**Question 1.**

(a)


Identities:
VN3 = VN2 × VN4
VN5 = VN1 + VN3
VN7 = VN6 + VN3
VN10 = VN8 × VN9
VN12 = VN2 + VN11
Changed/added entries underlined in red
Defs shown in magenta
Value numbers to constants mapping in orange
localaddr vr6, $160
VN1
{vr6 → VN1}, {}
muli vr7, vr0, $8
VN3
{vr6 → VN1, vr0 → VN2, vr7 → VN3}, {VN4 → 8}
addi vr8, vr6, vr7
VN5
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5}, {VN4 → 8}
localaddr vr9, $0
VN6
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6}, {VN4 → 8}
muli vr10, vr0, $8
VN3
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3}, {VN4 → 8}
addi vr11, vr9, vr10
VN7
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7}, {VN4 → 8}
ldci vr12, $10
VN8
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7, vr12 → VN8}, {VN4 → 8, VN8 → 10}
ldi vr13, (vr11)
VN9
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7, vr12 → VN8, vr13 → VN9}, {VN4 → 8, VN8 → 10}
muli vr14, vr13, vr12
VN10
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7, vr12 → VN8, vr13 → VN9, vr14 → VN10}, {VN4 → 8, VN8 → 10}
sti (vr8), vr14
ldci vr2, $1
[unchanged]
VN11
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7, vr12 → VN8, vr13 → VN9, vr14 → VN10, vr2 → VN11}, {VN4 → 8, VN8 → 10, VN11 → 1}
addi vr3, vr0, vr2
VN12
{vr6 → VN1, vr0 → VN2, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7, vr12 → VN8, vr13 → VN9, vr14 → VN10, vr2 → VN11, vr3 → VN12}, {VN4 → 8, VN8 → 10, VN11 → 1}
mov vr0, vr3
VN12
{vr6 → VN1, vr0 → VN12, vr7 → VN3, vr8 → VN5, vr9 → VN6, vr10 → VN3, vr11 → VN7, vr12 → VN8, vr13 → VN9, vr14 → VN10, vr2 → VN11, vr3 → VN12}, {VN4 → 8, VN8 → 10, VN11 → 1}

(b) The instruction `muli vr10, vr0, $8` can be replaced with `mov vr10, vr7`. This substitution is possible because `vr7` contains VN3, which is the product of VN2 (the value in `vr0`) and VN4 (the constant value 8.)

**Question 2.**

spills/restores in orange

underlined in magenta: vr will not be used again, MR can be reclaimed

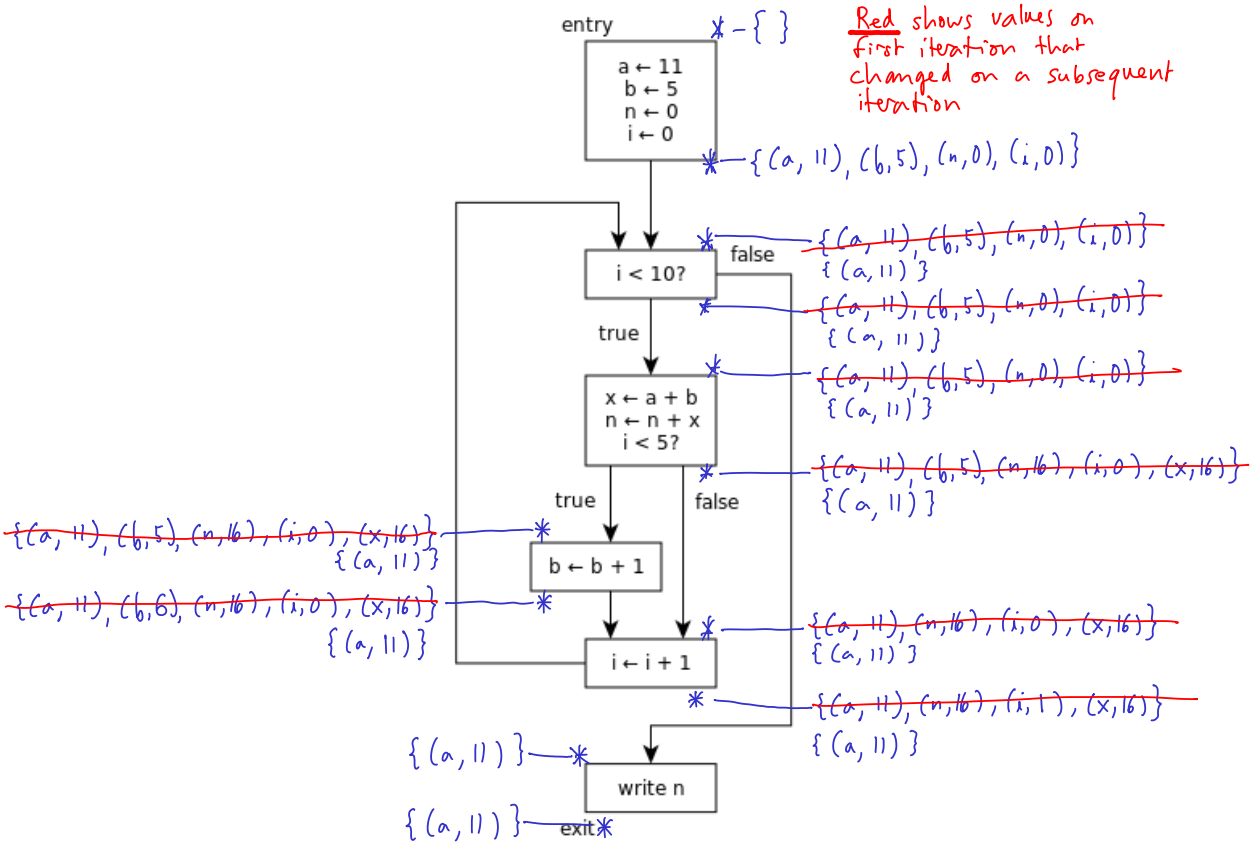
| | |
|---|---|
| localaddr vr19, $4000000 | vr19 → MR0 |
| muli vr20, vr0, $4000 | vr19 → MR0, vr20 → MR1 |
| addi vr21, vr19, vr20 | vr19 → MR0, vr20 → MR1, vr21 → MR2 |
| muli vr22, vr1, $8 | vr21 → MR2, vr22 → MR0 |
| addi vr23, vr21, vr22 | vr21 → MR2, vr22 → MR0, vr23 → MR1 |
| ldi vr4, (vr23) | vr23 → MR1, vr4 → MR0 |
| localaddr vr24, $2000000 | vr4 → MR0, vr24 → MR1 |
| muli vr25, vr2, $4000 | vr4 → MR0, vr24 → MR1, vr25 → MR2 |
| * spill vr4/MR0, loc 1 | |
| addi vr26, vr24, vr25 | vr24 → MR1, vr25 → MR2, vr26 → MR0 |
| muli vr27, vr1, $8 | vr26 → MR0, vr27 → MR1 |
| addi vr28, vr26, vr27 | vr26 → MR0, vr27 → MR1, vr28 → MR2 |
| ldi vr29, (vr28) | vr28 → MR2, vr29 → MR0 |
| muli vr30, vr3, vr29 | vr29 → MR0, vr30 → MR1 |
| * restore vr4, MR0, loc 1 | |
| addi vr31, vr4, vr30 | vr30 → MR1, vr4 → MR0, vr31 → MR2 |
| mov vr4, vr31 | vr31 → MR2, vr4 → MR0 |
| localaddr vr32, $4000000 | vr4 → MR0, vr32 → MR1 |
| muli vr33, vr0, $4000 | vr4 → MR0, vr32 → MR1, vr33 → MR2 |
| * spill vr4/MR0, loc 1 | |
| addi vr34, vr32, vr33 | vr32 → MR1, vr33 → MR2, vr34 → MR0 |
| muli vr35, vr1, $8 | vr34 → MR0, vr35 → MR1 |
| addi vr36, vr34, vr35 | vr34 → MR0, vr35 → MR1, vr36 → MR2 |
| * restore vr4, MR0, loc 1 | |
| sti (vr36), vr4 | vr36 → MR2, vr4 → MR0 |
| ldci vr5, $1 | vr5 → MR0 |
| addi vr6, vr1, vr5 | vr5 → MR0, vr6 → MR1 |
| mov vr1, vr6 | vr6 → MR1 |

**Question 3**.

(a.1) Combining {(p, 2), (q, 3), (r, 5)} and {(p, 2), (q, 4)} yields {(p, 2)}. I.e., only "p" has a specific known constant value. The variable "q" could be either 3 or 4, and "r" could be either 5 or some unknown value.

(a.2) The only members retained in the result set are the ones where the same variable is mapped to the same constant value. All other members are discarded. So, this is a *must* analysis.

(b)




Due to the loop, the values of variables "b", "n", and "i" change, while the value of "b" remains constant.

**Question 4 (628 only)**.

When dataflow values are combined, variables which have different values can be downgraded to special "positive" and "negative" values, if the original values were both positive or both negative.

For example, the dataflow values $\boxed{\{(a, 1), (b, 2), (c, -3), (d, 4)\}}$ and $\boxed{\{(a, 1), (b, 5), (c, -6), (d, -7)\}}$ could be combined to produce the value $\boxed{\{(a, 1), (b, positive), (c, negative)\}}$.

When modeling the effect of instructions in a basic block, the analysis must make conservative assumptions. For example, if the instruction $\boxed{n \leftarrow n + 1}$ is modeled, and the variable "n" currently has the value "positive", the analysis should remove the entry for "n", because integer overflow might cause "n" to become negative.