ISA Simulator Program

Soravis Varayuththasawad (5913094)

Detail

This ISA simulation program is created by using Python Language

- 24-bit ISA
- Opcode take 5 bits
- Take 2 operands in each instruction
- 1 operand has 3 bits
- A value take 16 bits

Opcode 5 bits	Operand 3 bits	Binary 16 bits

8 Registers

```
self.registers = {"000" : [], "001" : [], "010" : [], "011" : [], "100" : []
, "101" : [], "110" : [], "111" : []}
```

```
regisBi = {"r0" : "000", "r1" : "001", "r2" : "010", "r3" : "011", "r4" : "100",
"r5" : "101", "r6" : "110", "r7" : "111"}
```

Access by using dictionary of Python 2 times

Opcode As functions (class' attribute)

```
def mov(self, firstReg, secondReg):
    binary = "00001"
    clock = 1
```

mov

- mov r1 r2 : Move value from register r2 to register r1
- mov r1 7 : Move value 7 to register r1

Add

```
def add(self, firstReg, secondReg):
    binary = "00010"
    clock = 3
```

- add r1 r2 : Sum value between r1 and r2, put into r1
- add r1 7 : Sum value between r1 and 7 , put into r1

Sub

```
def sub(self, firstReg, secondReg):
    binary = "00011"
    clock = 3
```

- add r1 r2 : Subtract value from r1 with r2, put into r1
- add r1 7 : Subtract value from r1 with 7 , put into r1

Mul

```
def mul(self, firstReg, secondReg):
    binary = "00100"
    clock = 4
```

- mul r1 r2 : multiply value from r1 with r2, put into r1
- mul r1 7 : multiply value from r1 with 7 , put into r1

*** mul will not double size of binary

Div

```
def div(self, firstReg, secondReg):
    binary = "00101"
    clock = 4
```

- add r1 r2 : divide value from r1 with r2, put into r1
- add r1 7 : divide value from r1 with 7 , put into r1

Input Restriction

```
mov r1 3
add r2 5
sub r2 1
mov r4 r2
div r4 r1
mul r1 r2
end 0 0
```

Type a opcode first following with 2 operands
3 of them have to be separate by a space

Cannot put a value at the first operand

Instruction

```
Input instruction
- Opcode : mov, add, sub, mul, and div
- Operand 1 : R0 - R7
- Operand 2: R0 - R7 or a value
- type 'end 0 0' for stop the input stage
Enter Inputs :
```

Step of each instruction

```
Steps   Operations      Instructions(24-bit):      Clock cycles
[0]     mov r1 , 3      00001 001 0000000000000011      1
[1]     add r2 , 5      00010 010 0000000000000101      3
[2]     sub r2 , 1      00011 010 0000000000000001      3
[3]     mov r4 , r2     00001 100 0000000000000100      1
[4]     div r4 , r1     00101 100 0000000000000011      4
[5]     mul r1 , r2     00100 001 0000000000000100      4
--------------------------------------------------------------------------
```

Result of register in each instruction

```
Steps of Register
r1      =       3       [0000000000000011]
r2      =       5       [0000000000000101]
r2      =       4       [0000000000000100]
r4      =       4       [0000000000000100]
r4      =       1       [0000000000000001]
r1      =       12      [0000000000001100]
```

Final Result with CPI

```
Final Register Result
R0      []      [0000000000000000]
R1      12      [0000000000001100]
R2      4       [0000000000000100]
R3      []      [0000000000000000]
R4      1       [0000000000000001]
R5      []      [0000000000000000]
R6      []      [0000000000000000]
R7      []      [0000000000000000]
```

```
CPI of the program : 2.6666666666666665
```

Sample Code of Arithmetic Operation

```
def add(self, firstReg, secondReg):
    binary = "00010"
    clock = 3
    self.updateCurrent(firstReg, secondReg, "add", binary, clock)
    firstReg = self.registerDecode(firstReg)
    secondReg = self.registerDecode(secondReg)

    done = self.operationCases(firstReg, secondReg)

    if done is False:
        try:
            self.registers[secondReg]
        except KeyError:
            self.registers[firstReg] = binaryOperation(self.registers[firstReg], secondReg)

            done = True

    if  done is False:
        self.registers[firstReg] = binaryOperation(self.registers[firstReg], self.registers[secondReg])

    self.stepOfInput()
```

Sample Code for Binary

```
def biDecode(self, value):
    if value >= 0:
        isValuePositive = True
    else:
        isValuePositive = False
        return (self.to_twoscomplement(16, value))

    baseCase = False
    result = ""
    while baseCase is False:
        if value == 0 or value == 1:
            baseCase = True
        else:
            remainder = value % 2
            value = math.floor(value / 2)
            result += str(remainder)
    result += str(value)
    result = result[::-1]
    if isValuePositive is True:
        return(result.zfill(16))
    else:
        return(result.rjust(16,'1'))
```

Implement Code

```
def inputLoop(self):
    self.askForInput()
    userInput = ""
    while(self.isFinish is False):
        operation = ""
        operand1 = ""
        operand2 = ""
        userInput = input("")
        if("end 0 0" in userInput or "close" in userInput):
            break
        else:
            operation, operand1, operand2 = userInput.split(" ")
            if self.doOperation(operation, operand1, operand2) is True:
                print("!!! Error Input !!!")
                break
    if("close" in userInput):
        None
    else:
        self.s.getInputResult()
        self.s.getRegResult()
        self.s.getFinalResult()
        self.s.getCpi()
        self.s = Simulator()
        self.inputLoop()
```

Sample output

```
mov r1 8
mov r2 6
sub r2 r1
mov r3 r2
mul r3 2
mul r4 r3
div r4 2
end 0 0

Steps   Operations      Instructions(24-bit):      Clock cycles
[0]     mov r1 , 8      00001 001 0000000000001000      1
[1]     mov r2 , 6      00001 010 0000000000000110      1
[2]     sub r2 , r1     00011 010 0000000000001000      3
[3]     mov r3 , r2     00001 011 1111111111111110      1
[4]     mul r3 , 2      00100 011 0000000000000010      4
[5]     mul r4 , r3     00100 100 1111111111111100      4
[6]     div r4 , 2      00101 100 0000000000000010      4
--------------------------------------------------------------------------
Steps of Register
r1      =       8       [0000000000001000]
r2      =       6       [0000000000000110]
r2      =       -2      [1111111111111110]
r3      =       -2      [1111111111111110]
r3      =       -4      [1111111111111100]
r4      =       -4      [1111111111111100]
r4      =       -2      [1111111111111110]
--------------------------------------------------------------------------
Final Register Result
R0      []      [0000000000000000]
R1      8       [0000000000001000]
R2      -2      [1111111111111110]
R3      -4      [1111111111111100]
R4      -2      [1111111111111110]
R5      []      [0000000000000000]
R6      []      [0000000000000000]
R7      []      [0000000000000000]
--------------------------------------------------------------------------
CPI of the program : 2.5714285714285716
--------------------------------------------------------------------------
```