**Objectives:** Our objectives was to design a 16 Bit ISA which can solve a particular problems i. e. Simple arithmetic & logic operations, branching and loops.

**Operands:** Our goal is to use accumulator base ISA. For this reason we are going to take two operands.

**Types of Operands:** To implement arithmetic instruction we need register operands and for data transfer instruction from memory to register we need memory operands. So we need two types of operands.

* **Register based.**
* **Memory based.**

**Operations:** We will allocate 4 bits opcode, so the executable instructions number will be 24 or 16.

**Types of operations:** In our design there will be five different types of operation. The operations are:

* Arithmetic
* Logical
* Data Transfer
* Conditional Branch
* Unconditional Jump

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Category** | **Operation** | **Name** | **Type** | **Opcode** | **Syntax** | **Comments** |
| Unconditional | Jump | jmp | I | 0001 | Jmp 5 | Jump to line 5 |
| Conditional | Compare less  than | Slt | R | 0010 | Slt  rg1,rg2,rg0 | If(rg1<rg2)then rg0=1  Else rs0=0 |
| Logical | Bit –by-bit  and | and | R | 0011 | And  rg1,rg2,  rg3 | rg3=rg1&rg2 |
| Arithmetic | Add two  numbers | add | R | 0100 | Add  rg1,rg2,  rg3 | rg3=rg1+rg2 |
| Data Transfer | Store word | Sw | I | 0101 | Sw  rg3,rg0,5 | rg3=mem(rg0+5) |
| Arithmetic | subtraction | Sub | R | 0110 |  |  |
| Data transfer | Load word | Lw | I | 0111 | Lw  rg3,rg2,4 | rg3=mem(rg2+16) |
| Arithmetic | Add number  with an  immediate | addi | I | 1000 | rg1,rg1 2 | rg1=rg1+2 |
| Logical | Shift left | sll | I | 1001 | Sll  rg1,rg2,  rg3 | rg2=rg1<<rg3 |
| Conditional | Check  equality | beq | I | 1010 | Beq rg0,rg3,5 | If(rg3==rg0) then 5 |

**Formats:**

We would like to use two types of formats for our ISA. They are:

* **Register Type – R type**
* **Immediate Type – I type**
* **J type Isa format**

**List of Register:**

As we have allocated four bits register so the number of register will be 24  = 16.

For input and output purpose we included 2 register $in and $out. But we didn’t added them here as they can not be accessed by the the users.

|  |  |  |  |
| --- | --- | --- | --- |
| **Register Number** | **Conventional Name** | **Usage** | **Binary Value** |
| 0 | rg0 | General purpose | 0000 |
| 1 | rg1 | General purpose | 0001 |
| 2 | rg2 | General purpose | 0010 |
| 3 | rg3 | General purpose | 0011 |
| 4 | rg4 | General purpose | 0100 |
| 5 | rg5 | General purpose | 0101 |
| 6 | rg6 | General purpose | 0110 |
| 7 | rg7 | General purpose | 0111 |
| 8 | rg8 | General purpose | 1000 |
| 9 | rg9 | General purpose | 1001 |
| 10 | rg10 | General purpose | 1010 |
| 11 | rg11 | General purpose | 1011 |
| 12 | rg12 | General purpose | 1100 |
| 13 | rg13 | General purpose | 1101 |
| 14 | rg14 | General purpose | 1110 |
| 15 | rg15 | General purpose | 1111 |

**Translating Some HLL codes using our Designed 12 Bit ISA**

* + - 1. **r = p + q #rg1 = p, rg2 = q, rg3= r**

**add rg1, rg2, rg3 # rg3 gets rg1 + rg2**

* + - 1. **r = p – q #rg1 = p, rg2 = q, rg3= r**

**sub rg1, rg2,rg3 # rg3 gets rg1 – rg2**

* + - 1. **r= p AND q #rg1 = p, rg2 = q, rg3= r**

**And rg1, rg2, rg3 # rg3 gets rg1 && rg2**

* + - 1. **r = A[i] #rg1 = I, rg2=A , rg3=r**

**sll rg1,rg2, rg3 # rg1 = rg1<< rg3**

**add rg2,rg0,rg2 #rg2 = rg2 + rg0**

**lw 0 # getting the location value of A[i] from rg0 and store**

**5. if(i ==4)**

**i = i+2**

**else**

**i = i -2 # rg1 = i**

**sub rg0, rg0 # rg0 = 0**

**addi rg0, rg0, 4 # rg0 = 4**

**beq rg1, rg0 L #comparing rg1 = i with rg0 = 4. If equal then go to L**

**addi rg1,rg2, -2 # i = i – 2**

**jmp exit # Jump to exit**

**L: addi rg1, 2 # i = i + 2**

**6. for(int j = 3; j < p; j++){**

**q = q+8;**

**} # rg1 = j, rg2 = p, rg3 = q**

**sub rg1, rg1,rg1 # rg1 = 0**

**addi rg1, 3 # initializing rg1 = 3**

**sub rg0, rg0,rg0 # rg0 = 0**

**add rg0, rg2,rg0 # rg0 = rg2**

**L2: beq rg1 L1 #if rg1 = rg0 then go to L1, otherwise continue loop.**

**sll rg3, rg0 ,rg0 # rg0 = rg3 << rg0**

**add rg3 rg0 # rg3 = rg3 + 8**

**addi rg1 ,rg1 1 # increment rg1 by 1 / i++**

**jmp L2 # go to the loop**

**L1: exit.**