**Project - 1**

**TK Gate**

**Submitted By:-**

Amit Gupta *(Group Leader) 19ucs011*

Ram Ahuja *19ucs017*

Avinav Jain *19ucs021*

Premansh Sharma *19ucs084*

**ISA Design (Part 1):-**

**Instruction Length**: 16 bits

**Addressing Mode:** 3 address mode

**opcode**: 4 bits (16 instructions)

**oprands:** 3 bits / operand = 3 x 3 = 9 bits

*3 extra bits*

**Word Length:** 8 bits

**Memory:** 256 bytes

**Registers:-**

8 general purpose registers and 1 PC register

**Registers Binary Code:**-

R0 - 000

R1 - 001

R2 - 010

R3 - 011

R4 - 100

R5 - 101

R6 - 110

R7 - 111

**Instruction Details:-**

**Arithmetic:**

1. **Add:**

Binary code - 0000

Performs addition of values stored in two registers and stores the output in the third register.

Ex - Add R1 R2 R3 => Add values in R1 and R2 and store result in R3.

(0000-001-010-011-000)

1. **Subtract:**

Binary code - 0001

Performs subtraction of values stored in two registers and stores the output in the third register using 2’s complement.

Ex - Sub R1 R2 R3 => Subtract values in R1 from R2 and store result in R3.

(0001-001-010-011-000)

1. **Multiply:**

Binary code - 0010

Performs multiplication of values stored in two registers and stores the output in the third register.

Ex - Mul R1 R2 R3 => Multiply values in R1 and R2 and store result in R3.

(0010-001-010-011-000)

1. **Division:**

Binary code - 0011

Performs division of values stored in two registers and stores the output in the third register integrally.

Ex - Div R1 R2 R3 => Divide values in R1 from R2 and store result in R3.

(0011-001-010-011-000)

**Logical:**

1. **AND:**

Binary code - 0100

Performs logical AND of values stored in two registers and stores the output in the third register.

Ex - And R1 R2 R3 => Performs logical AND of values stored in R1 and R2 and store result in R3.

(0100-001-010-011-000)

1. **OR:**

Binary code - 0101

Performs logical OR of values stored in two registers and stores the output in the third register.

Ex - Or R1 R2 R3 => Performs logical OR of values stored in R1 and R2 and store result in R3.

(0101-001-010-011-000)

1. **NOT:**

Binary code - 0110

Performs logical Not of value stored in register and stores the output in the second register.

Ex - Not R1 R2 => Performs logical NOT of value stored in R1 and store result in R2.

(0110-001-000-010-000)

1. **XNOR:**

Binary code - 0111

Performs logical XNOR of values stored in two registers and stores the output in the third register.

Ex - Xnr R1 R2 R3 => Performs logical XNOR of values stored in R1 and R2 and store result in R3.

(0111-001-010-011-000)

**Data Transfer:**

1. **Load from memory:**

Binary code - 1001

Load data from memory to register.

Ex - Lod R1 (8) => Load data from memory location 8 to register R1.

(1000-001-00001000-0)

1. **Store to memory:**

Binary code - 1010

Store data from register to memory.

Ex - Str R1 (8) => Stores data to memory location 8 from register R1.

(1001-001-00001000-0)

1. **Move:**

Binary code - 1000

Move data between specified registers.

Ex - Mov R1 R2 => Move data from R1 to R2.

(1010-001-000-010-000)

**Branching:**

1. **Comparator:**

Binary Code - 1011

Compares two register values for equality, gives 1 for not equals and 0 for equals.

Ex - Cmp R1 R2 => Compares values in R1 and R2.

(1011-001-010-000000)

1. **Jump Unconditionally:**

Binary Code - 1100

Jumps Program Counter to PC + offset, where offset is the immediate value.

Ex - Jmp #5 => Increases PC by 5.

(1100-00000101-0100)

1. **Jump if less than:**

Binary Code - 1101

Jumps Program Counter to PC + offset, where offset is the immediate value if result from comparison is less than.

Ex - Jml #5 => Increases PC by 5 if result from comparison is less than.

(1101-00000101-0100)

1. **Jump if equals:**

Binary Code - 1110

Jumps Program Counter to PC + offset, where offset is the immediate value if result from comparison is equal.

Ex - Jme #5 => Increases PC by 5 if result from comparison is equal.

(1110-00000101-0100)

1. **Jump if greater than:**

Binary Code - 1110

Jumps Program Counter to PC + offset, where offset is the immediate value if result from comparison is greater than.

Ex - Jmg #5 => Increases PC by 5 if result from comparison is greater than.

(1111-00000101-0100)

In jump statements 2nd bit is given 1 to specify use of immediate value.