ISA

Karan Sachdev 5935219

ISA simulation CPU

- 24-bit
- Written in Python

Opcode 5-bits	Operand-1 3-bits	Binary Number 16-bits

Registers

- 8 registers

```
register_operands = []
register_operands.append('000')
register_operands.append('001')
register_operands.append('010')
register_operands.append('011')
register_operands.append('100')
register_operands.append('101')
register_operands.append('110')
register_operands.append('111')
```

Binary of Opcode

```
mov = '00000'
add = '00010'
sub = '00011'
mul = '00100'
div = '00101'
```

Clock Cycle of Opcode

```
cpi_mov = 1
cpi_add = 2
cpi_sub = 2
cpi_mul = 4
cpi_div = 4
```

mov

- Operand 1 and Operand 2

  mov r1 r3 means move register 1 into register 3

- Operand 1 and Value

  mov r1 -6 means move the value -6 into register 1

add

- Operand 1 and Operand 2

  add r4 r2 means add the value of register 4 and register 2, then store the value in register 4.

- Operand 1 and Value

  add r2 1 means add the value of register 4 with 1 and store the value in register 4

sub

- Operand 1 and Operand 2

  sub r3 r2 means subtract value of register 3 with register 2 and store the value in register 3.

- Operand 1 and Value

  sub r3 5 means subtract the value of register 3 with 5 and store the value in register 3.

mul

- Operand 1 and Operand 2

  mul r6 r7 means multiply the value in register 6 and register 7 and store the 32-bit binary number of the multiplication in register 6

- Operand 1 and Value

  mul r6 8 means multiply the value in register 6 with 8 and store the 32-bit binary number of the multiplication

Div

- Operand 1 and Operand 2

  div r7 r3 means divide the value of register 7 and register 3 and store the divided value in register 7

- Operand 1 and Value

  div r2 4 means divide the value of register 2 with 4 and store the divided value in register 2

Sample Input

```
mov r0 8
mov r2 10
sub r2 r0
mov r3 r0
mul r3 2
mov r6 r3
div r6 2
mov r5 r6
end 0 0
```

Sample Output

```
Enter the Assembly Code
Opcode -> mov, add, sub, mul, div
Operands -> r1 to r7
PC         Decoded      Encoded Instruction(24-bit):  Clock Cycles
PC[0]->    mov r0 8      00000 000 0000000000001000     1
PC[1]->    mov r2 10      00000 010 0000000000001010     1
PC[2]->    sub r2 r0      00011 010 0000000000000010     2
PC[3]->    mov r3 r0      00000 011 0000000000001000     1
PC[4]->    mul r3 2      00100 011 0000000000010000     4
PC[5]->    mov r6 r3      00000 110 00000000000000000000000000010000     1
PC[6]->    div r6 2      00101 110 0000000000001000     4
PC[7]->    mov r5 r6      00000 101 0000000000001000     1
Register Steps
r0: 8[0000000000001000]
r2: 10[0000000000001010]
r2: 2[0000000000000010]
r3: r0[0000000000001000]
rm :r3 = [00000000000000000000000000010000]
r6: r3[00000000000000000000000000010000]
r6: 8.0[0000000000001000] re:r6[0000000000000000]
r5: r6[0000000000001000]
```

Final Register Result/CPI

```
r0 [0000000000001000]
r1 [0000000000000000]
r2 [0000000000000010]
r3 [00000000000000000000000000010000]
r4 [0000000000000000]
r5 [0000000000001000]
r6 [0000000000001000]
r7 [0000000000000000]
CPI of Program:  1.875
```

Sample Code

```python
def dec_to_32bin(n):
    digit = ''
    if int(n) < 0:
        return two_complement(n)
    while n != 0:
        temp = n % 2
        digit += str(temp)
        n = n // 2
    z = 32 - len(digit)
    for i in range(z):
        digit += '0'
    return digit[::-1]

def dec_to_bin(n):
    digit = ''
    if int(n) < 0:
        return two_complement(n)
    while n != 0:
        temp = n % 2
        digit += str(temp)
        n = n // 2
    z = 16 - len(digit)
    for i in range(z):
        digit += '0'
    return digit[::-1]
```

```python
def two_complement(n):
    n = dec_to_bin(abs(n))
    return complement(n)


def complement(n):
    carry_bit = 0
    temp = ''
    for i in range(len(n)):
        if n[i] == '0':
            temp += '1'
        else:
            temp += '0'
    temp_2 = ''
    temp = temp[::-1]
    for i in range(len(n)):
        if i == 0 and temp[i] == '1':
            temp_2 += '0'
            carry_bit = 1
        elif i == 0 and temp[i] == '0':
            temp_2 += '1'
            carry_bit = 0
        else:
            if carry_bit == 1 and temp[i] == '0':
                temp_2 += '1'
                carry_bit = 0
            elif carry_bit == 1 and temp[i] == '1':
                temp_2 += '0'
                carry_bit = 1
            else:
                temp_2 += temp[i]
    temp_2 = temp_2[::-1]
    return temp_2
```

```python
def register_checker(n):
    if n == 'r0':
        return True
    elif n == 'r1':
        return True
    elif n == 'r2':
        return True
    elif n == 'r3':
        return True
    elif n == 'r4':
        return True
    elif n == 'r5':
        return True
    elif n == 'r6':
        return True
    elif n == 'r7':
        return True
    else:
        return False


def register_operand2(r_names, r):
    temp = 0
    for i in range(len(r_names)):
        if r_names[i] == r:
            temp = i
    return temp
```

Sample Code

```
print('Enter the Assembly Code')
print('Opcode -> mov, add, sub, mul, div')
print('Operands -> r1 to r7')
user_input = None
opcode = []
operand1 = []
operand2 = []
registers = ['0000000000000000'] * 8
r_names = []
r_names.append('r0')
r_names.append('r1')
r_names.append('r2')
r_names.append('r3')
r_names.append('r4')
r_names.append('r5')
r_names.append('r6')
r_names.append('r7')
while user_input != 'end 0 0':
    user_input = input()
    op, opr1, opr2 = user_input.split(' ')
    opcode.append(op)
    operand1.append(opr1)
    operand2.append(opr2)
index = []
register_steps = []
for i in range(len(operand1)):
    for j in range(len(r_names)):
        if operand1[i] == r_names[j]:
            index.append(j)
```

```
CPI = 0
counter = 0
print('PC        Decoded      Encoded Instruction(24-bit):                  Clock Cycles ')
for i in range(len(opcode)-1):
    if opcode[i] == 'mov':
        CPI += cpi_mov
        checker = register_checker(operand2[i])
        if checker:
            temp = register_operand2(r_names, operand2[i])
            registers[index[i]] = registers[temp]
        else:
            if int(operand2[i]) < 0:
                registers[index[i]] = two_complement(int(operand2[i]))
            else:
                registers[index[i]] = dec_to_bin(int(operand2[i]))
        register_steps.append('r' + str(index[i]) + ': '+ str(operand2[i]) + '[' + str(registers[index[i]]) + ']')
        print('PC['+str(i)+']->   '+opcode[i], operand1[i], operand2[i]+'    '+mov, register_operands[index[i]], registers[index[i]]+'            ',  cpi_mov)
    elif opcode[i] == 'add':
        CPI += cpi_add
        checker = register_checker(operand2[i])
        if checker:
            temp = register_operand2(r_names, operand2[i])
            temp_2 = registers[index[i]]
            temp_2 = bin_to_dec(temp_2)
            temp_3 = registers[temp]
            temp_3 = bin_to_dec(temp_3)
            temp_2 = temp_2 + temp_3
            registers[index[i]] = dec_to_bin(temp_2)
        else:
            temp = bin_to_dec(registers[index[i]])
            temp = temp + int(operand2[i])
            registers[index[i]] = dec_to_bin(int(temp))
        register_steps.append('r' + str(index[i]) + ': '+ str(temp) + '[' + str(registers[index[i]]) + ']')
```

Sample Code

```
        checker = register_checker(operand2[i])
        if checker:
            temp = register_operand2(r_names, operand2[i])
            temp_2 = registers[index[i]]
            temp_2 = bin_to_dec(temp_2)
            temp_3 = registers[temp]
            temp_3 = bin_to_dec(temp_3)
            temp_2 = int(temp_2 / int(temp_3))
            remainder = int(temp_2) % int(temp_3)
            registers[index[i]] = dec_to_bin(temp_2)
            register_steps.append('r' + str(index[i]) + ': ' + str(temp_2) + '[' + str(registers[index[i]]) + ']' + ' re:' + 'r' + str(index[i]) + '[' + str(dec_to_bin(remainder))
        else:
            val = registers[index[i]]
            temp = bin_to_dec(val)
            temp = int(temp) / int(operand2[i])
            remainder = int(temp) % int(operand2[i])
            registers[index[i]] = dec_to_bin(int(temp))
            register_steps.append('r' + str(index[i]) + ': ' + str(temp) + '[' + str(registers[index[i]]) + ']' + ' re:' + 'r' + str(index[i]) + '[' + str(dec_to_bin(remainder))
        print('PC['+str(i)+']->   ', opcode[i], operand1[i], operand2[i],'    ',div, register_operands[index[i]], registers[index[i]],'            ', cpi_div)
    counter += 1
print('Register Steps')
for i in range(len(register_steps)):
    print(register_steps[i])
print('---')
print('      ')
print('      ')
for i in range(len(registers)):
    print('r' + str(i), '[' + str(registers[i]) + ']')
CPI = CPI/counter


print('CPI of Program: ', CPI)
```