ISA Simulation:

Navin Singh 5935221

ISA Simulation CPU

-24-bits

-I’ve created ISA Simulation program by using Java language. It is a 24-bit ISA. This ISA has “Opcode” 5 bits and provide “Operation code” in 16 formats which maximum 2 operands which each operand has 3 bits. The last is the number in format of binary which this project supports 16 binary numbers.

Registers

- Register:
- r0 =“000”
- r1 = “001”
- r2 =“010”
- r3 = “011”
- r4 = “100”
- r5 = “101”
- r6 = “110”
- r7 = “111”

Binary of Opcode

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

Clock Cycles of opcode

ClockCycles:

- Mov = 1
- Add = 2
- Sub = 3
- Mul = 4
- Div = 5

Mov and Add

- 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
- Operand I and Operand II
- Add r1 r4 ➡ which adds the register r4 into register r1
- Operand I and Value
- Add r2 25 ➡ which adds the value 25 into register r1

Sub Div Mul

- Operand I and Operand II
- Sub r1 r4 ⟹ which subtract the register r4 into register r1
- Operand I and Value
- Sub r2 25 ⟹ which subtract the value 25 into register r1
- 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
- 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

```
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 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, or r7>
 and select the second operand <r0,.....,r7 or a decimal value >:
For example, 'mov r0 5' or 'mov r0 r1' or type 'end 0 0' end instruction.

mov r0 6
mov r1 -1
add r0 6
mov r2 r1
add r3 r0
mul r3 4
div r4 r3
div r5 6
end 0 0
```

The program will start with suggestion. The user must input the input from in program. The input from is Opcode spacebar Register spacebar
Register or Integer value, that is one line of one input order. You can input how much you want until you press end 0 0 to stop

Step of input order after execution

```
 PC    Decoded:   Encoded instructions(24-bit):   Clock cycles
PC[0]-> mov r0, 6: 00001 000  000000000000110        1
PC[1]-> mov r1, -1: 00001 001  1111111111111111       1
PC[2]-> add r0, 6: 00011 000  000000000001100        2
PC[3]-> mov r2, r1: 00000 010  001000000000000       1
PC[4]-> add r3, r0: 00010 011  000000000000000       2
PC[5]-> mul r3, 4: 00111 011  0000000000110000       4
PC[6]-> div r4, r3: 01000 100  0110000000000000      5
PC[7]-> div r5, 6: 01001 101  000000000000000        5
```

Steps of registers:

```
 After the program execution contents of the regiters are.....
r0 = 6 [0000000000000110]
r1 = -1 [1111111111111111]
r0 = 12 [0000000000001100]
r2 = -1 [1111111111111111]
r3 = 12 [0000000000001100]
rm:r3 = 48 [00000000000000000000000110000]
r4 = 0 [0000000000000000] re: 0 [0000000000000000]
r5 = 0 [0000000000000000] re: 0 [0000000000000000]
```

This is the step of register:
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]"

CPI of the program:

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

CPI = 2.625
```

Sample Code:

```java
public static void div(String reg, String val)
{
    int a = registerValue(reg).getValue() / Integer.parseInt(val);
    re.setValue(registerValue(reg).getValue() % Integer.parseInt(val));
    registerValue(reg).setValue(a);
}

public static void move(String reg, String val)
{
    registerValue(reg).setValue(Integer.parseInt(val));
}

public static void add(String reg, String val)
{
    registerValue(reg).setValue(registerValue(reg).getValue() + Integer.parseInt(val));
}

public static void sub(String reg, String val)
{
    registerValue(reg).setValue(registerValue(reg).getValue() - Integer.parseInt(val));
}

public static void mul(String reg, String val)
{
    long a = registerValue(reg).getValue() * Integer.parseInt(val);
    registerValue(reg).setValueWithout16bit(a);
}
```