MSB is sign, rest is magnitude

Sign-m 1's comp 2's comp

22 00010110 00010110 00010110

-33(00100001) 10100001 11011110 11011111

11010101 -85

11010101(flip it bit by bit) -42

11010101(flip it bit by bit and add 1) -43

* **One’s Complement**

Dec Hex

0-9 0-9

10-15 A-F

* + Simply perform bit by bit invert.
  + For Ex: 123 is 01111011, -123 is 10000100
* **Two’s Complement**
  + Simply perform bit by bit invert and add 1
  + For Ex: 123 is 01111011, after bit by bit invert is: 10000100, after add 1 is: 10000101
  + **Two’s Complement to Decimal:**
    - Same as simple conversion except that MSB is negative.
    - Ex: 1101 is - 8+4+1= 3

Convert Instruction to :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP-6 | RS-5 ($s0) | RT-5 ($a0) | RD-5 ($t0) | SHAMT-5 | FUNC-5 |
| 000000 | 10000 | 00100 | 01000 | 00000 | 101010 |

1. slt $t0, $s0, $a0 - If $s0 < $a0, $t0 = 1, else $t0 = 0

Operation: R[rd] = (R[rs] < R[rt])?1:0

2. lw $s5, -8($v0)

Ans: I-Type

|  |  |  |  |
| --- | --- | --- | --- |
| OP -6 | RS -5 ($v0) | RT – 5 ($s5) | IMM – 16 (-8) |
| 100011 | 00010 | 10101 | 1111 1111 1111 0111 |

Note that logically shift left by 2 is the same as multiply by 4. The instruction **beq** is located at the address 2810 and the destination (**done:**) is located at the address 4410. Thus

PC = PC + 4 + (SignExt(Imm) \* 4)

44 = 28 + 4 + (SignExt(Imm) \* 4)

44 = 32 + (SignExt(Imm) \* 4)

12 = SignExt(Imm) \* 4

3 = SignExt(Imm)

3 = Imm

Op: 23hex 00100011

R[rt] = M[R[rs]+SignExtension]

**I-Type:**

Destination PC = currentPC + 4 +(SignEXT(imm) \* 4)

**J-Type:**

PC = address \* 4

Calculating : j xxxx, just take destination / 4.

**Caller:** The procedure that calls other procedure

**Callee**: The procedure that is called by other procedure.

* Callee may want to use same registers as caller
  + Values stored in those registers were destroyed after the callee use them
  + Usually temporary saved registers ($s0 to $s7)
  + Same situation as in $ra
* Caller may want to use same registers as callee
  + Similarly, values are destroyed.
  + Usually temporary registers ($t0 to $t9)
* $t0 – $t9:
  + These registers are temporary registers
  + Caller should not expect that values stored in these register will not be destroyed by callee
  + If caller wants to use them, caller must save their values before the call and restore their values back after the call
* $s0 – $s7:
  + These registers are temporary saved registers
  + Callee must maintain their values
  + If callee wants to use them, callee must save their values before using them and restore their values before return to caller.
* Caller saves needed registers, sets up arguments, makes call.
  + Saves temporary registers ($t0 to $t9) as needed
  + Sets up arguments ($a0 to $a3) as needed
  + If more arguments are required, put arguments onto the stack
* Callee procedure when entering the procedure
  + Adjust the stack pointer according to activation frame size to hold temporary saved registers, locals, return address (non-left)
  + Save return address to the stack
  + Save any temporary saved registers to the stack
* Callee procedure body
  + Access stack items as needed (additional arguments, locals, etc)
* Callee procedure before leaving the procedure
  + Restore any temporary saved registers
  + Restore return address
  + Restore the stack (deactivate activation frame)
  + Return to caller
* Consider a leaf procedure that needs to use all temporary registers $t0 to $t9 and three temporary saved registers $s0 to $s2
* According to the convention, this procedure only needs to save values stored in registers $s0, $s1, and $s2 (3 words or 12 bytes)
* When entering the procedure, it must perform the following:

lw $s2, 8($sp) # Restore $s1 in memory

lw $s1, 4($sp) # Restore $s1 in memory

lw $s0, 0($sp) # Restore $s0 in memory

addi $sp, $sp, 12 # Deallocate activation frame

addi $sp, $sp, -12 # Allocate activation frame

sw $s0, 0($sp) # Store $s0 in memory

sw $s1, 4($sp) # Store $s1 in memory

sw $s2, 8($sp) # Store $s1 in memory

* **Big Endian:** MSB in the smallest address
* **Little- Endian:** LSB in the smallest address