Formats:

For our ISA design we kept 3 Formats for the instruction.

R - Type,I - Type, J - Type

R - Type Format:

- Bit 0 - 2   destination register.
- Bit 3 - 5 source 2 register.
- Bit 6 - 8 source 1 register.
- Bit 9 - 13 opcode.

R - Type Format :

Opcode					Rs1			Rs2			Rd		
13	12	11	10	9	8	7	6	5	4	3	2	1	0

I - Type Format:

- Bit 0 - 2 contains an immediate value.
- Bit 3 - 5 destination register.
- Bit 6 - 8 source register.
- Bit 9 - 13 opcode.

I - Type Format :

Opcode					Rs			Rd			Immediate		
13	12	11	10	9	8	7	6	5	4	3	2	1	0

J - Type Format:

- Bit 0 - 8 contains an immediate value.
- Bit 9 - 13 opcode.

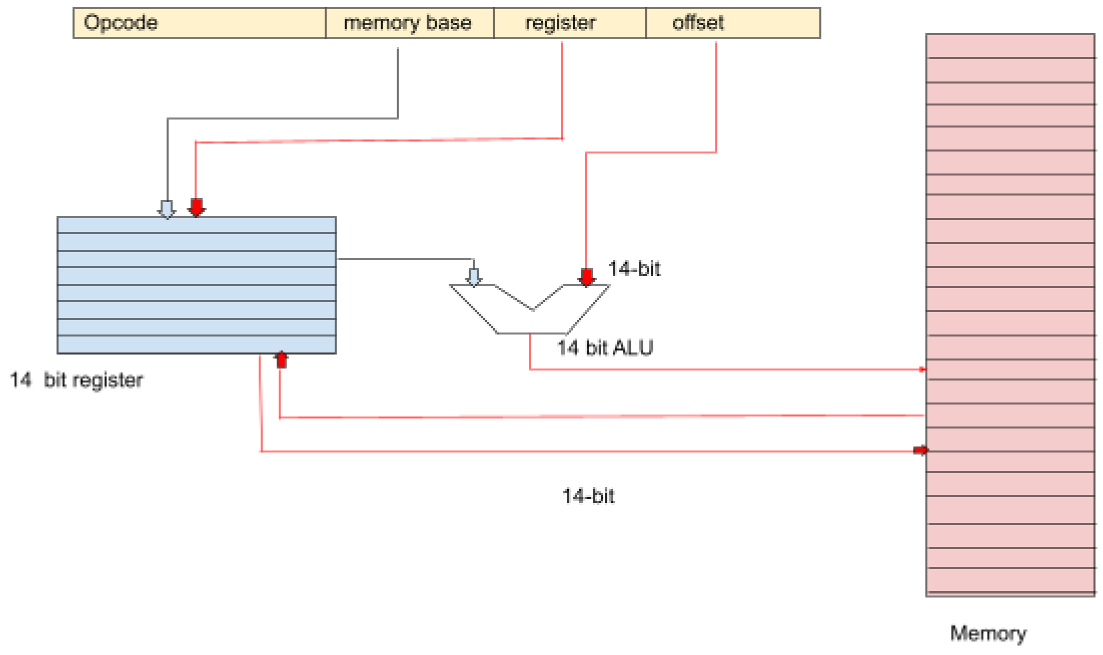
J - Type Format :

Opcode					Immediate								
13	12	11	10	9	8	7	6	5	4	3	2	1	0

<u>Instruction Table:</u>

No	***<u>Type</u>***	***<u>NAME</u>***	***<u>Format</u>***	***<u>EXAMPLE</u>***	***<u>Meaning</u>***	***<u>Opcode</u>***
01	Arithmetic	ADD	R	ADD R1, R2, R3	R3 = R1 + R2	00000
02	Arithmetic	ADDi	I	ADDi R1, R2, 5	R2 = R1 + 5	00001
03	Arithmetic	SUB	R	SUB R1, R2, R3	R3 = R1 - R2	00010
04	Arithmetic	SUBi	I	SUBi R1, R2, 5	R2 = R1 - 5	00011
05	Logical	AND	R	AND R1, R2, R3	R3 = R1 & R2	00100
06	Logical	OR	R	OR R1, R2, R3	R3 = R1 | R2	00101
07	Logical	XOR	R	XOR R1, R2, R3	R3 = R1 ^ R2	00110
08	Logical	SLL	I	SLL R1, R2, 4	R2 = R1 << 4;	00111
09	Logical	SRL	I	SRL R1, R2, 4	R2 = R1 >> 4;	01000
10	Conditional	COMP	R	COMP R1, R2, R3	If (R1 < R2) R3 = 1 else R3 = 0	01001
11	Conditional	COMPi	I	COMPi R1, R3, 5	If (R1 < 5) R3 = 1 else R3 = 0	01010
12	Conditional	EQL	I	EQL R1, R2, X	If (R1 == R2) branch to X	01011
13	Conditional	NEQL	I	NEQL R1, R2, X	If (R1 != R2) branch to X	01100
14	Unconditional	JUMP	J	JUMP label	Jumps to where label is	01101
15	Data Transfer	LOAD	I	LOAD R2, R1, 5	Loads value from memory to R1, takes R2 as base address and 5 as offset.	01110
16	Data Transfer	STOR	I	STOR R2 , R1, 5	Stores value saved in R1 into memory, takes R2 as base and 5 as offset.	01111
17	I/O	INPT	I	INPT R1	R1 = User Input	10000
18	I/O	OUT	I	OUT R1,	Print R1 to Display	10001

Memory (base) Addressing:



For Loop Execution:

```
for(i = 0; i <7; i++){
        sum += arr[i];
}cout << sum << endl;
[ Registers i = R1,  sum = R2, arr(base) = R3 ]
AND R1, ZERO, R1
#Start
COMPi R1, R4,  7
EQL R4, ZERO, #Break
ADD R3, R1, R4
LOAD R4, R5, 0
ADD R5, R2, R2
ADDi R1, R1, 1
JUMP #Start
#Break
OUT R
```