COMPUTER ARCHITECTURE

CPU ISA SIMULATION

Done by : Harpreet Singh Arora
5838428

WHAT IS CPU ISA?

The specifications that determine how to implement machine-language of the system that will help the compiler/OS designer to interact with the machine- Control Unit design.

DESCRIPTION OF ISA SIMULATION

I decided to choose a 24-bits CPU Instruction set architecture for my project. Also, the program is made of 3-bits registers consist of 8 GPRs which are r0 – r7 of 16-bits each and 2 additional registers which are “rm” for another 16-bit multiplication value and “re” for remainder of division. I chose Operations Code (Op Code) to be 5 bits. The ISA I programmed is capable of operations such as mov, add, sub, mul, div ,and end. The program is also capable of calculating Average CPI.

Operations

	Register	Immediate
Mov	“00000”	“00001”
Add	“00010”	“00011”
Sub	“00100”	“00101”
Mul	“00110”	“00111”
Div	“01000”	“01001”
End	0	0

Registers

r0 → “000”
r1 → “001”
r2 → “010”
r3 → “011”
r4 → “100”
r5 → “101”
r6 → “110”
r7 → “111”
rm → Storing multiplication values
re → Storing remainder of the division

Opcode 5 bits	Operand 3 bits	Binary number 16 bits

THE OP-CODE WORKS AS FOLLOWS:

Mov Operation

Operand I and Operand II

Mov r1 r4 → which moves the register r4 into register r1

Operand I and Value

Mov r2 25 → which moves the value 25 into register r1

Add Operation

Operand I and Operand II

Add r1 r4 → which add the register r4 into register r1

Operand I and Value

Add r2 25 → which add the value 25 into register r1

Sub Operation

Operand I and Operand II

Sub r1 r4 → which subtracts the register r4 into register r1

Operand I and Value

Sub r2 25 → which subtracts the value 25 into register r1

Mul Operation

Operand I and Operand II

Mul r1 r4 → which multiply the register r4 into register r1

Operand I and Value

Mul r2 25 → which multiply the value 25 into register r1

Div Operation

Operand I and Operand II

Div r1 r4 → which divide the register r4 into register r1

Operand I and Value

Div r2 25 → which divide the value 25 into register r1

PROGRAM CODE!

```java
String opcode = "";
String register = "";
String value = "";
String str = "";
ISA_Startup();

while (!opcode.equals("end")) {
    str = in.nextLine();
    list.add(str);
    StringTokenizer tokenizer = new StringTokenizer(str, " ");
    opcode = tokenizer.nextToken();
}

int size = list.size();

for (int i = 0; i < size - 1; i++) {
    str = list.get(i);
    StringTokenizer tokenizer = new StringTokenizer(str, " ");
    opcode = tokenizer.nextToken();
    register = tokenizer.nextToken();
    value = tokenizer.nextToken();

    String execute = execute(opcode, register, value);
    str = "PC[" + i + "] -> " + opcode + " " + register + ", " + value + ": " + execute;
    execution.add(str);

    if (opcode.equals("mul")) {
        finalResult.add("rm:" + register + " = " + returnReg(register).getValue() +
                " [" + rm.getBinary() + returnReg(register).getBinary() + "]");
    }

    else if (opcode.equals("div")) {
        finalResult.add(register + " = " + returnReg(register).getValue()
                + " [" + returnReg(register).getBinary() + "] " + "re: "
                + re.getValue() + " [" + re.getBinary() + " ]");
    }

    else {
        finalResult.add(register + " = " + returnReg(register).getValue()
                + " [" + returnReg(register).getBinary() + " ]");
    }
}
```

```java
public static String execute(String opcode, String register, String value) {
    String str = "";
    String bin = "";
    String tmp = "";

    if (opcode.equals("mov")) {

        if (value.startsWith("r")) {
            bin = inputCheck(value);
            tmp = inputCheck(value);
            opcode = "00000";
        } else {
            bin = value;
            tmp = value;
            opcode = "00001";
        }

        CPI = 1;
        move(register, bin);
    }

    else if (opcode.equals("add")) {

        if (value.startsWith("r")) {
            bin = inputCheck(value);
            tmp = inputCheck(value);
            opcode = "00010";
        } else {
            bin = value;
            tmp = value;
            opcode = "00011";
        }

        CPI = 2;
        add(register, bin);
    }

    else if (opcode.equals("sub")) {

        if (value.startsWith("r")) {
            bin = inputCheck(value);
            tmp = inputCheck(value);
            opcode = "00100";
        } else {
            bin = value;
            tmp = value;
            opcode = "00101";
        }

        CPI = 3;
        sub(register, bin);
    }
```

PROGRAM CODE!

```java
public class Register {
    private String operand;
    private String binary;
    private int value;


    public Register (String Operand) {
        operand = Operand;
        binary = Integer.toBinaryString(value);
        value = 0;
    }

    public Register() {
        value = 0;
        binary = Integer.toBinaryString(value);
        binary();
    }

    public void binary() {
        if (value >= 0) {
            int x = binary.length();
            String str = binary;
            for (;str.length() < 16;) {
                str += "0";
            }
            binary = str.substring(x) + binary;

        } else {
            binary = binary.substring(16);
        }
    }

```

```java
    public String getOperand() {
        return operand;
    }

    public String getBinary() {
        return binary;
    }

    public void setBinary(String Binary) {
        binary = Binary;
    }

    public int getValue() {
        return value;
    }

    public void setValue(int Value) {
        value = Value;
        binary = Integer.toBinaryString(Value);
        binary();
    }
}
```

PROGRAM RUN!

```
Details of the ISA:
=====================
It is a 24-bit ISA and can handle any 16-bit or 8-bit Arithmetic operations.
There are eight 16-bit GPRs (r0-r7) excluding two 16-bit additional registers;
'rm' and 're', are meant for storing 16-bit upper result after a
multiplication and 16-bit remainder after a division respectively.
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 23' or 'mov r0 r1' or type 'end 0 0' end instruction.
mov r0 10
mov r1 12
add r0 r1
mov r2 r0
sub r2 5
mul r2 2
mov r3 r2
add r3 2
div r3 6
mov r4 r3
end 0 0
```

The program will start with the instructions on how you could execute commands that are offered by this ISA Simulation program. First, you start off with "Opcode" then with "Register" and lastly with "Register or Integer Value". This whole code statement input will be written in one line.

PROGRAM RUN!

Then the program will show the result of the execution that you desired to command.

```
 PC   Decoded: Encoded instructions(24-Bit):  Clock cycles
PC[0] -> mov r0, 10: 00001 000  0000000000001010       1
PC[1] -> mov r1, 12: 00001 001  0000000000001100       1
PC[2] -> add r0, r1: 00010 000  0010000000000000       2
PC[3] -> mov r2, r0: 00000 010  0000000000000000       1
PC[4] -> sub r2, 5: 00101 010  0000000000000101       3
PC[5] -> mul r2, 2: 00111 010  0000000000000010       4
PC[6] -> mov r3, r2: 00000 011  0100000000000000       1
PC[7] -> add r3, 2: 00011 011  0000000000000010       2
PC[8] -> div r3, 6: 01001 011  0000000000000110       5
PC[9] -> mov r4, r3: 00000 100  0110000000000000       1
```

PROGRAM RUN!

```
 After the program execution contents of the registers are...........
r0 = 10 [0000000000001010 ]
r1 = 12 [0000000000001100 ]
r0 = 22 [0000000000010110 ]
r2 = 22 [0000000000010110 ]
r2 = 17 [0000000000010001 ]
rm:r2 = 34 [00000000000000000000000000100010]
r3 = 34 [0000000000100010 ]
r3 = 36 [0000000000100100 ]
r3 = 6 [0000000000000110] re: 0 [0000000000000000 ]
r4 = 6 [0000000000000110 ]
```

For opcode “mul” it will display the result in form such as “rm:r_ (“_” will display the targeted register) which will turn the result into 32-bit form rather than usual 16-bit.

“rm : r_ = ? [32-bit binary result]”

For opcode “div” it will show the result of the division and it can hold the remainder which will be stored in the register “re”.

“r_ = ? [16-bit binary result] re: ? [16-bit binary remainder]”

Then the program will show the result that has been in executed step wise.

PROGRAM RUN!

Lastly, the program will show the average CPI that is found out using Clock cycle according to the input command the user desires.

```
CPI of program............

CPI = 2.1
```

THANK YOU! ☺