# Computer Architecture

## (CS2202) Section 541

Parichart Meesin 6010017

# ISA SIMULATION CPU SIZE 16-BIT

❑ Implementation by using Kotlin Language.

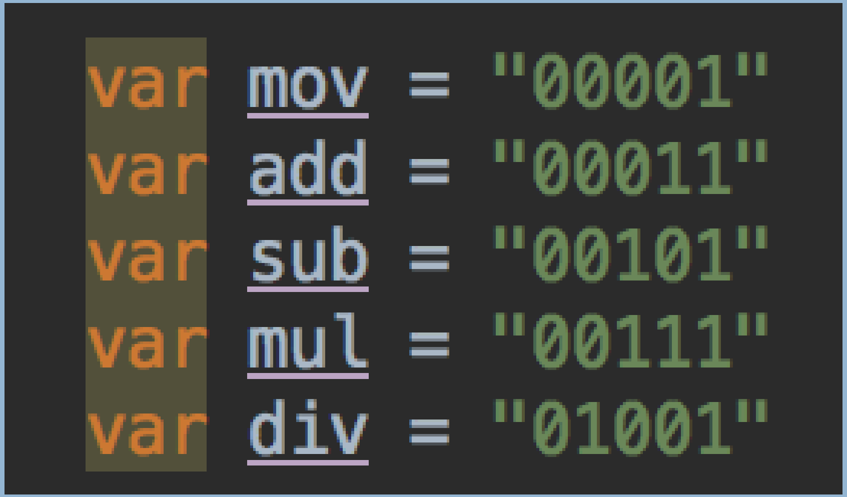
❑ 24-bits ISA

- 5-bits Opcode, with maximum 2 operands.
- 3-bits Operand.
- 16-bits binary format of number.

❑ 8 Registers ( r0 – r7 )

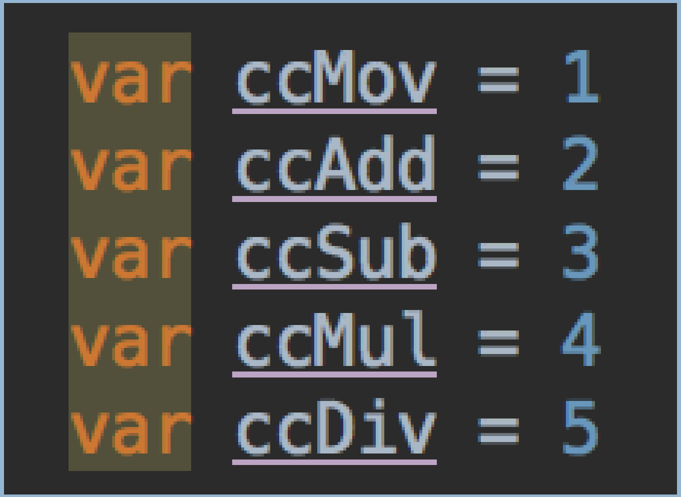
# OPCODE

❑ Binary of Opcode



```
var mov = "00001"
var add = "00011"
var sub = "00101"
var mul = "00111"
var div = "01001"
```

❑ Clock cycle for each Opcode



```
var ccMov = 1
var ccAdd = 2
var ccSub = 3
var ccMul = 4
var ccDiv = 5
```

# OPCODE FOR EACH INSTRUCTION

- Mov
  - Operand 1 and Operand 2
    - mov r1 r2 -> Value of r2 will be moved to r1
  - Operand 1 and Integer
    - mov r1 2 -> Value of 2 will be moved to r1

- Add
  - Operand 1 and Operand 2
    - add r1 r2 -> Value of r2 will be added to r1
  - Operand 1 and Integer
    - add r1 2 -> Value of 2 will be added to r1

# OPCODE FOR EACH INSTRUCTION ( CONT

- Sub
    - Operand 1 and Operand 2
        - sub r1 r2 -> Subtraction value of r1 by value of r2
    - Operand 1 and Integer
        - sub r1 2 -> Subtraction value of r1 by value of 2

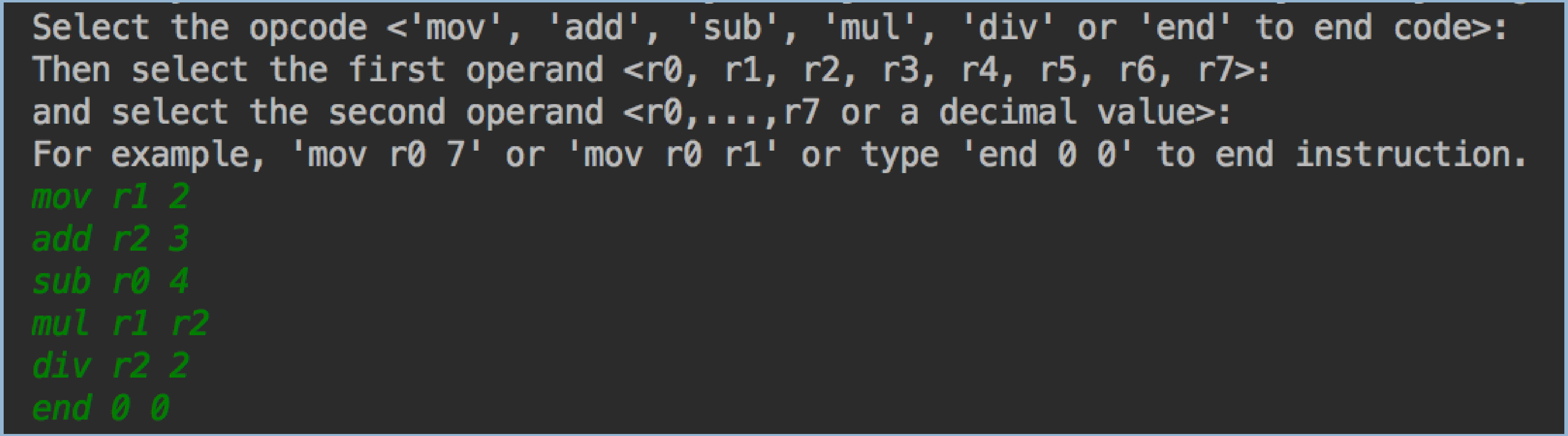
- Mul
    - Operand 1 and Operand 2
        - mul r1 r2 -> Multiple value of r1 with r2
    - Operand 1 and Integer
        - mul r1 2 -> Multiple value of r1 with 2

# OPCODE FOR EACH INSTRUCTION ( CONT

- Div
  - Operand 1 and Operand 2
    - div r1 r2 -> Value of r1 will be divided by value of r2
  - Operand 1 and Integer
    - div r1 2 -> Value of r1 will be divided by 2

# PROGRAM RUNNING

- ❑ Input the ”Opcode”
- ❑ Input the ”Operand 1”
- ❑ Input the ”Operand 2 or Decimal Value”
- ❑ Input the “end 0 0” to end the instruction

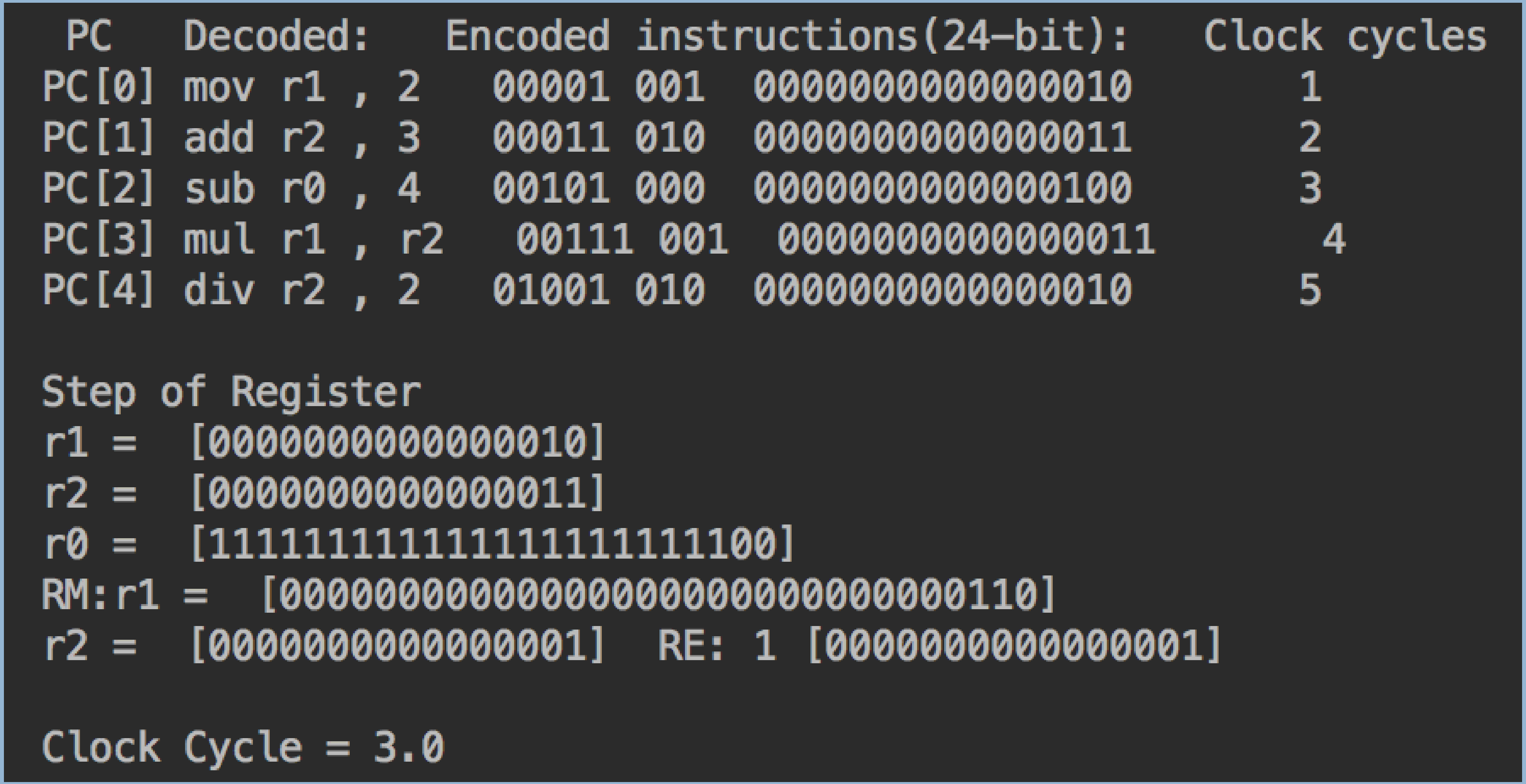


```
Select the opcode <'mov', 'add', 'sub', 'mul', 'div' or 'end' to end code>:
Then select the first operand <r0, r1, r2, r3, r4, r5, r6, r7>:
and select the second operand <r0,...,r7 or a decimal value>:
For example, 'mov r0 7' or 'mov r0 r1' or type 'end 0 0' to end instruction.
mov r1 2
add r2 3
sub r0 4
mul r1 r2
div r2 2
end 0 0
```

# RESULT

- ❑ The program will show the step of input
- ❑ The program will show the step of register
  - ❑ For multiply, the result will change to 32-bits binary, so the result will show in the form of “RM: Ri = [32-bits binary]”
  - ❑ For division, it is possible that the result can have the remainder, so the remainder will keep in “RE”, and show it in the form of “Ri = [16-bits binary] RE: [16-bits binary]”
- ❑ In the end of instruction, it will show CPI calculated
  - ❑ Finding CPI by using total clock cycles and number of PC
    - ❑ CPI = Total clock cycles / number of PC

# RESULT

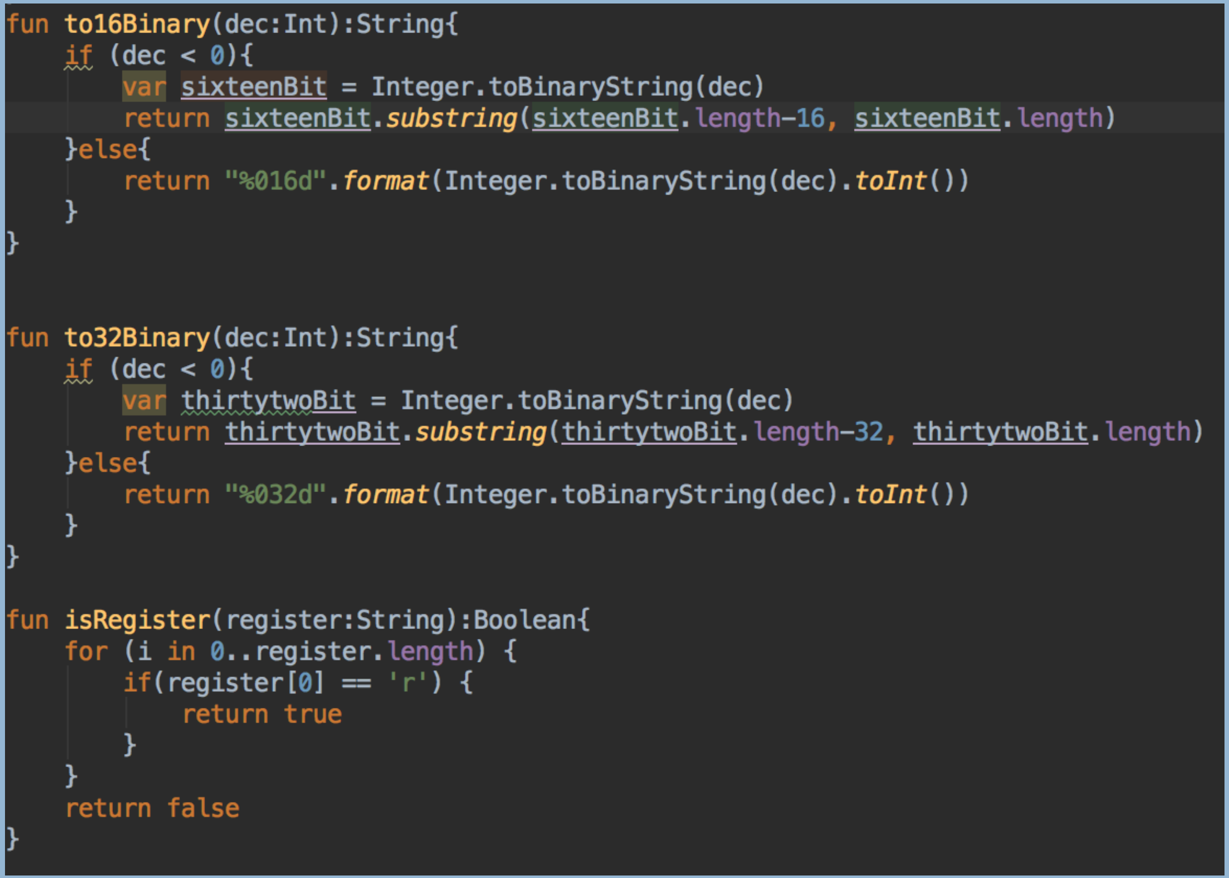


```
 PC   Decoded:   Encoded instructions(24-bit):   Clock cycles
PC[0] mov r1 , 2   00001 001  0000000000000010       1
PC[1] add r2 , 3   00011 010  0000000000000011       2
PC[2] sub r0 , 4   00101 000  0000000000000100       3
PC[3] mul r1 , r2   00111 001  0000000000000011       4
PC[4] div r2 , 2   01001 010  0000000000000010       5

Step of Register
r1 =  [0000000000000010]
r2 =  [0000000000000011]
r0 =  [111111111111111111111100]
RM:r1 =  [00000000000000000000000000000110]
r2 =  [0000000000000001]  RE: 1 [0000000000000001]

Clock Cycle = 3.0
```

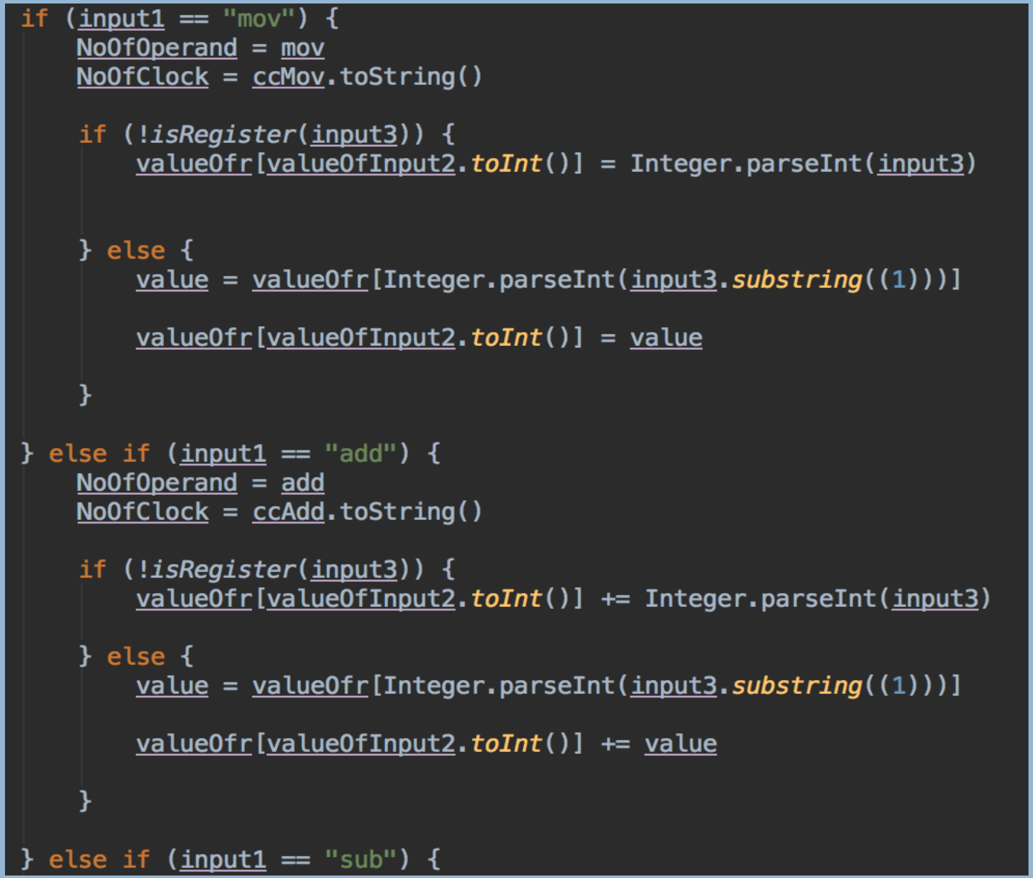
# CODING PART



```
fun to16Binary(dec:Int):String{
    if (dec < 0){
        var sixteenBit = Integer.toBinaryString(dec)
        return sixteenBit.substring(sixteenBit.length-16, sixteenBit.length)
    }else{
        return "%016d".format(Integer.toBinaryString(dec).toInt())
    }
}


fun to32Binary(dec:Int):String{
    if (dec < 0){
        var thirtytwoBit = Integer.toBinaryString(dec)
        return thirtytwoBit.substring(thirtytwoBit.length-32, thirtytwoBit.length)
    }else{
        return "%032d".format(Integer.toBinaryString(dec).toInt())
    }
}

fun isRegister(register:String):Boolean{
    for (i in 0..register.length) {
        if(register[0] == 'r') {
            return true
        }
    }
    return false
}
```



```
if (input1 == "mov") {
    NoOfOperand = mov
    NoOfClock = ccMov.toString()

    if (!isRegister(input3)) {
        valueOfr[valueOfInput2.toInt()] = Integer.parseInt(input3)


    } else {
        value = valueOfr[Integer.parseInt(input3.substring((1)))]

        valueOfr[valueOfInput2.toInt()] = value

    }

} else if (input1 == "add") {
    NoOfOperand = add
    NoOfClock = ccAdd.toString()

    if (!isRegister(input3)) {
        valueOfr[valueOfInput2.toInt()] += Integer.parseInt(input3)

    } else {
        value = valueOfr[Integer.parseInt(input3.substring((1)))]

        valueOfr[valueOfInput2.toInt()] += value

    }

} else if (input1 == "sub") {
```