# Bitwise Operation Dry Run Exercises

The following examples illustrate dry runs of C++ programs that focus on bitwise operations (using only <iostream>). Each snippet is followed by a step-by-step table showing variable states and logic. The final row (or explanation) gives the output of the program. The exercises are grouped by difficulty: **Extremely Hard**, **Medium-Hard**, and **Medium**.

## Extremely Hard Bitwise Dry Runs

These examples combine loops, bit shifts, and complex bitwise expressions to create very challenging puzzles.

### Dry Run 1: Bitwise Loop with XOR and OR

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int a = 17, b = 31, c = 0;  
 for(int i = 0; i < 5; i++){  
 if (a & (1 << i)) {  
 a = a ^ (b >> i);  
 c |= (b & (1 << (i+1)));  
 } else {  
 b <<= i;  
 c ^= (a & (1 << (i+2)));  
 }  
 }  
 cout << (c | (a & b));  
 return 0;  
}

| Step | i | a | b | c | Explanation |
| --- | --- | --- | --- | --- | --- |
| 1 | – | 17 | 31 | 0 | **Initialize**: a=17 (10001₂), b=31 (11111₂), c=0. |
| 2 | 0 | 14 | 31 | 2 | **i=0 (bit 1)**: (a&1)=1, so a=17^(31>>0)=17^31=14; c |= (31&(1<<1))=2. |
| 3 | 1 | 1 | 31 | 6 | **i=1 (bit 2)**: (a&(1<<1))=14&2=2, so a=14^(31>>1)=14^15=1; c |= (31&(1<<2))=4 (c becomes 6). |
| 4 | 2 | 1 | 124 | 6 | **i=2 (bit 3)**: (a&(1<<2))=1&4=0, so false branch. b <<= 2 → b=31<<2=124; c ^= (1&(1<<4))=0. |
| 5 | 3 | 1 | 992 | 6 | **i=3 (bit 4)**: (a&(1<<3))=1&8=0, false branch. b <<= 3 → b=124<<3=992; c ^= (1&(1<<5))=0. |
| 6 | 4 | 1 | 15872 | 6 | **i=4 (bit 5)**: (a&(1<<4))=1&16=0, false branch. b <<= 4 → b=992<<4=15872; c ^= (1&(1<<6))=0. |
| 7 | – | – | – | 6 | **Output**: Compute (c | (a & b)) = 6 | (1 & 15872) = 6. |

The code prints **6**.

### Dry Run 2: Nested Loops with Bitwise AND/OR/XOR

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int a = 3, b = 5;  
 int c = 0;  
 for(int i = 0; i < 3; i++){  
 for(int j = 0; j < 3; j++){  
 a = a << (b & 1);  
 if ((a & (1 << j)) != 0) {  
 c = (c << 1) | (b & 1);  
 } else {  
 c = (c >> 1) ^ ((~a) & 1);  
 }  
 b = b >> j;  
 }  
 }  
 cout << c;  
 return 0;  
}

| Step | i | j | a | b | c | Explanation |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 6 | 5 | 1 | **Init**: a=3(11₂), b=5(101₂), c=0. b&1=1 so a <<=1→a=6. (a&1)==0, so c=(0>>1)^(~6&1)=1. Then b>>=0→b=5. |
| 2 | 0 | 1 | 12 | 2 | 1 | b&1=1 so a<<=1→a=12. (a&(1<<1))=12&2=0, so c=(1>>1)^(~12&1)=1. b>>=1→b=2. |
| 3 | 0 | 2 | 12 | 0 | 2 | b&1=0 so a<<=0→a=12. (a&(1<<2))=12&4=4≠0, so c=(1<<1)|(0)=2. b>>=2→b=0. |
| 4 | 1 | 0 | 12 | 0 | 0 | b&1=0 so a<<=0→a=12. (a&1)=12&1=0, so c=(2>>1)^(~12&1)=0. b>>=0→b=0. |
| 5 | 1 | 1 | 12 | 0 | 1 | b&1=0 so a<<=0→a=12. (a&(1<<1))=12&2=0, so c=(0>>1)^(~12&1)=1. b>>=1→b=0. |
| 6 | 1 | 2 | 12 | 0 | 2 | b&1=0 so a<<=0→a=12. (a&(1<<2))=12&4=4≠0, so c=(1<<1)|(0)=2. b>>=2→b=0. |
| 7 | 2 | 0 | 12 | 0 | 0 | b&1=0 so a<<=0→a=12. (a&1)=12&1=0, so c=(2>>1)^(~12&1)=0. b>>=0→b=0. |
| 8 | 2 | 1 | 12 | 0 | 1 | b&1=0 so a<<=0→a=12. (a&(1<<1))=12&2=0, so c=(0>>1)^(~12&1)=1. b>>=1→b=0. |
| 9 | 2 | 2 | 12 | 0 | 2 | b&1=0 so a<<=0→a=12. (a&(1<<2))=12&4=4≠0, so c=(1<<1)|(0)=2. b>>=2→b=0. |
| 10 | – | – | – | – | 2 | **Output**: c is 2. |

The code prints **2**.

### Dry Run 3: Bitwise NOT and Shifts with Negative Number

#include <iostream>  
using namespace std;  
  
int main() {  
 signed int x = -5;  
 unsigned int y = 4;  
 int result = 0;  
 for (int i = 0; i < 4; i++) {  
 if (x & (1 << i)) {  
 result += (x << i) ^ (y >> (3 - i));  
 } else {  
 result -= (~(x >> i) & y);  
 }  
 }  
 cout << result;  
 return 0;  
}

| Step | i | x | y | x&(1<<i) | result | Explanation |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | -5 | 4 | 1 ≠ 0 | -5 | **Init**: x=-5 (…11111011₂), y=4 (100₂). (x&(1<<0))=1. True branch: (x<<0)^(y>>3) = -5^0 = -5. result = -5. |
| 2 | 1 | -5 | 4 | 2 ≠ 0 | -14 | (x&(1<<1))=2. True: (x<<1)^(y>>2) = -10 ^ 1 = -9. result = -5 + (-9) = -14. |
| 3 | 2 | -5 | 4 | 0 | -14 | (x&(1<<2))=0. False: ~(x>>2)&y = ~( -2 ) & 4. x>>2 = -2 (…11111110₂), ~(-2)=1. 1&4=0. result -= 0 → -14. |
| 4 | 3 | -5 | 4 | 8 ≠ 0 | -50 | (x&(1<<3))=8. True: (x<<3)^(y>>0) = -40 ^ 4 = -36. result = -14 + (-36) = -50. |
| 5 | – | – | – | – | -50 | **Output**: result = -50. |

The code prints **-50**.

### Dry Run 4: Bitwise Conditions with XOR and Shifts

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int x = 9;  
 unsigned int res = 1;  
 for(int i = 0; i < 4; i++){  
 if (x & 1) {  
 res <<= (x & (1 << i));  
 } else {  
 res ^= (x << i);  
 }  
 x ^= (res >> 1);  
 }  
 cout << res;  
 return 0;  
}

| Step | i | x | res | Explanation |
| --- | --- | --- | --- | --- |
| 1 | 0 | 9 | 2 | **Init**: x=9(1001₂), res=1. x&1=1 so res <<= (9&(1<<0)) = 1<<1 = 2. Then x ^= res>>1: x = 9 ^ (2>>1=1) = 8. |
| 2 | 1 | 8 | 18 | x&1=0 so res ^= (8<<1) = 2 ^ 16 = 18. Then x ^= (18>>1=9): x = 8 ^ 9 = 1. |
| 3 | 2 | 1 | 18 | x&1=1 so res <<= (1&(1<<2)=0) = 18<<0 = 18. Then x ^= (18>>1=9): x = 1 ^ 9 = 8. |
| 4 | 3 | 8 | 82 | x&1=0 so res ^= (8<<3) = 18 ^ 64 = 82. Then x ^= (82>>1=41): x = 8 ^ 41 = 33. |
| 5 | – | 33 | 82 | **Output**: res = 82. |

The code prints **82**.

### Dry Run 5: Bitwise NOT, Shifts, and Conditional OR

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int x = 13, y = 7, z = 0;  
 for(int i = 0; i < 3; i++) {  
 unsigned int t = x;  
 x = (x ^ y) << (y & 3);  
 y = (t & y) >> 1;  
 if (x & (1 << (i+1))) {  
 z |= (x << (2 - i));  
 } else {  
 z |= (~y << i);  
 }  
 }  
 cout << z;  
 return 0;  
}

| Step | i | x | y | z | Explanation |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 80 | 2 | 4294967293 | **Init**: x=13, y=7. Compute t=13. Then x=(13^7)<<(7&3)=10<<3=80, y=(13&7)>>1=5>>1=2. (x&(1<<1))=80&2=0, so z |= (~2<<0)=~2 = 0xFFFFFFFD. |
| 2 | 1 | 328 | 0 | 4294967295 | t=80. x=(80^2)<<(2&3)=82<<2=328, y=(80&2)>>1=0. (x&(1<<2))=328&4=0, so z |= (~0<<1)=0xFFFFFFFE. Now z=0xFFFFFFFF. |
| 3 | 2 | 328 | 0 | 4294967295 | t=328. x=(328^0)<<(0&3)=328, y=(328&0)>>1=0. (x&(1<<3))=328&8=8≠0, so z |= (328<<(0)). But z is already 0xFFFFFFFF. |
| 4 | – | 328 | 0 | 4294967295 | **Output**: z = 4294967295. |

The code prints **4294967295**.

## Medium-Hard Bitwise Dry Runs

These exercises are challenging but a bit simpler. They involve basic loops or conditions with bitwise operators.

### Dry Run 6: Summing Shifted ANDs

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int n = 10;  
 int total = 0;  
 for(int i = 0; i < 4; i++){  
 total += ((n << i) & (n << (i+1)));  
 }  
 cout << total;  
 return 0;  
}

| Step | i | (n<<i) | (n<<(i+1)) | (n<<i)&(n<<(i+1)) | Explanation |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 10 | 20 | 8 | 10(01010₂) & 20(10100₂) = 8(01000₂). total = 0+8 = 8. |
| 2 | 1 | 20 | 40 | 0 | 20(10100₂)&40(101000₂) = 0. total = 8+0 = 8. |
| 3 | 2 | 40 | 80 | 0 | 40(101000₂)&80(1010000₂) = 0. total = 8+0 = 8. |
| 4 | 3 | 80 | 160 | 0 | 80(1010000₂)&160(10100000₂) = 0. total = 8+0 = 8. |
| 5 | – | – | – | – | **Output**: total = 8. |

The code prints **8**.

### Dry Run 7: Simple Bitwise Conditional

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int a = 5, b = 9;  
 int result = 1;  
 if ((a & b) & 1) {  
 result = (a << 1) ^ (b >> 1);  
 } else {  
 result = (a & (~b)) | ((~(a ^ b)) << 1);  
 }  
 cout << result;  
 return 0;  
}

| Step | Expression | Value | Explanation |
| --- | --- | --- | --- |
| 1 | a = 5 (0101₂), b = 9 (1001₂) | – | (a & b) = 0001₂ = 1. (1 & 1) = 1, so the **if** branch is taken. |
| 2 | (a << 1) ^ (b >> 1) | 14 | Calculate (5<<1)=10 (1010₂), (9>>1)=4 (0100₂), XOR: 1010₂ ^ 0100₂ = 1110₂ (14). |
| 3 | **Output**: 14 | – | result = 14. |

The code prints **14**.

### Dry Run 8: Bitwise Conditional with XOR, OR, AND

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int m = 12, n = 25;  
 int res = 0;  
 if ((m ^ n) & (m | n)) {  
 res = (~m & n) | (m & ~n);  
 } else {  
 res = (m & n) ^ (m | n);  
 }  
 cout << res;  
 return 0;  
}

| Step | Expression | Value | Explanation |
| --- | --- | --- | --- |
| 1 | m=12(01100₂), n=25(11001₂) | – | (m^n) = 10101₂ (21), (m|n)= 11101₂ (29). (21 & 29) = 10101₂ which is nonzero, so **if** branch is taken. |
| 2 | (~m & n) | (m & ~n) | 21 | ~m = ...10011₂ (if 5-bit, that is 10011₂=19), (~m & n) = 10011₂ & 11001₂ = 10001₂ (17). ~n = 00110₂ (6), (m & ~n) = 01100₂ & 00110₂ = 00100₂ (4). OR: 10001₂ | 00100₂ = 10101₂ (21). |
| 3 | **Output**: 21 | – | res = 21. |

The code prints **21**.

### Dry Run 9: Multi-Branch Bitwise Logic

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int p = 10, q = 6, r = 8;  
 int out = 0;  
 if (p & 1) {  
 out = (p & q) - (r >> 1);  
 } else if (q & 1) {  
 out = (q | r) << 1;  
 } else {  
 out = (p ^ r) + (q << 2);  
 }  
 cout << out;  
 return 0;  
}

| Step | Condition | Taken? | Value | Explanation |
| --- | --- | --- | --- | --- |
| 1 | (p & 1) | false | – | p=10 (1010₂), so (p&1)=0. **if** false. |
| 2 | (q & 1) | false | – | q=6 (0110₂), so (q&1)=0. **else if** false. |
| 3 | Else branch | – | – | Neither condition above was true, so execute **else** block. |
| 4 | (p ^ r) + (q << 2) | 26 |  | p^r = 10^8 = 2 (0010₂), q<<2 = 6<<2 = 24. Sum = 2 + 24 = 26. |
| 5 | **Output**: 26 |  |  | out = 26. |

The code prints **26**.

### Dry Run 10: Counting Set Bits in a Loop

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int s = 14;  
 int count = 0;  
 while (s) {  
 count += (s & 1);  
 s >>= 1;  
 }  
 cout << count;  
 return 0;  
}

| Step | s | (s&1) | count | Explanation |
| --- | --- | --- | --- | --- |
| 1 | 14 | 0 | 0 | **Init**: s=14(1110₂). count=0. s&1=0, so count stays 0. s>>=1 → s=7. |
| 2 | 7 | 1 | 1 | s=7(0111₂). s&1=1, so count=1. s>>=1 → s=3. |
| 3 | 3 | 1 | 2 | s=3(0011₂). s&1=1, so count=2. s>>=1 → s=1. |
| 4 | 1 | 1 | 3 | s=1(0001₂). s&1=1, so count=3. s>>=1 → s=0. |
| 5 | 0 | – | 3 | **Output**: count = 3. |

The code prints **3**.

### Dry Run 11: Bitwise Reverse via Loop

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int t = 19;  
 int sum = 0;  
 for(int i = 0; i < 5; i++){  
 if ((t >> i) & 1) sum += (1 << (4 - i));  
 }  
 cout << sum;  
 return 0;  
}

| Step | i | (t>>i)&1 | Added Value | sum | Explanation |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1<<(4-0)=16 | 16 | t=19(10011₂). (t>>0)&1 = 1, so add 16. sum=16. |
| 2 | 1 | 1 | 1<<(4-1)=8 | 24 | (t>>1)&1 = 1, so add 8. sum=24. |
| 3 | 2 | 0 | 0 | 24 | (t>>2)&1 = 0, add 0. sum=24. |
| 4 | 3 | 0 | 0 | 24 | (t>>3)&1 = 0, add 0. sum=24. |
| 5 | 4 | 1 | 1<<(4-4)=1 | 25 | (t>>4)&1 = 1, so add 1. sum=25. |
| 6 | – | – | – | 25 | **Output**: sum = 25. |

The code prints **25**.

### Dry Run 12: Loop with Multiplication and Bitwise Test

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int val = 7;  
 int result = 1;  
 for(int i = 0; i < 3; i++){  
 if (val & 1) {  
 result \*= (val << i);  
 } else {  
 result \*= (val >> i);  
 }  
 }  
 cout << result;  
 return 0;  
}

| Step | i | (val&1) | Value Multiplied Into result | result | Explanation |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | (7<<0)=7 | 7 | val=7(0111₂), (val&1)=1, multiply result by 7. result=1\*7=7. |
| 2 | 1 | 1 | (7<<1)=14 | 98 | val still 7, multiply by 14. result=7\*14=98. |
| 3 | 2 | 1 | (7<<2)=28 | 2744 | val still 7, multiply by 28. result=98\*28=2744. |
| 4 | – | – | – | 2744 | **Output**: result = 2744. |

The code prints **2744**.

### Dry Run 13: Bitwise Comparison

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int a = 4, b = 9;  
 int ans = 0;  
 if ((a ^ b) < (a & b))  
 ans = (a | b) - (a & b);  
 else  
 ans = (a & b) + (a | b);  
 cout << ans;  
 return 0;  
}

| Step | Expressions | Value | Explanation |
| --- | --- | --- | --- |
| 1 | a=4(0100₂), b=9(1001₂) | – | Compute (a ^ b)=1101₂=13, (a & b)=0000₂=0. Check 13 < 0 (false). |
| 2 | (a & b) + (a | b) | 13 | Use else-branch: (a&b)=0, (a|b)=1101₂=13. Sum = 0+13 = 13. |
| 3 | **Output**: 13 | – | ans = 13. |

The code prints **13**.

## Medium Bitwise Dry Runs

These examples are more straightforward and cover basic bitwise usage.

### Dry Run 14: Check Odd or Even with Bitwise

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int n = 15;  
 cout << (n & 1);  
 return 0;  
}

| Step | n | (n&1) | Explanation |
| --- | --- | --- | --- |
| 1 | 15 | 1 | n=15(1111₂). (n&1)=1, so output 1. |
| 2 | – | – | **Output**: 1. |

The code prints **1** (odd).

### Dry Run 15: Simple Shifts and OR

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int a = 1, b = 4;  
 int x = a << 2; // shift a left by 2  
 int y = b >> 1; // shift b right by 1  
 cout << (x | y);  
 return 0;  
}

| Step | Expression | Value | Explanation |
| --- | --- | --- | --- |
| 1 | a<<2 = 1<<2 | 4 | a=1. 1<<2=4. |
| 2 | b>>1 = 4>>1 | 2 | b=4. 4>>1=2. |
| 3 | (x | y) = 4 | 2 |
| 4 | **Output**: 6 | – |  |

The code prints **6**.

### Dry Run 16: Bitwise AND

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int x = 3, y = 5;  
 cout << (x & y);  
 return 0;  
}

| x | y | (x&y) | Explanation |
| --- | --- | --- | --- |
| 3 (011₂) | 5 (101₂) | 1 (001₂) | 3&5 = 1. Output 1. |

The code prints **1**.

### Dry Run 17: Bitwise XOR

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int p = 12, q = 5;  
 cout << (p ^ q);  
 return 0;  
}

| p | q | (p^q) | Explanation |
| --- | --- | --- | --- |
| 12 (1100₂) | 5 (0101₂) | 9 (1001₂) | 12^5 = 9. Output 9. |

The code prints **9**.

### Dry Run 18: Compound AND Assignment

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int d = 6;  
 d &= 3;  
 cout << d;  
 return 0;  
}

| Step | Expression | Value | Explanation |
| --- | --- | --- | --- |
| 1 | d = 6 (110₂) | – | Initialize d = 6. |
| 2 | d &= 3 | 2 | 6&3 = 2. Output 2. |

The code prints **2**.

### Dry Run 19: Left Shift Assignment

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int c = 3;  
 c <<= 4;  
 cout << c;  
 return 0;  
}

| Step | c (before) | c (after) | Explanation |
| --- | --- | --- | --- |
| 1 | 3 (11₂) | – | Initialize c = 3. |
| 2 | – | 48 | Shift left by 4: 3<<4=48. Output 48. |

The code prints **48**.

### Dry Run 20: Bitwise OR Assignment

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int v = 7;  
 v |= 3;  
 cout << v;  
 return 0;  
}

| Step | v (before) | Expression | v (after) | Explanation |
| --- | --- | --- | --- | --- |
| 1 | 7 (111₂) | – | – | Initialize v = 7. |
| 2 | 7 | v | = 3 | 7 (111₂) |

The code prints **7**.

### Dry Run 21: Shift and OR Combination

#include <iostream>  
using namespace std;  
  
int main() {  
 unsigned int m = 9;  
 unsigned int result = (m >> 1) | (m << 2);  
 cout << result;  
 return 0;  
}

| m | m>>1 | m<<2 | (m>>1)|(m<<2) | Explanation | |:---:|:----:|:----:|:--------:|:----------------------------------------| | 9 (1001₂) | 4 (0100₂) | 36 (100100₂) | 44 (101100₂) | (4|36)=44. Output 44. |

The code prints **44**.

Each table above walks through the code execution step by step. The **Output** row (or explanation) shows the final printed value for that dry-run example.