ISA simulation program
IN JAVA
CREATED BY Grid Kornsutatipkul 6013732
What is ISA?

• An instruction set architecture or ISA is distinguished from a microarchitecture, which is the set of processor design techniques used, in a particular processor, to implement the instruction set.
Classification of ISAs

• **Complex instruction set computer** (CISC)
  • has many specialized instructions, some of which may only be rarely used in practical programs.

• **Reduced instruction set computer** (RICS)
  • simplifies the processor by efficiently implementing only the instructions that are frequently used in programs, while the less common operations are implemented as subroutines, having their resulting additional processor execution time offset by infrequent use.

• **Others**: very long instruction word (VLIW), explicitly parallel instruction computing (EPIC), minimal instruction set computer (MISC), one instruction set computer (OISC)
Instructions types

• **Data handling and memory operations**
  • Load or Store data (mov)

• **Arithmetic and logic operations**
  • Add, subtract, multiply, or divide

• **Others**: Control flow operations, Coprocessor instructions, Complex instructions
Instruction parts

- MIPS 24-bit instructions
  - Opcode
  - Operand
  - Operand2 or Value

<table>
<thead>
<tr>
<th>Opcode (5 bits)</th>
<th>Operand (3 bits)</th>
<th>Value or Operand2 (16 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mov (00001)</td>
<td>R1 (001)</td>
<td>10 (0000000000001010)</td>
</tr>
</tbody>
</table>

Instruction: Mov R1 10
ISA Simulation (JAVA)

```
"C:\Program Files\Java\jdk1.8.0_172\bin\java" ...

Input instruction
- Opcode : mov, add, sub, mul, and div
- Operand 1 : R0 - R7
- Operand 2: R0 - R7 or a value
- Type 'end 0 0' for stop the input stage

Enter Inputs :

mov r1, r0
add r0, 20
mul r1, 20
sub r0, 19
mov r3, 2

Steps | Decoded | Encoded Instructions(24-bits) | Clock cycles
--- | --- | --- | ---
[0] | mov r1, 10 | 00000100100000000000000000010 | 1
[1] | add r0, 20 | 00001000000000000000000001000 | 2
[2] | mul r1, 20 | 00001000000000000000000010000 | 4
[3] | sub r0, 19 | 00001100000000000000000001100 | 3
[4] | mov r3, 2 | 00000100100000000000000000010 | 1

Steps of Register
r1 = 10 [0000000000010100]
r0 = 20 [0000000000010100]
r1 = 20 [0000000000101000]
r0 = 19 [0000000000010111]
r1 = 2 [0000000000000000]

Final Register Result:
r0 = 19 [0000000000010111]
r1 = 20 [0000000001010000]
r2 = 0 [0000000000000000]
r3 = 2 [0000000000010010]
r4 = 0 [0000000000010000]
r5 = 0 [0000000000000000]
r6 = 0 [0000000000000000]
```

| r5 | [0000000000000000] |
| r6 | [0000000000000000] |
| r7 | [0000000000000000] |

CFI of the program : 2.0

Pipelined Execution of the Program

It is assumed that the CPU has 4-pipeline stages: IF (Instruction Fetch), ID (Instruction decoding), EX (Execution) and WB (Writes back). And each stage completes within one clock cycle and RAW hazard (any) will be solved with FORMATTING (from the o/p register of ALU to the i/p of next instruction’s ALU stage) without causing any stall.

```
1 2 3 4 5 6 7 8
1. mov r1 10 : IF | ID | EX | WB
2. add r0 20 : IF | ID | EX | WB
3. mul r1 20 : IF | ID | EX | WB
4. sub r0 19 : IF | ID | EX | WB
5. mov r3 2 : IF | ID | EX | WB

Pipelined execution took 8 clock cycles for the program execution.
```

Process finished with exit code 0
ISA Simulation (JAVA)

- Code
  - Main class (collecting input part)
ISA Simulation (JAVA)

- Code
  - Main class (printing output)
ISA Simulation (JAVA)

- Code
  - Register class

```java
public class Register {
    private String registerAddress;
    private int value;

    public Register(int value, String registerAddress) {
        this.registerAddress = registerAddress;
        this.value = value;
    }

    public int getValue() { return value; }
    public void setValue(int value) { this.value = value; }
    public String getValue16Bit() { 
        if (value > 0) {
            return String.format("%04x", Integer.toHexString(Integer.toString(value)));
        } else {
            String sixteenBitVal = Integer.toString(value);
            return sixteenBitVal.substring(sixteenBitVal.length() - 2, sixteenBitVal.length());
        }
    }

    public String getRegisterAddress() { return registerAddress; }
    public void setRegisterAddress(String registerAddress) { this.registerAddress = registerAddress; }

    public String getRegisterAddress16Bit() {
        switch (this.registerAddress) {
            case "$r0":
                return "0000";
            case "$r1":
                return "0001";
            case "$r2":
                return "0010";
            case "$r3":
                return "0011";
            case "$r4":
                return "0100";
            case "$r5":
                return "0101";
            case "r6":
                return "0110";
            case "r7":
                return "0111";
            case "r8":
                return "1000";
            case "r9":
                return "1001";
            case "r10":
                return "1010";
            case "r11":
                return "1011";
            case "r12":
                return "1100";
            case "r13":
                return "1101";
            case "r14":
                return "1110";
            case "r15":
                return "1111";
            default:
                return "";
        }
    }
}
```
ISA Simulation (JAVA)

- Code
  - Instructions class

```java
public class Instruction {
    private int step;
    private String opcode;
    private Register register;
    private String operand1;
    private int value;
    private int clockCycle;

    public Instruction(int step, String opcode, Register register, int clockCycle) {
        this.step = step;
        this.opcode = opcode;
        this.register = register;
        this.operand1 = operand1;
        this.value = register.getValue();
        this.clockCycle = clockCycle;
    }

    public Instruction(int step, String opcode, Register register, String operand2, int clockCycle) {
        this.step = step;
        this.opcode = opcode;
        this.register = register;
        this.operand2 = operand2;
        this.value = register.getValue();
        this.clockCycle = clockCycle;
    }

    public String toString() {
        switch (this.opcode) {
            case "add":
                return "00001001";
            case "mul":
                return "00100011";
            case "div":
                return "0000000010011000";
            default:
                return "error";
        }
    }
}
```