Ifkhartul Hakim Kawsar
1705045

Ans. to -1

~~addi $t3, $t2, 1~~

lui $t3, 2 # $t3's ~~two~~ upper part is 0010
ori $t3, 3 # $t3 contains 131075 now
~~add~~ add $t4, $t3, $t2 # $t4 contains y+131075
~~slt $t5, $t4, $t1~~
slt $t1, ~~$t5~~ $t4, $t1 # ~~x is~~ x/$t1 is set if y+131075<x

Ans. to -2

For "jal X", it goes as $ra = PC+4 and PC = 4X
For "JX", it only goes as PC = 4X
That means "jal X" jums and links as it sat the $ra register to the next instruction's location, so that after jumping back it can work from the next line. But, unconditional jump only makes PC to the specific locations.

Ans. to-3

In this case, compiler detects most frequent variables and dedicate register for them. Others are kept in memory. These variables are kept moving between memory and registers when needed. It means, when necessary, from memory, we move them to register, then moved back to memory.

Ans. to-4

In PC-relation addressing:—

If we are instructed to make a jump of X instructions (X is signed) or like this "beq $s0, $s1, X", it works as

$PC = PC + 4 + X*4$

Here, we see, PC makes a jump forward or backward relative to its next instruction. Because, during every clock cycle,

PC inherently increases by 4. This makes the whole execution faster, as there are usually less number of jumps. Otherwise, we would have to change PC after every instruction in an individual clock cycle.

So, it means, jumping distance or X should be calculated with respect to next ~~ne~~ instruction to current instruction.

If PC = 10000 now, and we want to make PC ~~20000~~ to point to ~~20000~~ 10500; when $s1 == $s2, we write,

beq $s1, $s2, 124

we see,

10000 + 4 + 124 × 4 = 10500