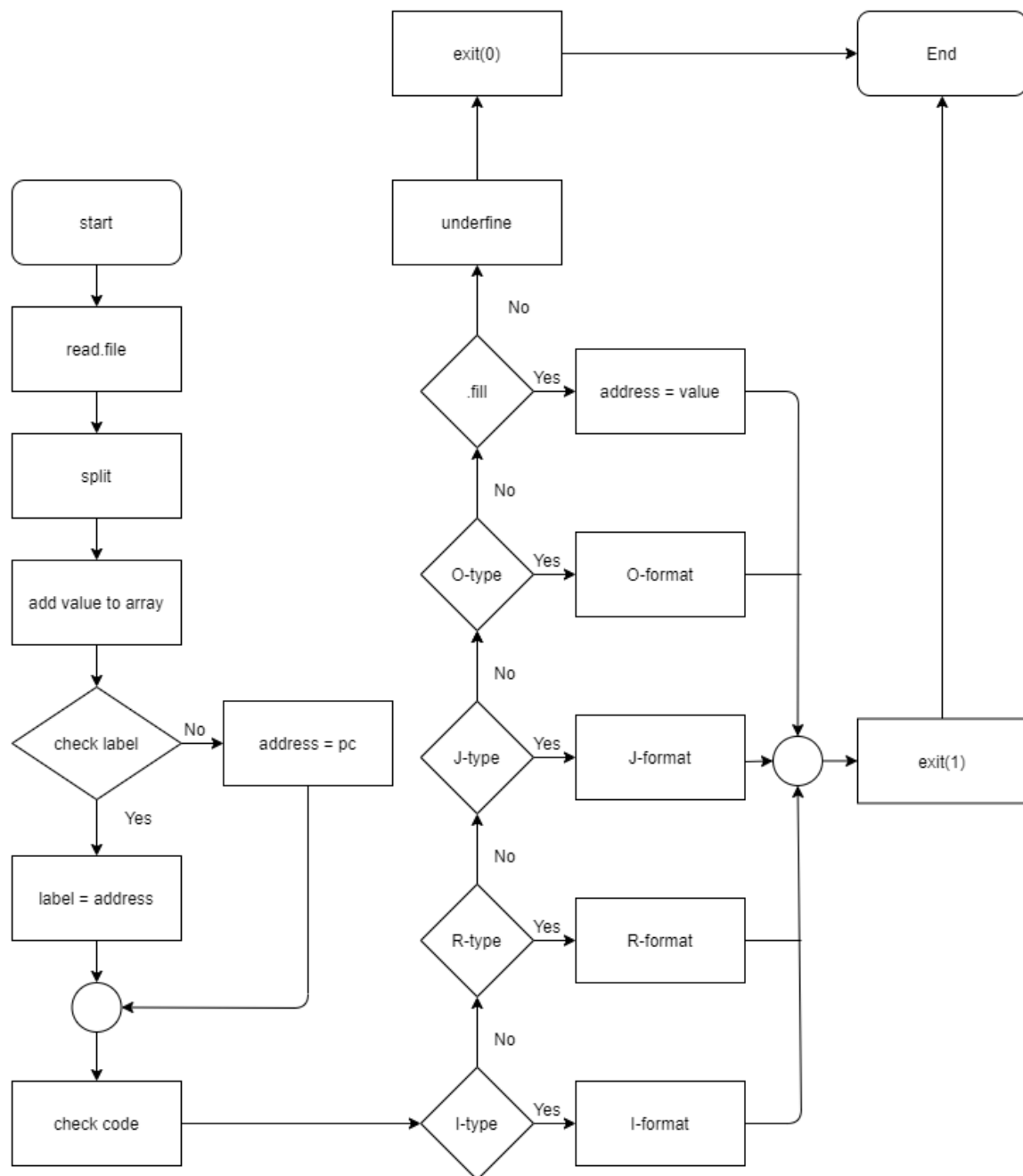


ส่วนที่ 1 Assembly to Machine Code Converter and Simulator

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 ไม่ใช่ (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 ไม่ใช่ (ควรตั้งไว้ที่ 0)

จบการทำงาน

ถ้าไม่ ดูว่าเป็น O-type ไหม

ถ้าเป็น

O-format

instructions (halt, noop)

Bits 24-22 opcode

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

จบการทำงาน

ถ้าไม่ ดูว่าเป็น .fill ไหม

ถ้าเป็น

เพิ่มค่าให้กับ Address

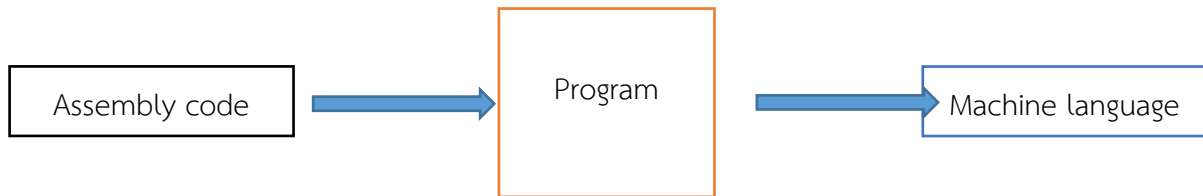
ถ้าไม่

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
```

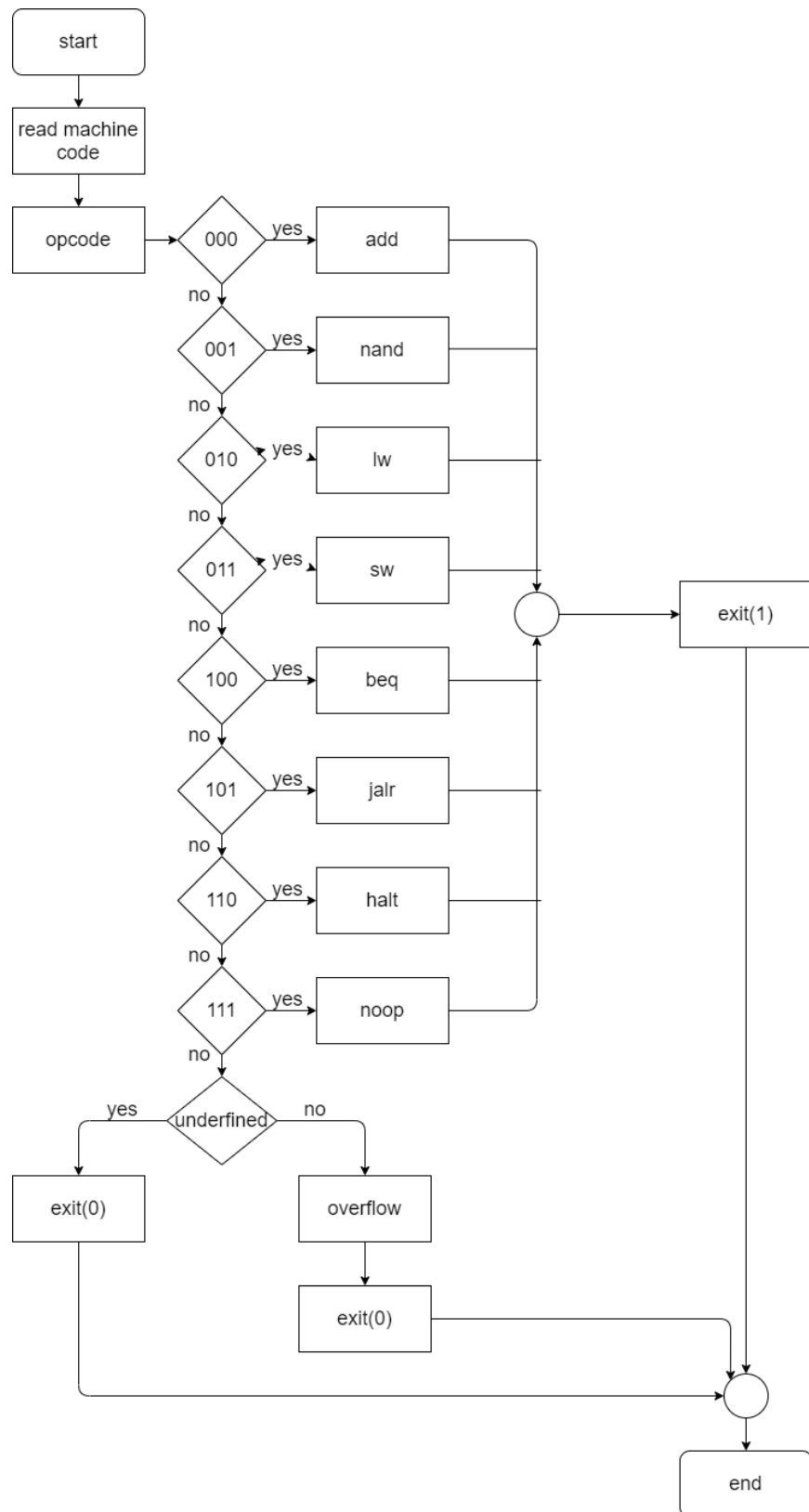


```
1 8454151
2 9043971
3 655361
4 16842754
5 16777218
6 29360128
7 25165824
8 5
9 -1
10 2
11
```

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

```
Register:
  register[ 0 ]: 0
  register[ 1 ]: 5
  register[ 2 ]: 0
  register[ 3 ]: 0
  register[ 4 ]: 0
  register[ 5 ]: 0
  register[ 6 ]: 0
  register[ 7 ]: 0
  register[ 8 ]: 0
  register[ 9 ]: 0
  register[ 10 ]: 0
  register[ 11 ]: 0
  register[ 12 ]: 0
  register[ 13 ]: 0
  register[ 14 ]: 0
  register[ 15 ]: 0
  register[ 16 ]: 0
  register[ 17 ]: 0
  register[ 18 ]: 0
  register[ 19 ]: 0
```

```
Register:
  register[ 0 ]: 0
  register[ 1 ]: 5
  register[ 2 ]: -1
  register[ 3 ]: 0
  register[ 4 ]: 0
  register[ 5 ]: 0
  register[ 6 ]: 0
  register[ 7 ]: 0
  register[ 8 ]: 0
  register[ 9 ]: 0
  register[ 10 ]: 0
  register[ 11 ]: 0
  register[ 12 ]: 0
  register[ 13 ]: 0
  register[ 14 ]: 0
  register[ 15 ]: 0
  register[ 16 ]: 0
  register[ 17 ]: 0
  register[ 18 ]: 0
  register[ 19 ]: 0
```

ภาพผลของการนำ Machine language มาผ่านโปรแกรม Simulator

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

1.3 Combination

1.3.1 ผลการทดลองและการวิเคราะห์ผลการทดลอง

```
Register:
    register[ 0 ]: 0
    register[ 1 ]: 10
    register[ 2 ]: 21
    register[ 3 ]: 40
    register[ 4 ]: 8
    register[ 5 ]: 1
    register[ 6 ]: 0
    register[ 7 ]: 0
    register[ 8 ]: 0
    register[ 9 ]: 0
    register[ 10 ]: 0
    register[ 11 ]: 0
    register[ 12 ]: 0
    register[ 13 ]: 0
    register[ 14 ]: 0
    register[ 15 ]: 0
    register[ 16 ]: 0
    register[ 17 ]: 0
    register[ 18 ]: 0
    register[ 19 ]: 0

exit(1)
```

ภาพผลการทำงานของโปรแกรม Combination

จากภาพจะเห็นว่า Code จบด้วยการ exit(1) ซึ่งหมายความว่า โปรแกรมทำงานสำเร็จ หากท่านต้องการศึกษาเพิ่มเติมสามารถศึกษา code ได้เพิ่มเติมในลิงค์ในส่วนของภาคผนวกได้

ส่วนที่ 3 ภาคผนวก

3.1 Assembly to Machine Code Converter

```
com_A.py
1  import numpy as np
2  import math
3
4  #with open('dat.txt' , 'r') as file:
5  #con = file.read().split('\n')
6  #del con[-1]
7  #
8  #for i in con:
9  #    count += 1
10 #    arr.append(i.split(','))
11
12 out = []
13 address = []
14 #####ส่งไฟล์ที่เป็น Assembly
15 with open('code.txt' , 'r') as file:
16     con = file.read().split('\n')
17
18     if not(con[-1]):
19         del con[-1]
20
21     for i in con:
22         out.append(i.split('\t'))
23
24
25
26 #####เปลี่ยนเป็น 2's
27 def twosCom_decBin(dec, digit):
28     if dec>=0:
29         bin1 = bin(dec).split("0b")[1]
30         while len(bin1)<digit :
31             bin1 = '0'+bin1
32         return bin1
33     else:
34         bin1 = -1*dec
35         return bin(dec-pow(2,digit)).split("0b")[1]
36
37 label = []
38 code = []
39 first = []
40 second = []
41 third = []
42 line = len(out)
43
44
45 #####setค่าแฉับบรรทัด
46 for i in range (len(out)):
47     for j in range (len(out[i])):
48         if(j == 0):
49             label.append(out[i][j])
50         elif(j == 1):
51             code.append(out[i][j])
52         elif(j == 2):
53             first.append(out[i][j])
54         elif(j == 3):
55             second.append(out[i][j])
56         else:
57             third.append(out[i][j])
58
59 labelin = dict()
60 addr = dict()
61 value = dict()
```

```

>>
60 ##-----เก็บ label
61 for i in range (line):
62     if(label[i] != ''):
63         labelin.update({label[i] : f'address[{i}]'})
64         value.update({f'address[{i}]' : i})
65     else:
66         labelin.update({f'{int(i)}': f'address[{i}]'})
67
68 ##-----fill ค่าไปใน address
69 for i in range (line):
70     if(code[i] == '.fill'):
71         try:
72             addr.update({f'address[{i}]': int(first[i])})
73         except:
74             index = 0
75             for j in range (line):
76                 if(first[i] == label[j]):
77                     index = int(j)
78             addr.update({f'address[{i}]': index})
79
80

```

```

80 ##-----กำหนด machine code ของแต่ละคำสั่ง
81
82 for i in range (line):
83     if(code[i] == 'lw'):
84         if(not(third[i].isdecimal())):
85             # print(addr.get(labelin.get(third[i])))
86             op = 2 << 22
87             addr.update({f'address[{i}]': op
88                 + (int(first[i]) << 19)
89                 + (int(second[i]) << 16)
90                 + (int(value.get(labelin.get(third[i]))))
91             })
92         else:
93             op = 2 << 22
94             addr.update({f'address[{i}]': op
95                 + (int(first[i]) << 19)
96                 + (int(second[i]) << 16)
97                 + int(third[i])
98             })
99
100     elif(code[i] == 'add'):
101         if(third[i].isdecimal()):
102             op = 0 << 22
103             addr.update({f'address[{i}]': op
104                 + (int(first[i]) << 19)
105                 + (int(second[i]) << 16)
106                 + (int(third[i]))
107             })
108         else:
109             op = 0 << 22
110             addr.update({f'address[{i}]': op
111                 + (int(first[i]) << 19)
112                 + (int(second[i]) << 16)
113                 + (int(addr.get(labelin.get(third[i]))))
114             })
115
116     elif(code[i] == 'nand'):
117         if(third[i].isdecimal()):
118             op = 1 << 22
119             addr.update({f'address[{i}]': op
120                 + (int(first[i]) << 19)
121                 + (int(second[i]) << 16)
122                 + (int(third[i]))
123             })
124         else:

```

3.2 Behavior Simulator

assembler.py

```
1 out = []
2 address = []
3
4 ##-----2's converter
5 def twoscom_decBin(dec, digit):
6     if dec>=0:
7         bin1 = bin(dec).split("0b")[1]
8         while len(bin1)<digit :
9             bin1 = '0'+bin1
10        return bin1
11    else:
12        bin1 = -1*dec
13        return bin(dec-pow(2,digit)).split("0b")[1]
14
15 def twoscom_bindec(bin, digit):
16     while len(bin)<digit :
17         bin = '0'+bin
18     if bin[0] == '0':
19         return int(bin, 2)
20     else:
21         return -1 * (int(''.join('1' if x == '0' else '0' for x in bin), 2) + 1)
22
23 def binaryToDecimal(binary):
24
25     binary1 = binary
26     decimal, i, n = 0, 0, 0
27     while(binary1 != 0):
28         dec = binary1 % 10
29         decimal = decimal + dec * pow(2, i)
30         binary1 = binary1//10
31         i += 1
32     return decimal
33
```

```
33
34 ##-----ดึงไฟล์ machine code
35 with open('machine_code.txt','r') as file:
36     con = file.read().split('\n')
37
38     if not(con[-1]):
39         del con[-1]
40
41     for i in con:
42         out.append(int(i))
43
44 ##-----คำนวณ address
45 for i in range (len(out)):
46     address.append(out[i])
47
48 binary = []
49
50 ##-----ค่าที่ติดลบ ทำเป็น 2's
51 for i in range (len(address)):
52     if(address[i] < 0):
53         print(twoscom_decBin(address[i] , 16))
54         binary.append(twoscom_decBin(address[i] , 16))
55     else:
56         binary.append(bin(address[i])[2:])
57
58 ##-----เติม 0 ให้อยู่ครบ
59 for i in range (len(binary)):
60     if(len(binary[i]) != 25):
61         add = 25 - len(binary[i])
62         binary[i] = '0'*add + binary[i]
63
64 print(binary)
65
```

```
65
66 ##-----init reg กับ mem
67 reg = [0] * 20
68 memory = []
69
70 for i in range (len(address)):
71     memory.append(address[i])
72
73 for i in range (25):
74     memory.append(0)
75
76 ##-----check op code ว่าคือ function โหนด เริ่มการรันโค้ด
77 i = 0
78 while i < len(binary):
79     print(f'pc[{i}]:')
80     print('Memory:')
81     ##-----display memory reg
82     for j in range (len(memory)):
83         print(f'\tmemory[ {j} ]:', memory[j])
84
85     print('Register:')
86
87     for j in range (len(reg)):
88         print(f'\tregister[ {j} ]:', reg[j])
89
90     print('\n\n')
```

```

91  ##-----จำแนกชนิดของ โคลด
92  if(binary[i][:3] == '000'):
93      #      print('Radd')
94      regA = binary[i][3:6]
95      regB = binary[i][6:9]
96      regdst = binary[i][22:]
97      reg[binaryToDecimal(int(regdst))] = reg[binaryToDecimal(int(regA))] + reg[binaryToDecimal(int(regB))]
98      i = i + 1
99  elif(binary[i][:3] == '001'):
100     #      print('Rnand')
101     regA = binary[i][3:6]
102     regB = binary[i][6:9]
103     regdst = binary[i][22:]
104     reg[binaryToDecimal(int(regdst))] = not (reg[binaryToDecimal(int(regA))] and reg[binaryToDecimal(int(regB))])
105     i = i + 1
106  elif(binary[i][:3] == '010'):
107     #      print('lw')
108     regA = binary[i][3:6]
109     regB = binary[i][6:9]
110     offset = binary[i][9:]
111     #      print(offset)
112     reg[binaryToDecimal(int(regB))] = memory[reg[binaryToDecimal(int(regA))] + binaryToDecimal(int(offset))]
113     #      print('=====', reg[binaryToDecimal(int(regA))]
114     #          + binaryToDecimal(int(offset)))
115     #      reg[binaryToDecimal] = reg[regA] + offset
116     #      print(regA , regB , offset)
117     i = i + 1
118  elif(binary[i][:3] == '011'):
119     #      print('sw')
120     regA = binary[i][3:6]
121     regB = binary[i][6:9]
122     offset = binary[i][9:]
123     print(binaryToDecimal(int(offset)) , binaryToDecimal(int(regA)) , binaryToDecimal(int(regB)))
124     memory[binaryToDecimal(int(offset)) + reg[binaryToDecimal(int(regA))]] = reg[binaryToDecimal(int(regB))]

```

Source code

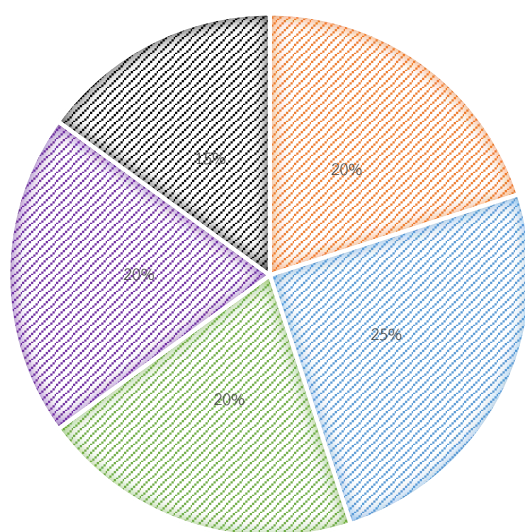
-Simulator: [code](#)

-Converter: [code](#)

ส่วนที่ 4 สัดส่วนในการทำงานของสมาชิกในกลุ่ม

แผนภาพแสดงสัดส่วนในการทำงานของนักศึกษา

610610567 610610570 600612158 600610788 610610700



ส่วนที่ 5 ตารางผลการดำเนินงาน

ตารางบันทึกการทำงานของนักศึกษา

| ลำดับ ที่ | 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 | ทำการทดลอง นอกเหนือจากการคูณ 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 | | | | | | | | |