COMPUTER ARCHITECTURE

ISA SIMULATOR PROGRAM

SUKIT W. 5913101

DESCRIBTION

LANGUAGE : JAVA

24 - bit ISA

Opcode : 5 bit
Operand 1 : 3 bit
BinaryNumber : 16 bit

```
Encoded instruction(24-bits):
00001 011 0000000000000011
```

Opcode
5 bit

Operand
3 bit

BinaryNumber
16 bit

DESCRIBTION

8 Register (R0 - R7)

with 2 array of value and address

```
int[] register = new int[8];
String[] regAddress = new String[8];
AddressSetup(regAddress);
```

Each of them has an address in 3 bit binary 000 - 111

```
private static void adressSetup(String[] regAdress) {
    regAdress[0] = "000";
    regAdress[1] = "001";
    regAdress[2] = "010";
    regAdress[3] = "011";
    regAdress[4] = "100";
    regAdress[5] = "101";
    regAdress[6] = "110";
    regAdress[7] = "111";
}
```

INPUT

My program read opcode, operand 1 and 2 separated by space consecutively

The program can contain only 20 instruction per one run

```
command = scan.next();

if (!command.equals("end")) {
    String op1 = scan.next();
    String op2 = scan.next();
```

Example : `mov r3 3`

assigned to String command

assigned to String op1

assigned to String op2

OPCODE : MOV

Assign value from operand 2 into operand 1 (register)

Update part

```
if (command.equals("mov")) {
    updateDecode(decode, decodeStr, command, op1, op2, commandCounter);
    updateEncode(encode, encodeStr, "00001", regAddress[regNum1], bin, 16, commandCounter);
    clockCycle[commandCounter - 1] = 1;
    // do move
    if (isOp2int) {
        register[regNum1] = op2Value;
    } else {
        register[regNum1] = register[regNum2];
    }
```

Check if operand 2 is integer

true : assign value 2 to register 1

false : assign value in register 2 to register 1

OPCODE : ADD

Add value from operand 2 into operand 1 (register)

Update part

```
} else if (command.equals("add")) {
    updateDecode(decode, decodeStr, command, op1, op2, commandCounter);
    updateEncode(encode, encodeStr, "00010", regAddress[regNum1], bin, 16, commandCounter);
    clockCycle[commandCounter - 1] = 2;
    // do add
    if (isOp2int) {
        register[regNum1] += op2Value;
    } else {
        register[regNum1] += register[regNum2];
    }
    history[commandCounter - 1] =  register[regNum1];
```

Check if operand 2 is integer

true : add value 2 to register 1

false : add value in register 2 to register 1

OPCODE : SUB

Subtract value from operand 2 into operand 1 (register)

Update part

```
} else if (command.equals("sub")) {
    updateDecode(decode, decodeStr, command, op1, op2, commandCounter);
    updateEncode(encode, encodeStr, "00011", regAddress[regNum1], bin, 16, commandCounter);
    clockCycle[commandCounter - 1] = 2;
    // do sub
    if (isOp2int) {
        register[regNum1] -= op2Value;
    } else {
        register[regNum1] -= register[regNum2];
    }
    history[commandCounter - 1] =  register[regNum1];
```

Check if operand 2 is integer

true : Subtract value 2 to register 1

false : Subtract value in register 2 to register 1

OPCODE : MUL

Multiply value from operand 2 into operand 1 (register)

Update part

```
} else if (command.equals("mul")) {
    updateDecode(decode, decodeStr, command, op1, op2, commandCounter);
    updateEncode(encode, encodeStr, "00100", regAddress[regNum1], bin, 16, commandCounter);
    clockCycle[commandCounter - 1] = 4;
    isMul[commandCounter - 1] = true;
    // do multiply
    if (isOp2int) {
        register[regNum1] *= op2Value;
    } else {
        register[regNum1] *= register[regNum2];
    }
    history[commandCounter - 1] =  register[regNum1];
```

Check if operand 2 is integer

true : Multiply value 2 to register 1

false : Multiply value in register 2 to register 1

OPCODE : DIV

Divide value from operand 2 into operand 1 (register)

Update part

```
} else if (command.equals("div")) {
    updateDecode(decode, decodeStr, command, op1, op2, commandCounter);
    updateEncode(encode, encodeStr, "00101", regAddress[regNum1], bin, 16, commandCounter);
    clockCycle[commandCounter - 1] = 4;
    isDiv[commandCounter - 1] = true;

    // do division
    if (isOp2int) {
        register[regNum1] /= op2Value;
        divRE[divCounter] = register[regNum1] % op2Value;
    } else {
        register[regNum1] /= register[regNum2];
        divRE[divCounter] = register[regNum1] % register[regNum2];
    }
    divCounter++;
    history[commandCounter - 1] =  register[regNum1];
```

Check if operand 2 is integer

true : Divide value 2 to register 1
false : Divide value in register 2 to register 1

END PROGRAM

When you enter “end” into the program,
it will show PC history, step of register, result and average cc.

```
PC      Decoded          Encoded instruction(24-bits):    Clock cycles
PC[0]   mov r3 3         00001 011 0000000000000011             1
PC[1]   mov r4 4         00001 100 0000000000000100             1
PC[2]   mov r0 10        00001 000 0000000000001010             1
PC[3]   mov r4 3         00001 100 0000000000000011             1
PC[4]   add r4 3         00010 100 0000000000000011             2
PC[5]   add r0 2         00010 000 0000000000000010             2
PC[6]   sub r0 8         00011 000 0000000000001000             2
PC[7]   mul r0 r3        00100 000 0000000000000011             4
PC[8]   div r3 2         00101 011 0000000000000010             4
```

```
Step of Register
R3 = 3 [0000000000000011]
R4 = 4 [0000000000000100]
R0 = 10 [0000000000001010]
R4 = 3 [0000000000000011]
R4 = 6 [0000000000000110]
R0 = 12 [0000000000001100]
R0 = 4 [0000000000000100]
RM : R0 = 12 [00000000000000000000000000001100]
R3 = 1 [0000000000000001] RE :1 [0000000000000001]
```

```
Final result
R0 = 0 [0000000000000011]
R1 = 0 [0000000000000100]
R2 = 0 [0000000000000000]
R3 = 3 [0000000000000000]
R4 = 4 [0000000000000000]
R5 = 0 [0000000000000000]
R6 = 0 [0000000000000000]
R7 = 0 [0000000000000000]
```

```
Averege Clock Cycle = 2.0
```

RESULT EXAMPLE :

```
PC      Decoded          Encoded instruction(24-bits):    Clock cycles
PC[0]   mov r1 8         00001 001 0000000000001000             1
PC[1]   mov r2 6         00001 010 0000000000000110             1
PC[2]   sub r2 r1        00011 010 0000000000001000             2
PC[3]   mov r3 r2        00001 011 1111111111111110             1
PC[4]   mul r3 2         00100 011 0000000000000010             4
PC[5]   mov r4 r3        00001 100 1111111111111100             1
PC[6]   div r4 2         00101 100 0000000000000010             4

Step of Register
R1 = 8 [0000000000001000]
R2 = 6 [0000000000000110]
R2 = -2 [1111111111111110]
R3 = -2 [1111111111111110]
RM : R3 = -4 [11111111111111111111111111111100]
R4 = -4 [1111111111111100]
R4 = -2 [1111111111111110] RE :0 [0000000000000000]


Final result
R0 = 0 [0000000000001000]
R1 = 8 [0000000000000110]
R2 = -2 [1111111111111110]
R3 = -4 [1111111111111110]
R4 = -2 [1111111111111100]
R5 = 0 [1111111111111100]
R6 = 0 [1111111111111110]
R7 = 0 [0000000000000000]

Averege Clock Cycle = 2.0
```

CODE PART EXAMPLE :

```
StringBuilder decodeStr = new StringBuilder();
StringBuilder encodeStr = new StringBuilder();

command = scan.next();

if (!command.equals("end")) {
    String op1 = scan.next();
    String op2 = scan.next();

    commandCounter++;

    int regNum1 = Integer.parseInt(op1.substring(1));
    registerHistory[commandCounter - 1] = regNum1;
    int regNum2 = 0;

    // in case of Integer OP3
    boolean isOp2int = (op2.charAt(0) >= '0' && op2.charAt(0) <= '9');
    int op2Value = 0;
    String bin;

    if (isOp2int) {
        op2Value = Integer.parseInt(op2);
        bin = Integer.toBinaryString(op2Value);
    } else {
        regNum2 = Integer.parseInt(op2.substring(1));
        bin = Integer.toBinaryString(register[regNum2]);
    }

    if (command.equals("mov")) {
        updateDecode(decode, decodeStr, command, op1, op2, commandCounter);
        updateEncode(encode, encodeStr, "00001", regAddress[regNum1], bin, 16, commandCounter);
        clockCycle[commandCounter - 1] = 1;
        // do move
        if (isOp2int) {
            register[regNum1] = op2Value;
        } else {
            register[regNum1] = register[regNum2];
        }
```

METHODS EXAMPLE :

```
private static void updateEncode(String[] encode, StringBuilder encodeStr, String commandCode, String regAddress,
        String bin, int bit, int counter) {
    encodeStr.append(commandCode);
    encodeStr.append(" ");
    encodeStr.append(regAddress);
    encodeStr.append(" ");
    encodeStr.append(fillLenght(bin, bit));

    encode[counter - 1] = encodeStr.toString();
}
```

```
private static void updateDecode(String[] decode, StringBuilder decodeStr, String command, String op1, String op2,
        int counter) {
    decodeStr.append(command);
    decodeStr.append(" ");
    decodeStr.append(op1);
    decodeStr.append(" ");
    decodeStr.append(op2);

    decode[counter - 1] = decodeStr.toString();

}
```

```
private static String fillLenght(String bin, int lenght) {
    StringBuilder result = new StringBuilder(bin);
    int i = result.length();

    if (i < lenght) {
        for (i = result.length(); i < lenght; i++) {
            result.insert(0, "0");
        }
    } else if (i > lenght) {
        result.delete(0, lenght - 1);
    }

    return result.toString();
```