Karlsruhe Institute of Technology

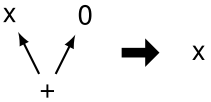
TRR 89

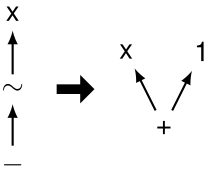
# OPTGEN: A Generator for Local Optimizations

**Sebastian Buchwald**

Institute for Program Structures and Data Organization, Karlsruhe Institute of Technology (KIT)

# Local Optimizations




**Local optimizations:**

- IR level
  - SSA form
  - Data dependency graph
- Do not require any global analysis
- Can be applied at any time during compilation

# Generation of Local Optimizations

Karlsruhe Institute of Technology

## Goal

Generate all local optimizations (up to a given cost limit).

**Input:**

- Set of operations and their costs
- Cost limit
- Bit width

**Output:**

- Complete set of verified local optimizations

# Related Work – Peephole Generators

## Assembly level

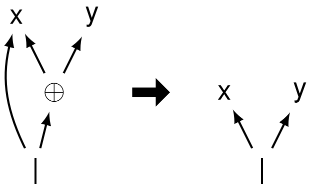
```
mov x, r0
mov y, r1
xor r0, r1, r2
...
...
or r0, r2, r3
```

➡

```
mov x, r0
mov y, r1
...
...
or r0, r1, r3
```

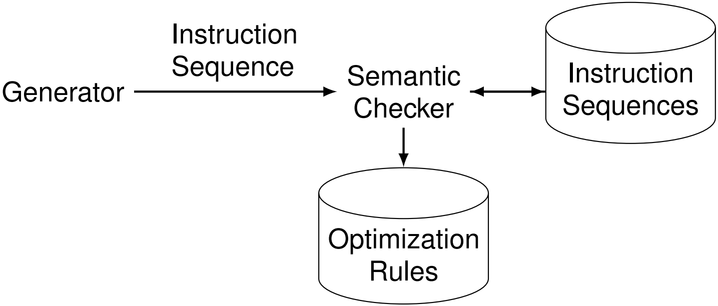
- Peephole of *k* instructions
- Architecture-specific
- Precise cost model

## IR level



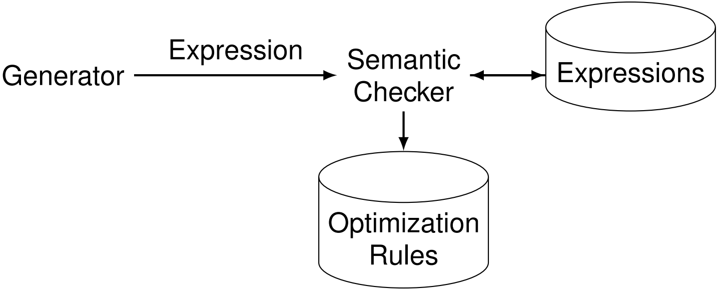

- Pattern of *k* values
- Independent of Architecture
- SSA form

# Common Design of Peephole Generators



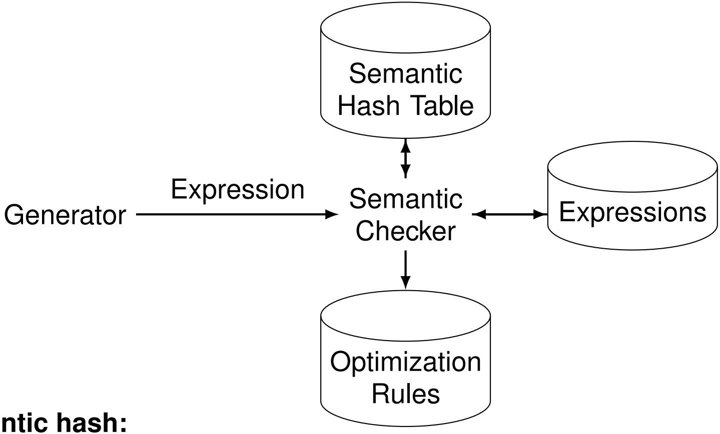

| | |
|---|---|
| Generator | Generates all possible instructions sequences |
| Semantic Checker | Proofs the equivalence of two instruction sequences |

# Design of OPTGEN (so far)




| | |
|---|---|
| Generator | Generates all possible expressions |
| Semantic Checker | Proofs the equivalence of two expressions |

# Design of OPTGEN (so far)




**Semantic hash:**

- Evaluate expression for precomputed test inputs
- `semantic_hash(x) = semantic_hash(x | 0)`

# Example

**OPTGEN parameters:**

- Operations:
  - Constants (cost: 0)
  - And (cost: 1)
  - Or (cost: 1)
  - Not (cost: 1)
- Cost limit: 2
- Bit width: 8

# Example – Costs 0

Karlsruhe Institute of Technology

**Enumerate expressions with costs 0:**

- `x`
- 0
- 1
- ...
- 255

# Example – Costs 1

## Combine expressions with existing operations:

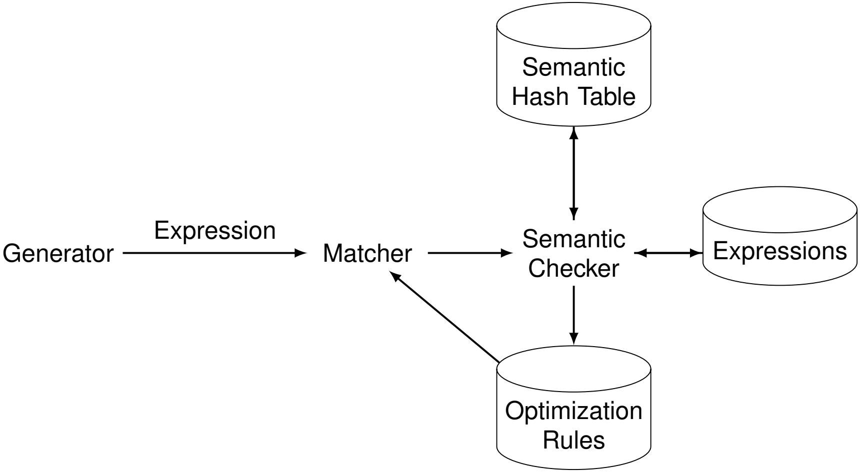
- `y`
- `x & x`
  - Same semantic hash class as `x`
  - SMT check: `x & x = x`
  - Optimization: `x & x` $\rightarrow$ `x`
- `x & 0`
  - Same semantic hash class as `0`
  - SMT check: `x & 0 = 0`
  - Optimization: `x & 0` $\rightarrow$ `0`

# Example – Costs 2

**Combine expressions with existing operations:**

- `(x & y) & 0`
  - Rule `x & 0` $\rightarrow$ `0` applicable
  - No further action

# Design of OPTGEN (so far)



# Example – Constant Folding Rules

## Constant folding rules:

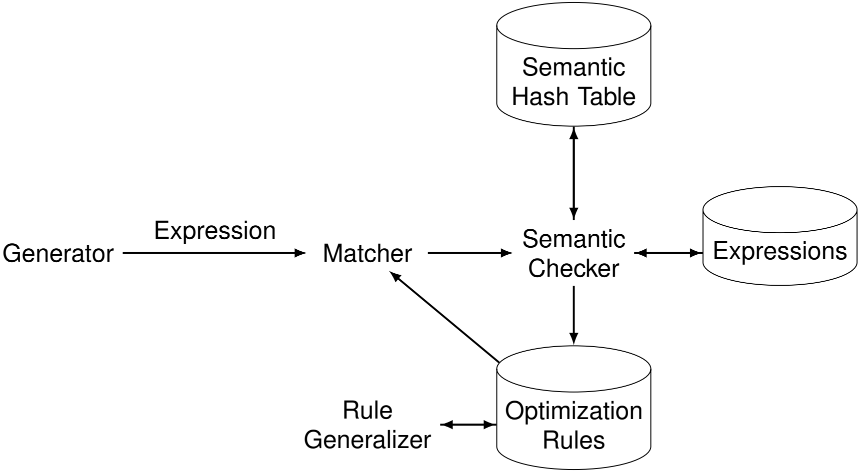
- `0 & 0` $\rightarrow$ `0`
- `0 & 1` $\rightarrow$ `0`
- `0 & 2` $\rightarrow$ `0`
- ...
- `255 & 255` $\rightarrow$ `255`

$2^{16}$ rules

## Expected rule:

- `c0 & c1` $\rightarrow$ `eval(c0 & c1)`

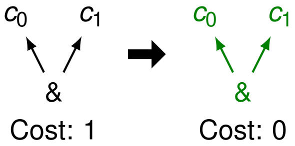
# Design of OPTGEN



# Example – Generalize Rules

**Generalize constant folding rules:**

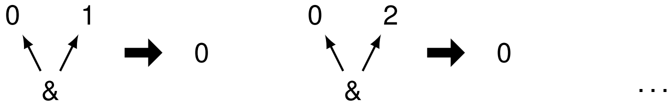
1. Introduce *symbolic constants*
   - Like variables
   - Allow constant folding



# Example – Generalize Rules

**Generalize constant folding rules:**

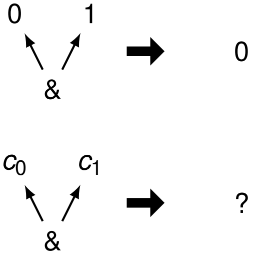
2. Collect syntactically equivalent rules



# Example – Generalize Rules

**Generalize constant folding rules:**

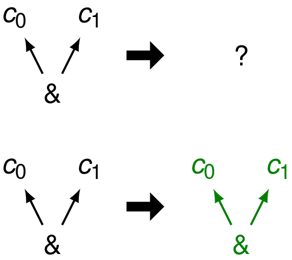
3. Replace constants of LHS with symbolic constants



# Example – Generalize Rules

**Generalize constant folding rules:**

4. Iterate through generated expressions to find appropriate RHS



# Example – Conditional Rules

## Symbolic rules not sufficient:

- `(x | 2) & 1` $\rightarrow$ `x & 1`
- `(x | 1) & 2` $\rightarrow$ `x & 2`
- `(x | 1) & 3` $\not\rightarrow$ `x & 3`

# Example – Conditional Rules

**Symbolic rules not sufficient:**

- `(x | 2) & 1` $\rightarrow$ `x & 1`
- `(x | 1) & 2` $\rightarrow$ `x & 2`
- `(x | 1) & 3` $\not\rightarrow$ `x & 3`

**Solution:**

- Conditional rule: `c0 & c1 == 0` $\Rightarrow$ `(x | c0) & c1` $\rightarrow$ `x & c1`
- Iterate through generated expressions to find appropriate condition
  - Condition: `c0 & c1 == 0`

# Example – Result

OPTGEN finds 42 optimizations:

- 19 rules with symbolic constants
  - 8 rules with condition
  - 11 rules without condition
- 12 rules with non-symbolic constants
- 11 rules without constants

# Example – Result

OPTGEN finds 42 optimizations:

- 19 rules with symbolic constants
  - 8 rules with condition
  - 11 rules without condition
- 12 rules with non-symbolic constants
- 11 rules without constants

**Question**

What happens if we use a bit width of 32 bit?

# Example – Result

OPTGEN finds 42 optimizations:

- 19 rules with symbolic constants
  - 8 rules with condition
  - 11 rules without condition
- 12 rules with non-symbolic constants
- 11 rules without constants

## Question

What happens if we use a bit width of 32 bit?

# Extension to 32 Bit: Correctness

**Basic idea:**

- Generate rules for 8 bit
- Extend rules from 8 bit to 32 bit
- Verify extended rules for 32 bit

**Extension of bit width:**

- Rules without non-symbolic constants
  - Independent of bit width
  - `x & x` $\rightarrow$ `x`
- Rules with non-symbolic constants
  - Try to prepend or append 0/1 bits
  - `x & 0xFF` $\rightarrow$ `x`
    - `x & 0xFF 000000` $\rightarrow$ `x`
    - `x & 0xFF FFFFFF` $\rightarrow$ `x`
    - `x & 0x000000 FF` $\rightarrow$ `x`
    - `x & 0xFFFFFF FF` $\rightarrow$ `x`
  - Works fine in practice

# Extension to 32 Bit: Completeness

Karlsruhe Institute of Technology

**Basic idea:**

- Increase bit width until the number of rules stabilizes

| Bit width | Number of rules |
|---|---|
| 1 | 24 |
| 2 | 38 |
| 3 | 42 |
| 4 | 42 |
| ... | ... |
| 32 | 42 |

**Drawback:**

- Does not work for all operations

# Evaluation

**Full run:**

- Operations: Constants, Minus, Not, Add, And, Or, Sub, Xor
- Cost limit: 2
- Generation: 8 bit
- Verification: 32 bit
- 6 h 7 min 0 s
- 1 046 568 kB

**Testsuite:**

- LLVM: 23 missing optimizations
- GCC: 27 missing optimizations
- ICC: 62 missing optimizations

# Optimization Differences

| | Optimization | LLVM | GCC | ICC |
|---|---|---|---|---|
| 2. | `-(x & 0x80000000)` $\rightarrow$ `x & 0x80000000` | $\times$ | ✓ | $\times$ |
| 6. | `(x | 0x80000000) + 0x80000000` $\rightarrow$ `x & 0x7FFFFFFF` | ✓ | $\times$ | $\times$ |
| 11. | `x & (x + 0x80000000)` $\rightarrow$ `x & 0x7FFFFFFF` | ✓ | $\times$ | $\times$ |
| 14. | `-x & 1` $\rightarrow$ `x & 1` | $\times$ | ✓ | $\times$ |
| 17. | `x | (x + 0x80000000)` $\rightarrow$ `x | 0x80000000` | ✓ | $\times$ | $\times$ |
| 20. | `x | (x` $\oplus$ `y)` $\rightarrow$ `x | y` | ✓ | $\times$ | $\times$ |
| $*$ 21. | `((c0 | -c0) & ~c1) == 0` $\Rightarrow$ `(x + c0) | c1` $\rightarrow$ `x | c1` | ✓ | $\times$ | ✓ |
| 25. | `0 - (x & 0x80000000)` $\rightarrow$ `x & 0x80000000` | $\times$ | ✓ | $\times$ |
| 30. | `x` $\oplus$ `(x + 0x80000000)` $\rightarrow$ `0x80000000` | ✓ | $\times$ | $\times$ |
| 35. | `(0x7FFFFFFF - x)` $\oplus$ `0x80000000` $\rightarrow$ `~x` | $\times$ | ✓ | $\times$ |
| 36. | `(0x80000000 - x)` $\oplus$ `0x80000000` $\rightarrow$ `-x` | $\times$ | ✓ | $\times$ |
| 43. | `~(x + c)` $\rightarrow$ `~c - x` | ✓ | $\times$ | $\times$ |
| 54. | `~(c - x)` $\rightarrow$ `x + ~c` | ✓ | $\times$ | $\times$ |
| 60. | `(c0 & ~c1) == 0` $\Rightarrow$ `(x` $\oplus$ `c0) | c1` $\rightarrow$ `x | c1` | ✓ | $\times$ | $\times$ |
| Missing optimizations | | 5 | 9 | 13 (+ 32) |

Compiler: LLVM, GCC, ICC

# Unsupported Optimizations

| | Optimization | LLVM | GCC | ICC |
|---|---|---|---|---|
| 5. | `x + (x & 0x80000000)` → `x & 0x7FFFFFFF` | × | × | × |
| 13. | `x & (0x7FFFFFFF - x)` → `x & 0x80000000` | × | × | × |
| ∗ 16. | `is_power_of_2(c1) && c0 & (2 * c1 - 1) == c1 - 1` | | | |
| | ⇒ `(c0 - x) & c1` → `x & c1` | × | × | × |
| 19. | `x \| (0x7FFFFFFF - x)` → `x \| 0x7FFFFFFF` | × | × | × |
| ∗ 22. | `is_power_of_2(∼c1) && c0 & (2 * ∼c1 - 1) == ∼c1 - 1` | | | |
| | ⇒ `(c0 - x) \| c1` → `x \| c1` | × | × | × |
| 23. | `-x \| 0xFFFFFFFE` → `x \| 0xFFFFFFFE` | × | × | × |
| 26. | `0x7FFFFFFF - (x & 0x80000000)` → `x \| 0x7FFFFFFF` | × | × | × |
| 27. | `0x7FFFFFFF - (x \| 0x7FFFFFFF)` → `x & 0x80000000` | × | × | × |
| 28. | `0xFFFFFFFE - (x \| 0x7FFFFFFF)` → `x \| 0x7FFFFFFF` | × | × | × |
| 29. | `(x & 0x7FFFFFFF) - x` → `x & 0x80000000` | × | × | × |
| 31. | `x ⊕ (0x7FFFFFFF - x)` → `0x7FFFFFFF` | × | × | × |
| 32. | `(x + 0x7FFFFFFF) ⊕ 0x7FFFFFFF` → `-x` | × | × | × |
| 34. | `-x ⊕ 0x80000000` → `0x80000000 - x` | × | × | × |
| 39. | `(0x7FFFFFFF - x) ⊕ 0x7FFFFFFF` → `x` | × | × | × |
| 48. | `-x ⊕ 0x7FFFFFFF` → `x + 0x7FFFFFFF` | × | × | × |
| 52. | `(x \| c) - c` → `x & ∼c` | × | × | × |
| 57. | `-c0 == c1` ⇒ `(x \| c0) + c1` → `x & ∼c1` | × | × | × |
| 62. | `0x7FFFFFFF - (x ⊕ c)` → `x ⊕ (0x7FFFFFFF - c)` | × | × | × |

# Conclusion

OPTGEN

- is the first generator that supports arbitrary constants
- guarantees correctness and completeness of generated optimizations
- has revealed missing optimizations in all state-of-the-art compilers

**There is more wisdom in the paper.**

# No

# Optimizations 1/5

| | Optimization | LLVM | GCC | ICC |
|---|---|---|---|---|
| 1. | `-~x` $\rightarrow$ `x + 1` | ✓ | ✓ | × |
| 2. | `-(x & 0x80000000)` $\rightarrow$ `x & 0x80000000` | × | ✓ | × |
| 3. | `~-x` $\rightarrow$ `x - 1` | ✓ | ✓ | × |
| 4. | `x + ~x` $\rightarrow$ `0xFFFFFFFF` | ✓ | ✓ | × |
| 5. | `x + (x & 0x80000000)` $\rightarrow$ `x & 0x7FFFFFFF` | × | × | × |
| 6. | `(x \| 0x80000000) + 0x80000000` $\rightarrow$ `x & 0x7FFFFFFF` | ✓ | × | × |
| 7. | `(x & 0x7FFFFFFF) + (x & 0x7FFFFFFF)` $\rightarrow$ `x + x` | ✓ | ✓ | × |
| 8. | `(x & 0x80000000) + (x & 0x80000000)` $\rightarrow$ `0` | ✓ | ✓ | × |
| 9. | `(x \| 0x7FFFFFFF) + (x \| 0x7FFFFFFF)` $\rightarrow$ `0xFFFFFFFE` | ✓ | ✓ | × |
| 10. | `(x \| 0x80000000) + (x \| 0x80000000)` $\rightarrow$ `x + x` | ✓ | ✓ | × |
| 11. | `x & (x + 0x80000000)` $\rightarrow$ `x & 0x7FFFFFFF` | ✓ | × | × |
| 12. | `x & (x \| y)` $\rightarrow$ `x` | ✓ | ✓ | × |
| 13. | `x & (0x7FFFFFFF - x)` $\rightarrow$ `x & 0x80000000` | × | × | × |
| 14. | `-x & 1` $\rightarrow$ `x & 1` | × | ✓ | × |
| 15. | `(x + x) & 1` $\rightarrow$ `0` | ✓ | ✓ | × |
| 16. | `is_power_of_2(c1) && c0 & (2 * c1 - 1) == c1 - 1` $\Rightarrow$ `(c0 - x) & c1` $\rightarrow$ `x & c1` | × | × | × |
| Sum | | 23 | 27 | 62 |

# Optimizations 2/5

| Optimization | LLVM | GCC | ICC |
|---|---|---|---|
| 17. `x | (x + 0x80000000)` → `x | 0x80000000` | ✓ | × | × |
| 18. `x | (x & y)` → `x` | ✓ | ✓ | × |
| 19. `x | (0x7FFFFFFF - x)` → `x | 0x7FFFFFFF` | × | × | × |
| 20. `x | (x` ⊕ `y)` → `x | y` | ✓ | × | × |
| 21. `((c0 | -c0) & ~c1) == 0` ⇒ `(x + c0) | c1` → `x | c1` | ✓ | × | ✓ |
| 22. `is_power_of_2(~c1) && c0 & (2 * ~c1 - 1) == ~c1 - 1` ⇒ `(c0 - x) | c1` → `x | c1` | × | × | × |
| 23. `-x | 0xFFFFFFFE` → `x | 0xFFFFFFFE` | × | × | × |
| 24. `(x + x) | 0xFFFFFFFE` → `0xFFFFFFFE` | ✓ | ✓ | × |
| 25. `0 - (x & 0x80000000)` → `x & 0x80000000` | × | ✓ | × |
| 26. `0x7FFFFFFF - (x & 0x80000000)` → `x | 0x7FFFFFFF` | × | × | × |
| 27. `0x7FFFFFFF - (x | 0x7FFFFFFF)` → `x & 0x80000000` | × | × | × |
| 28. `0xFFFFFFFE - (x | 0x7FFFFFFF)` → `x | 0x7FFFFFFF` | × | × | × |
| 29. `(x & 0x7FFFFFFF) - x` → `x & 0x80000000` | × | × | × |
| 30. `x` ⊕ `(x + 0x80000000)` → `0x80000000` | ✓ | × | × |
| 31. `x` ⊕ `(0x7FFFFFFF - x)` → `0x7FFFFFFF` | × | × | × |
| 32. `(x + 0x7FFFFFFF)` ⊕ `0x7FFFFFFF` → `-x` | × | × | × |
| Sum | 23 | 27 | 62 |

# Optimizations 3/5

| | Optimization | LLVM | GCC | ICC |
|---|---|---|---|---|
| 33. | $(x + 0x80000000) \oplus 0x7FFFFFFF \rightarrow \sim x$ | ✓ | ✓ | × |
| 34. | $-x \oplus 0x80000000 \rightarrow 0x80000000 - x$ | × | × | × |
| 35. | $(0x7FFFFFFF - x) \oplus 0x80000000 \rightarrow \sim x$ | × | ✓ | × |
| 36. | $(0x80000000 - x) \oplus 0x80000000 \rightarrow -x$ | × | ✓ | × |
| 37. | $(x + 0xFFFFFFFF) \oplus 0xFFFFFFFF \rightarrow -x$ | ✓ | ✓ | × |
| 38. | $(x + 0x80000000) \oplus 0x80000000 \rightarrow x$ | ✓ | ✓ | × |
| 39. | $(0x7FFFFFFF - x) \oplus 0x7FFFFFFF \rightarrow x$ | × | × | × |
| 40. | $x - (x \,\&\, c) \rightarrow x \,\&\, \sim c$ | ✓ | ✓ | × |
| 41. | $x \oplus (x \,\&\, c) \rightarrow x \,\&\, \sim c$ | ✓ | ✓ | × |
| 42. | $\sim x + c \rightarrow (c - 1) - x$ | ✓ | ✓ | × |
| 43. | $\sim(x + c) \rightarrow \sim c - x$ | ✓ | × | × |
| 44. | $-(x + c) \rightarrow -c - x$ | ✓ | ✓ | × |
| 45. | $c - \sim x \rightarrow x + (c + 1)$ | ✓ | ✓ | × |
| 46. | $\sim x \oplus c \rightarrow x \oplus \sim c$ | ✓ | ✓ | × |
| 47. | $\sim x - c \rightarrow \sim c - x$ | ✓ | ✓ | × |
| 48. | $-x \oplus 0x7FFFFFFF \rightarrow x + 0x7FFFFFFF$ | × | × | × |
| Sum | | 23 | 27 | 62 |

# Optimizations 4/5

| Optimization | | Compiler | | |
|---|---|---|---|---|
| | | LLVM | GCC | ICC |
| 49. | $-x \oplus 0xFFFFFFFF \rightarrow x - 1$ | ✓ | ✓ | × |
| 50. | $x \,\&\, (x \oplus c) \rightarrow x \,\&\, \sim c$ | ✓ | ✓ | × |
| 51. | $-x - c \rightarrow -c - x$ | ✓ | ✓ | × |
| 52. | $(x \mid c) - c \rightarrow x \,\&\, \sim c$ | × | × | × |
| 53. | $(x \mid c) \oplus c \rightarrow x \,\&\, \sim c$ | ✓ | ✓ | × |
| 54. | $\sim(c - x) \rightarrow x + \sim c$ | ✓ | × | × |
| 55. | $\sim(x \oplus c) \rightarrow x \oplus \sim c$ | ✓ | ✓ | × |
| 56. | $\sim c0 == c1 \Rightarrow (x \,\&\, c0) \oplus c1 \rightarrow x \mid c1$ | ✓ | ✓ | × |
| 57. | $-c0 == c1 \Rightarrow (x \mid c0) + c1 \rightarrow x \,\&\, \sim c1$ | × | × | × |
| 58. | $(x \oplus c) + 0x80000000 \rightarrow x \oplus (c + 0x80000000)$ | ✓ | ✓ | × |
| 59. | $((c0 \mid -c0) \,\&\, c1) == 0 \Rightarrow (x \oplus c0) \,\&\, c1 \rightarrow x \,\&\, c1$ | ✓ | ✓ | × |
| 60. | $(c0 \,\&\, \sim c1) == 0 \Rightarrow (x \oplus c0) \mid c1 \rightarrow x \mid c1$ | ✓ | × | × |
| 61. | $(x \oplus c) - 0x80000000 \rightarrow x \oplus (c + 0x80000000)$ | ✓ | ✓ | × |
| 62. | $0x7FFFFFFF - (x \oplus c) \rightarrow x \oplus (0x7FFFFFFF - c)$ | × | × | × |
| 63. | $0xFFFFFFFF - (x \oplus c) \rightarrow x \oplus (0xFFFFFFFF - c)$ | ✓ | ✓ | × |
| Sum | | 23 | 27 | 62 |

# Optimizations 5/5

| Optimization | | Compiler | | |
|---|---|---|---|---|
| | | LLVM | GCC | ICC |
| 1. | $\sim$(x \| $\sim$y) $\rightarrow$ $\sim$x & y | $\times$ | $\checkmark$ | |
| 2. | $\sim$(x & $\sim$y) $\rightarrow$ $\sim$x \| y | $\times$ | $\checkmark$ | |
| 3. | (x + x) & (y + y) $\rightarrow$ (x & y) + (x & y) | $\times$ | | |
| 4. | (x + x) \| (y + y) $\rightarrow$ (x \| y) + (x \| y) | $\times$ | | |
| 5. | (x & y) \| (z & y) $\rightarrow$ y & (x \| z) | $\checkmark$ | $\times$ | $\checkmark$ |
| 6. | x - ((x - y) + (x - y)) $\rightarrow$ y + (y - x) | | $\checkmark$ | $\times$ |
| 7. | (x - y) - (x + z) $\rightarrow$ -(y + z) | $\checkmark$ | $\checkmark$ | $\times$ |
| 8. | ((x - y) + (x - y)) - x $\rightarrow$ x - (y + y) | $\checkmark$ | $\checkmark$ | $\times$ |
| 9. | (x + x) $\oplus$ (y + y) $\rightarrow$ (x $\oplus$ y) + (x $\oplus$ y) | $\times$ | | |
| 10. | (x & y) $\oplus$ (z & y) $\rightarrow$ y & (x $\oplus$ z) | $\checkmark$ | $\times$ | $\checkmark$ |

State-of-the-art compilers apply optimizations rules even if the operands are shared. If the compiler supports the optimization $\checkmark/\times$ indicates whether the compiler prevents the optimization in case of shared operands. If the compiler does not support the optimization the item is left blank.