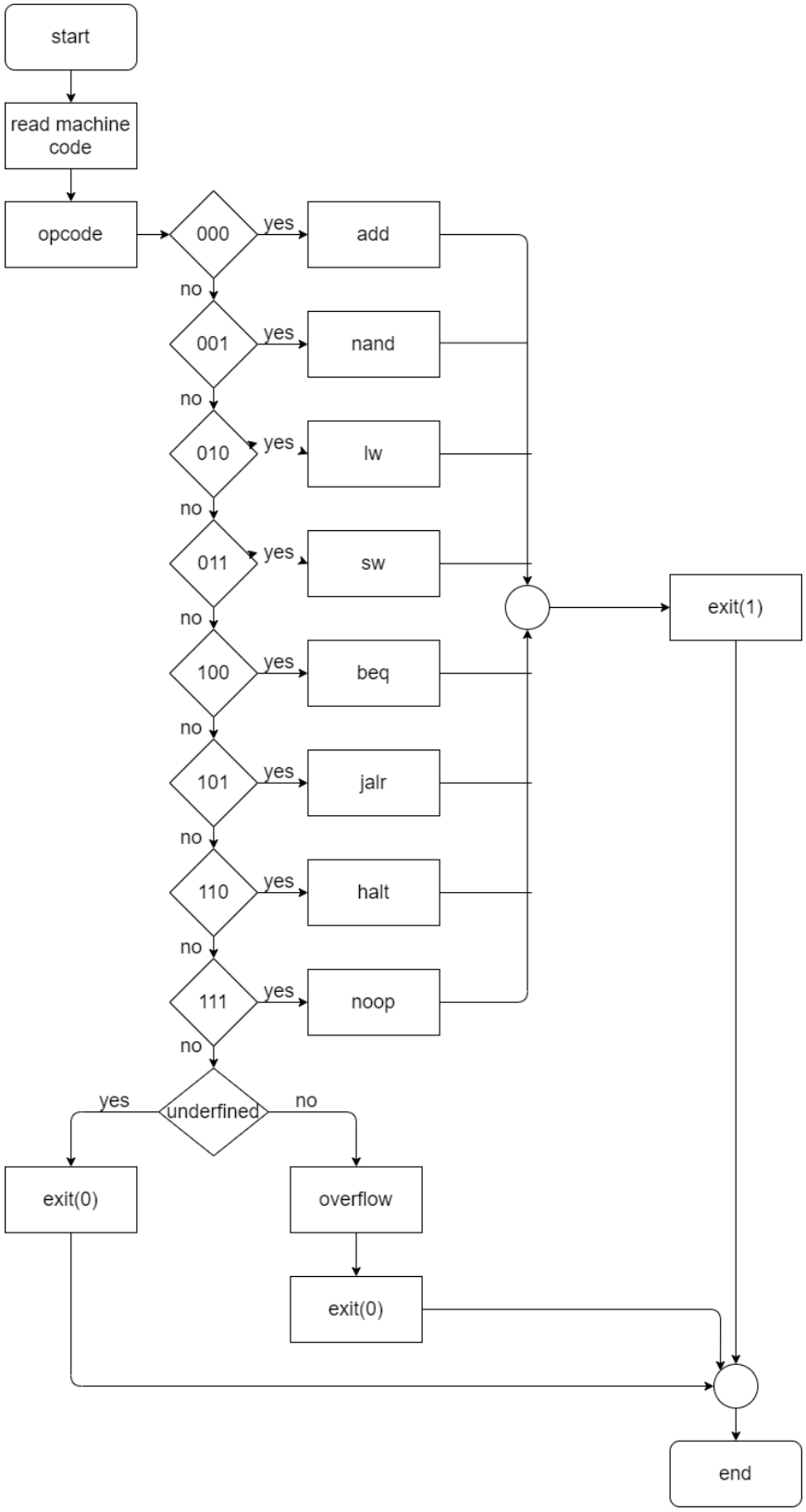


```
1   8454151
2   9043971
3   655361
4   16842754
5   16777218
6   29360128
7   25165824
8   5
9   -1
10  2
11
```

ภาพผลของการนำ Assembly code มาผ่านโปรแกรมแล้วจะได้ Machine code

จากภาพจะเห็นว่า Assembly code ในแต่ละบรรทัดก็จะเปลี่ยนได้เป็นเลขฐานสิบดังภาพ ซึ่งหากโปรแกรมได้รับ code ที่ถูก defined ไว้ก็จะสามารถแปลงเป็น machine language ได้อย่างถูกต้อง แต่จะเห็นว่า .fill จะเก็บเป็นค่าของ value นั้นๆ เลย ซึ่ง start ก็คือ address ของ row ที่เท่ากับ 2

1.2 Behavior Simulator



Flowchart การทำงานของโปรแกรม Simulator

1.2.1 หลักการทำงาน (Pseudocode) ของ Simulator

1.อ่าน Machine Code

2.ตรวจสอบ Op code

Op code คือ ‘000’ ใช่หรือไม่

ถ้าใช่

‘000’ คือ ฟังก์ชัน add

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘001’ ใช่หรือไม่

ถ้าใช่

‘001’ คือ ฟังก์ชัน nand

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘010’ ใช่หรือไม่

ถ้าใช่

‘010’ คือ ฟังก์ชัน lw

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘011’ ใช่หรือไม่

ถ้าใช่

‘011’ คือ ฟังก์ชัน sw

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘100’ ใช่หรือไม่

ถ้าใช่

‘100’ คือ ฟังก์ชัน beq

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘101’ ใช่หรือไม่

ถ้าใช่

‘101’ คือ ฟังก์ชัน jalr

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘110’ ใช่หรือไม่

ถ้าใช่

‘110’ คือ ฟังก์ชัน halt

จบการทำงาน

ถ้าไม่ ดูว่า Op code คือ ‘111’ ใช่หรือไม่

ถ้าใช่

‘111’ คือ ฟังก์ชัน noop

จบการทำงาน

ถ้าไม่ ดูว่า Op code นั้น Undefined หรือไม่

ถ้าใช่

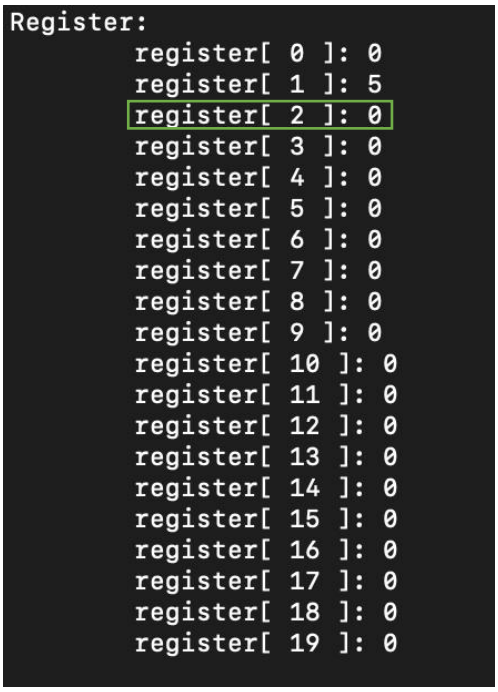
จบการทำงาน

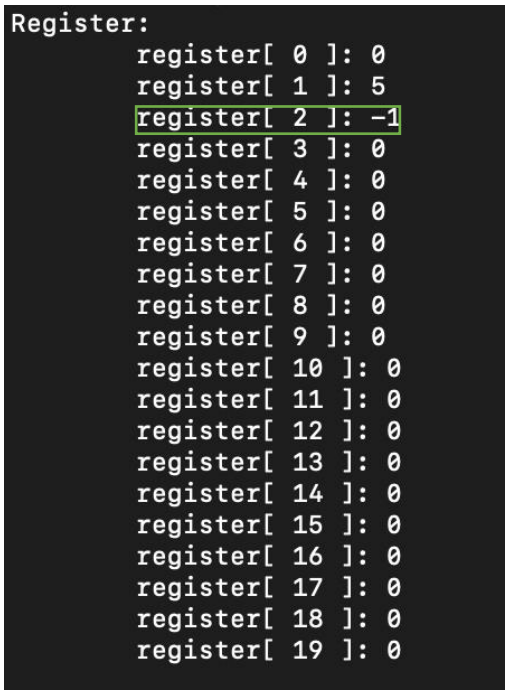
ถ้าไม่ แสดงว่า Overflow

จบการทำงาน

1.2.2 ผลการทดลองและการวิเคราะห์ผลการทดลอง

โปรแกรม Simulator ที่แสดงถึงการทำงานของ Register และ Memory



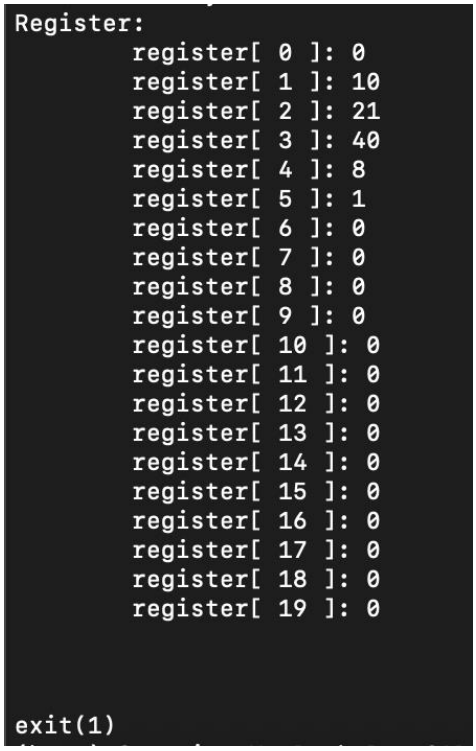


ภาพผลของการนำ Machine language มาผ่านโปรแกรม Simulator

จากภาพจะเห็นว่าใน Register[2] มีการเก็บค่า -1 เพิ่มขึ้นมา ซึ่งในการทำงานของ Register จริงๆ หากได้รับ Machine language มาก็จะทำการเปลี่ยนค่าใน Register แบบนี้เช่นกัน

1.3 Combination

1.3.1 ผลการทดลองและการวิเคราะห์ผลการทดลอง



ภาพผลของการทำงานของโปรแกรม Combination

จากภาพจะเห็นว่า Code จบด้วยการ exit(1) ซึ่งหมายความว่า โปรแกรมทำงานสำเร็จ หากท่านต้องการศึกษาเพิ่มเติมสามารถศึกษา code ได้เพิ่มเติมในลิงค์ในส่วนของ**ภาคผนวก**ได้

ส่วนที่ 3 ภาคผนวก

3.1 Assembly to Machine Code Converter

com_A.py

```
import numpy as np
import math

#with open('dat.txt' , 'r') as file:
#con = file.read().split('\n')
#del con[-1]
#
#for i in con:
#    count += 1
#    arr.append(i.split(','))

out = []
address = []
##---------------ดึงไฟล์ที่เป็น Assembly
with open('code.txt' , 'r') as file:
    con = file.read().split('\n')

    if not(con[-1]):
        del con[-1]

    for i in con:
        out.append(i.split('\t'))

```

```
##---------------เปลี่ยนเป็น 2's
def twosCom_decBin(dec, digit):
    if dec>=0:
            bin1 = bin(dec).split("0b")[1]
            while len(bin1)<digit :
                bin1 = '0'+bin1
            return bin1
    else:
            bin1 = -1*dec
            return bin(dec-pow(2,digit)).split("0b")[1]

label = []
code = []
first = []
second = []
third = []
line = len(out)
```

```
##---------------setค่าแต่บรรทัด
for i in range (len(out)):
    for j in range (len(out[i])):
        if(j == 0):
            label.append(out[i][j])
        elif(j == 1):
            code.append(out[i][j])
        elif(j == 2):
            first.append(out[i][j])
        elif(j == 3):
            second.append(out[i][j])
        else:
            third.append(out[i][j])

labelin = dict()
addr = dict()
value = dict()
```

```
##---------------เก็บ label
for i in range (line):
    if(label[i] != ''):
        labelin.update({label[i] : f'address[{i}]'})
        value.update({f'address[{i}]' : i})
    else:
        labelin.update({f'{int(i)}': f'address[{i}]'})
```

```
##---------------fill ค่าไปใน address
for i in range (line):
    if(code[i] == '.fill'):
        try:
            addr.update({f'address[{i}]': int(first[i])})
        except:
            index = 0
            for j in range (line):
                if(first[i] == label[j]):
                    index = int(j)

            addr.update({f'address[{i}]': index})
```

```
##---------------กำหนด machine code ของแต่ละคำสั่ง
for i in range (line):
    if(code[i] == 'lw'):
        if(not(third[i].isdecimal())):
#            print(addr.get(labelin.get(third[i])))
            op = 2 << 22
            addr.update({f'address[{i}]': op
            + (int(first[i]) << 19)
            + (int(second[i]) << 16)
            + (int(value.get(labelin.get(third[i]))))
            })
        else:
            op = 2 << 22
            addr.update({f'address[{i}]': op
            + (int(first[i]) << 19)
            + (int(second[i]) << 16)
            + int(third[i])
            })

    elif(code[i] == 'add'):
        if(third[i].isdecimal()):
            op = 0 << 22
            addr.update({f'address[{i}]': op
            + (int(first[i]) << 19)
            + (int(second[i]) << 16)
            + (int(third[i]))
            })
        else:
            op = 0 << 22
            addr.update({f'address[{i}]': op
            + (int(first[i]) << 19)
            + (int(second[i]) << 16)
            + (int(addr.get(labelin.get(third[i]))))
            })

    elif(code[i] == 'nand'):
        if(third[i].isdecimal()):
            op = 1 << 22
            addr.update({f'address[{i}]': op
            + (int(first[i]) << 19)
            + (int(second[i]) << 16)
            + (int(third[i]))
            })
        else:
```

3.2 Behavior Simulator

assembler.py

```python
out = []
address = []

##---------------2's converter
def twosCom_decBin(dec, digit):
    if dec>=0:
        bin1 = bin(dec).split("0b")[1]
        while len(bin1)<digit :
            bin1 = '0'+bin1
        return bin1
    else:
        bin1 = -1*dec
        return bin(dec-pow(2,digit)).split("0b")[1]

def twosCom_binDec(bin, digit):
    while len(bin)<digit :
        bin = '0'+bin
    if bin[0] == '0':
        return int(bin, 2)
    else:
        return -1 * (int(''.join('1' if x == '0' else '0' for x in bin), 2) + 1)

def binaryToDecimal(binary):

    binary1 = binary
    decimal, i, n = 0, 0, 0
    while(binary != 0):
        dec = binary % 10
        decimal = decimal + dec * pow(2, i)
        binary = binary//10
        i += 1
    return decimal


##---------------ดึงไฟล์ machine code
with open('machine_code.txt','r')as file:
    con = file.read().split('\n')

    if not(con[-1]):
        del con[-1]

    for i in con:
        out.append(int(i))

##---------------กำหนด address
for i in range (len(out)):
    address.append(out[i])

binary = []

##---------------ค่าที่ติดลบ ทำเป็น 2's
for i in range (len(address)):
    if(address[i] < 0):
#       print(twosCom_decBin(address[i] , 16))
        binary.append(twosCom_decBin(address[i] , 16))

    else:
        binary.append(bin(address[i])[2:])

##---------------เติม 0 ให้ครบ
for i in range (len(binary)):
    if(len(binary[i]) != 25):
        add = 25 - len(binary[i])
        binary[i] = '0'*add + binary[i]
print(binary)
```

```python

##---------------init reg กับ mem
reg = [0] * 20
memory = []

for i in range (len(address)):
    memory.append(address[i])

for i in range (25):
    memory.append(0)

##---------------check op code ว่าคือ function ไหน เริ่มการรันโค้ด
i = 0
while i < len(binary):
    print(f'pc[{i}]:')
    print('Memory:')
    ##---------------display memory reg
    for j in range (len(memory)):
        print(f'\tmemory[ {j} ]:', memory[j])

    print('Register:')

    for j in range (len(reg)):
        print(f'\tregister[ {j} ]:' , reg[j])

    print('\n\n')
```

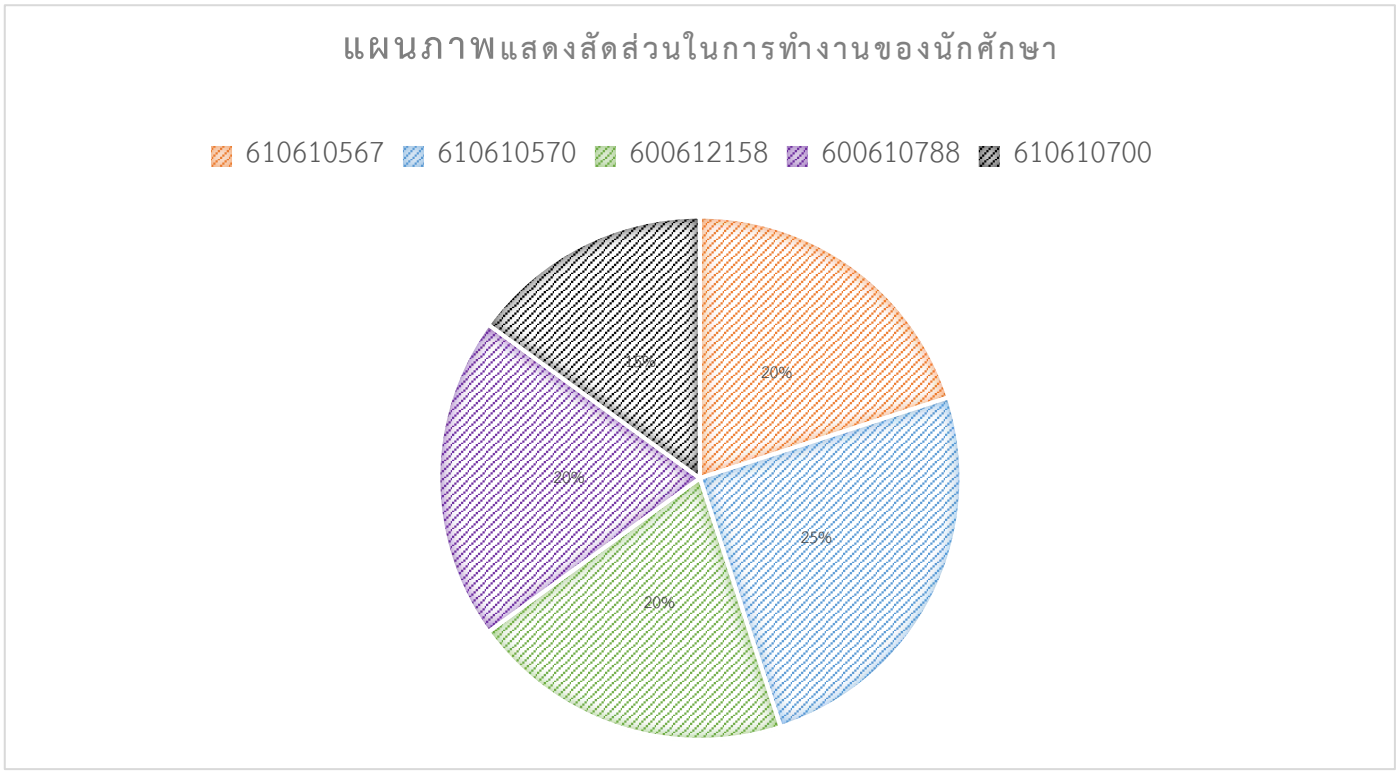
```
    ##---------------จำแนกชนิดของ โค้ด
    if(binary[i][:3] == '000'):
#         print('Radd')
        regA = binary[i][3:6]
        regB = binary[i][6:9]
        regdst = binary[i][22:]
        reg[binaryToDecimal(int(regdst))] = reg[binaryToDecimal(int(regA))] + reg[binaryToDecimal(int(regB))]
        i = i + 1
    elif(binary[i][:3] == '001'):
#         print('Rnand')
        regA = binary[i][3:6]
        regB = binary[i][6:9]
        regdst = binary[i][22:]
        reg[binaryToDecimal(int(regdst))] = not (reg[binaryToDecimal(int(regA))] and reg[binaryToDecimal(int(regB))])
        i = i + 1
    elif(binary[i][:3] == '010'):
#           print('lw')
        regA = binary[i][3:6]
        regB = binary[i][6:9]
        offset = binary[i][9:]
#           print(offset)
        reg[binaryToDecimal(int(regB))] = memory[reg[binaryToDecimal(int(regA))] + binaryToDecimal(int(offset))]
#           print('======', reg[binaryToDecimal(int(regA))]
#           + binaryToDecimal(int(offset)))
#           reg[binaryToDecimal] = reg[regA] + offset
#         print(regA , regB , offset)
        i = i + 1
    elif(binary[i][:3] == '011'):
#         print('sw')
        regA = binary[i][3:6]
        regB = binary[i][6:9]
        offset = binary[i][9:]
        print(binaryToDecimal(int(offset)) , binaryToDecimal(int(regA)) , binaryToDecimal(int(regB)))
        memory[binaryToDecimal(int(offset)) + reg[binaryToDecimal(int(regA))]] = reg[binaryToDecimal(int(regB))]
```

Source code

-Simulator: code

-Converter: code

ส่วนที่ 4 สัดส่วนในการทำงานของสมาชิกในกลุ่ม



ส่วนที่ 5 ตารางผลการดำเนินงาน

ตารางบันทึกการทำงานของนักศึกษา

ลำดับที่	กิจกรรม	สัดส่วน (%)	ระยะเวลาในการดำเนินงาน							หมายเหตุ
			15/10	20/10	25/10	30/10	4/11	9/11	14/11	
1	ออกแบบและทำ Pseudo code	10								
2	เขียนโปรแกรมที่รับ assembly language program และสร้าง machine language ตาม assembly program	10								
3	เขียน behavioral simulator สำหรับ machine code	10								
4	เขียน recursive function และการคูณของเลข 2 จำนวน ด้วย assembly language program	15								
5	วิเคราะห์ผลการทดลอง	10								
6	ทำการทดลองนอกเหนือจากการคูณ และ combination(n,r)	10								ไม่เสร็จ
7	เขียน behavioral simulator สำหรับ machine code ของการทดลองที่นอกเหนือจากการคูณ และ combination(n,r)	10								ไม่เสร็จ
8	วิเคราะห์ผลการทดลองที่นอกเหนือจากการคูณ และ combination(n,r)	10								
9	สรุปผลและทำรายงาน	15								