Optimalizácia

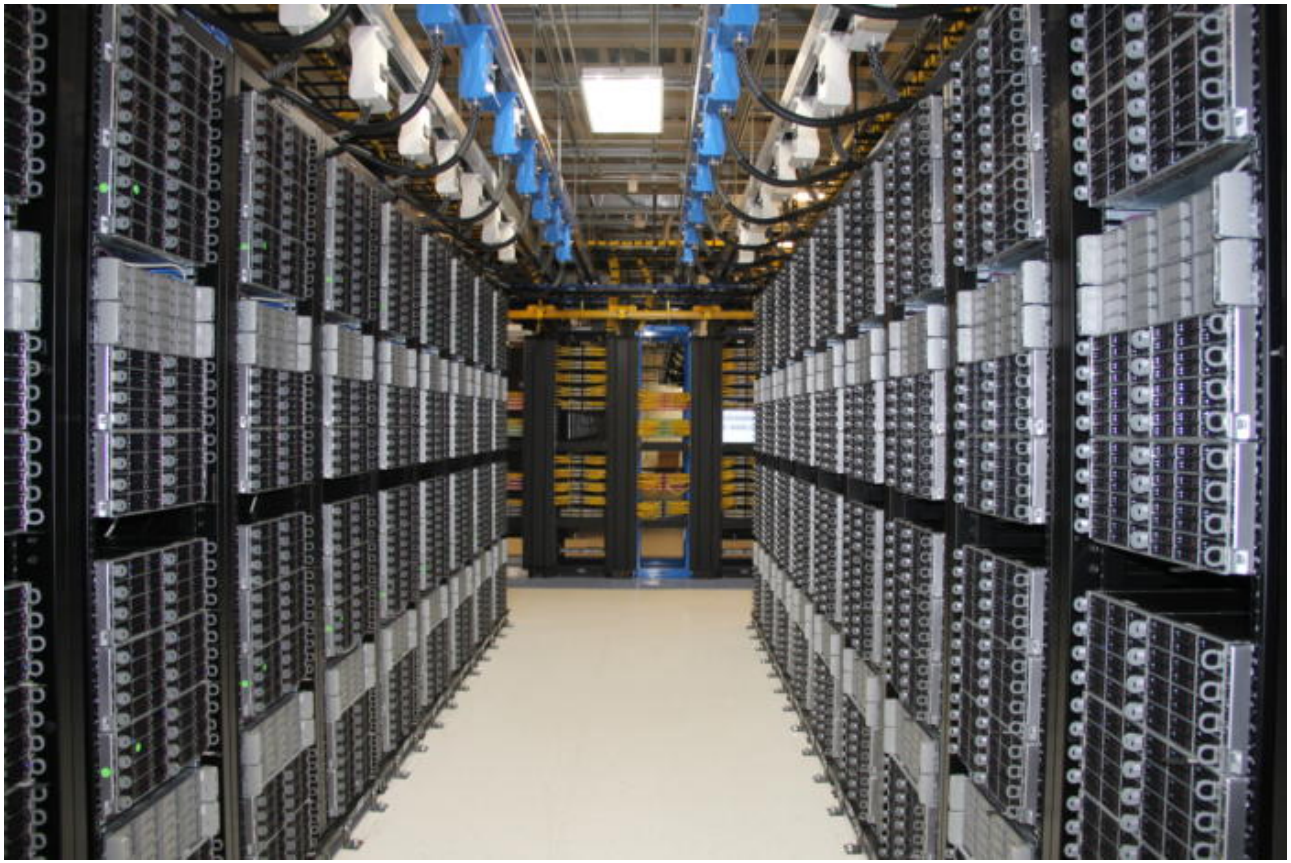
kuko

27.10.2020

Vybrané partie z dátových štruktúr

Načo optimalizovať?


Software is getting
slower more rapidly than
hardware becomes faster.
Niklaus Wirth
quotefancy



Tip #1: NEoptimalizujte

Knuth: Premature optimization is the root of all evil

- korektnosť
- udržiavateľnosť
- efektívnosť

When do I optimize my code?

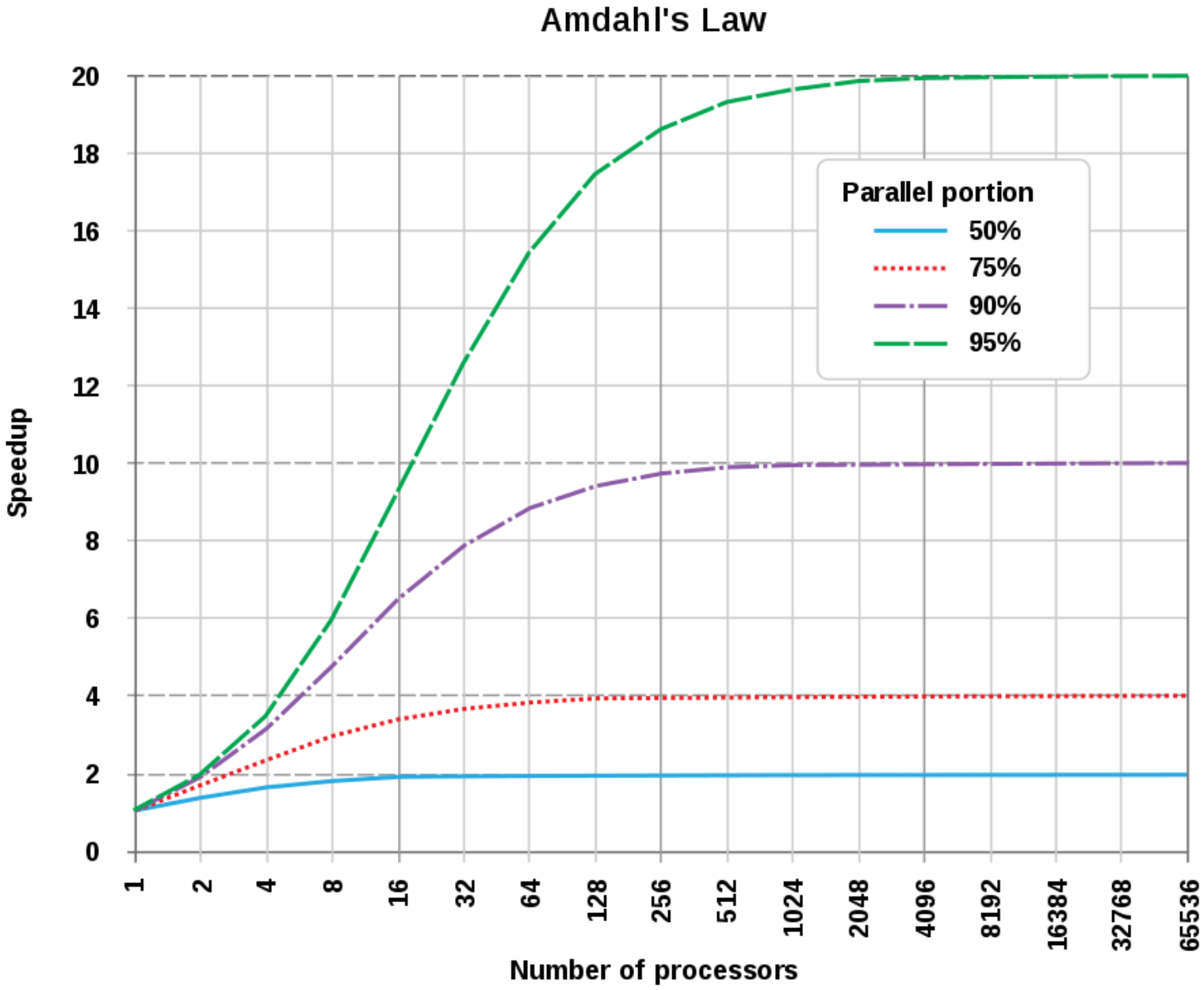
1. ***Don't***
2. ***Don't yet***
3. ***Profile before optimizing***

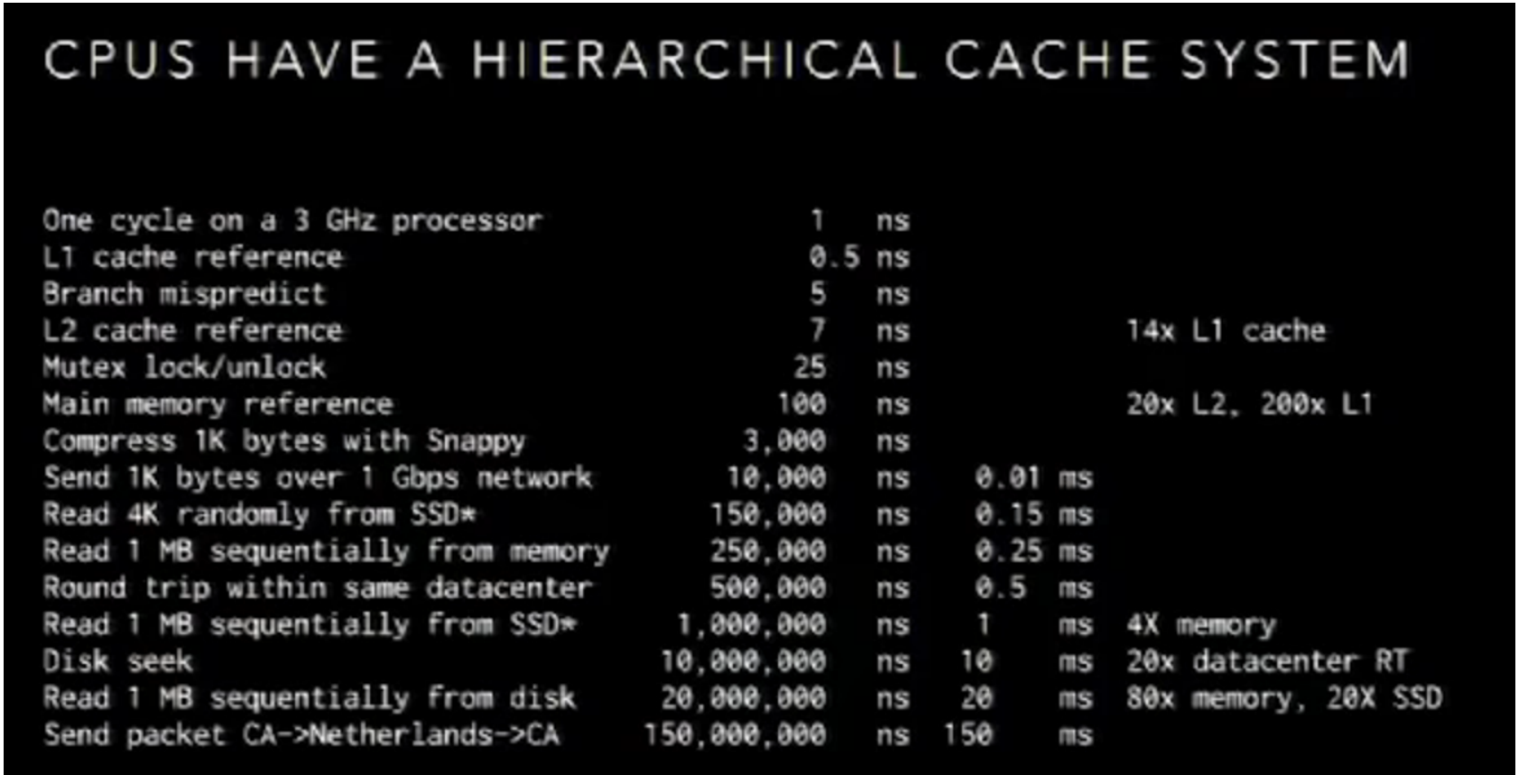
Two independent parts A B

Original process 

Make B 5x faster 

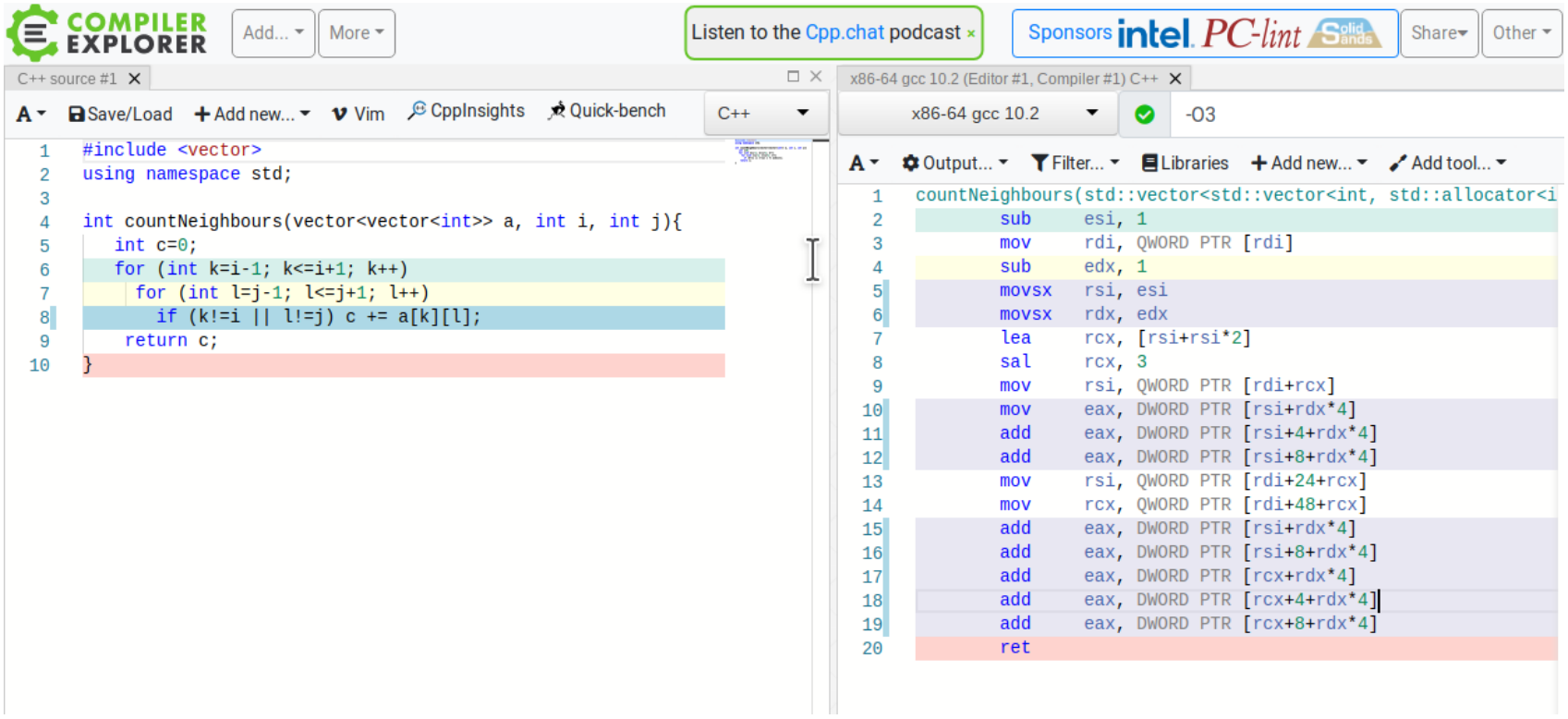
Make A 2x faster 


Amdahl's Law
Speedup
Number of processors
Parallel portion
50%
75%
90%
95%
0
2
4
6
8
10
12
14
16
18
20
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536

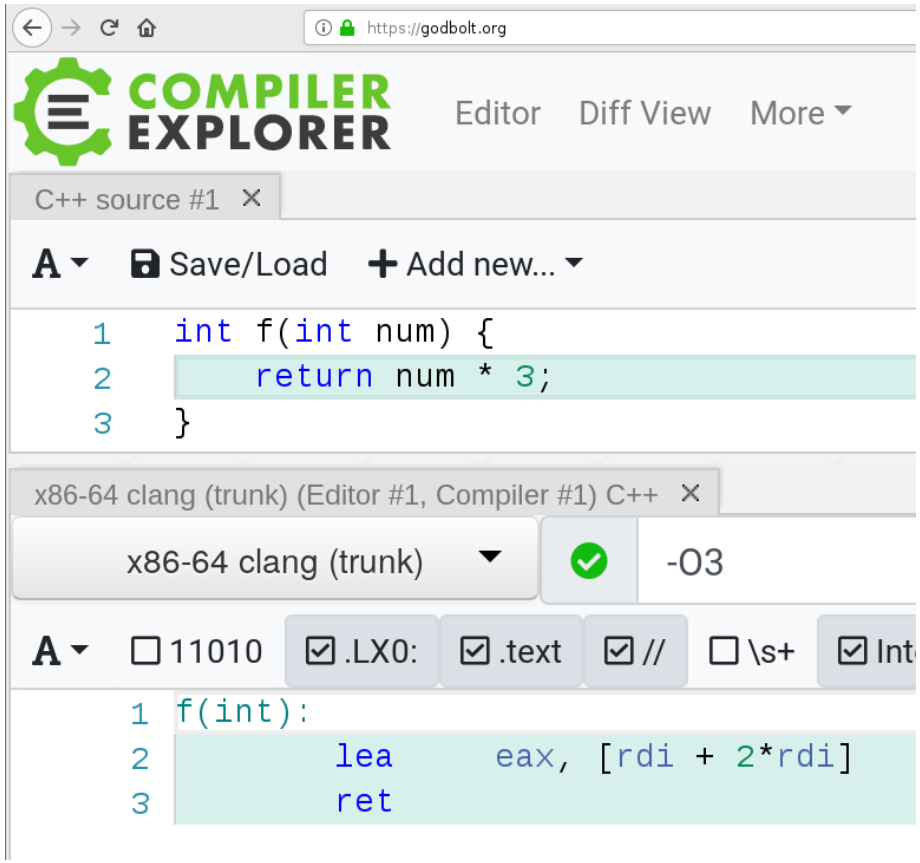


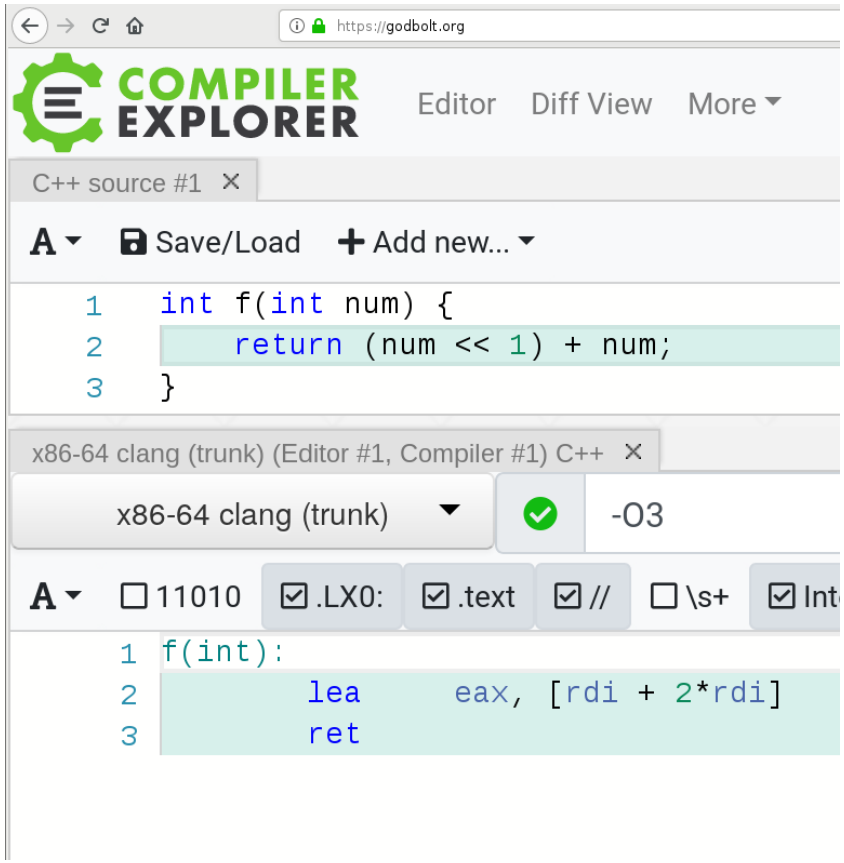
https://gist.github.com/hellerbarde/2843375

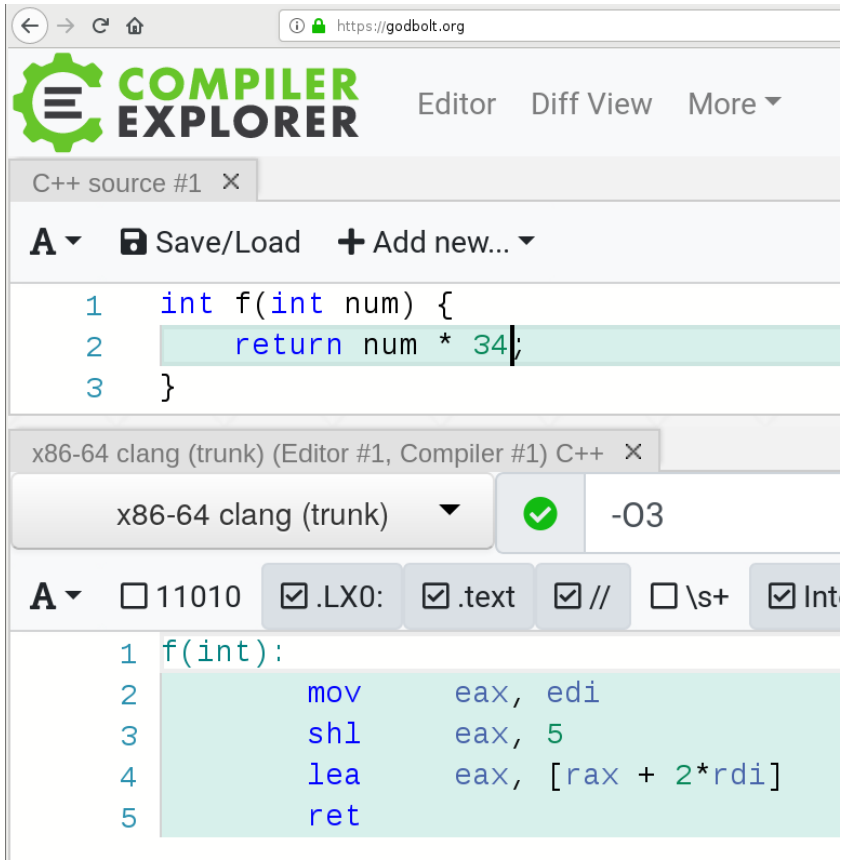
Tip #1': Nerobte prácu kompilátora



https://godbolt.org/z/fsnhb9


https://godbolt.org
COMPILER EXPLORER
Editor
Diff View
More
C++ source #1
Save/Load
Add new...
1 int f(int num) {
2 return num * 3;
3 }
x86-64 clang (trunk) (Editor #1, Compiler #1) C++
x86-64 clang (trunk)
-O3
11010
.LX0:
.text
//
\s+
1 f(int):
2 lea eax, [rdi + 2*rdi]
3 ret


https://godbolt.org
COMPILER EXPLORER
Editor
Diff View
More
C++ source #1
Save/Load
Add new...
1 int f(int num) {
2 return (num << 1) + num;
3 }
x86-64 clang (trunk) (Editor #1, Compiler #1) C++
x86-64 clang (trunk)
-O3
11010
.LX0:
.text
//
\s+
1 f(int):
2 lea eax, [rdi + 2*rdi]
3 ret


https://godbolt.org
COMPILER EXPLORER
Editor
Diff View
More
C++ source #1
Save/Load
Add new...
1 int f(int num) {
2 return num * 34;
3 }
x86-64 clang (trunk) (Editor #1, Compiler #1) C++
x86-64 clang (trunk)
-O3
11010
.LX0:
.text
//
\s+
1 f(int):
2 mov eax, edi
3 shl eax, 5
4 lea eax, [rax + 2*rdi]
5 ret

Ako optimalizovať

1. časť: Menej práce

Tip #2: Zvoľte správny algoritmus a DŠ

- asymptotická zložitosť
  - ignoruje konštanty
  - správanie pre veľké n
  - horný odhad

- triedenie
- vyhľadávanie
- najbližší menší
- vyhľadávanie v texte
- vyhľadávanie regulárnych výrazov

Vyhľadávanie v texte

TABLE 3 *An example of BM algorithm matching process, 4 shifts, 14 comparisons*

	J	U	L	I	E	T	T	H	O	T	E	L	T	A	N	G	O	F	O	X	T	R	O	T
1	F	O	X	T	R	O	T																	
2				F	O	X	T	R	O	T														
3											F	O	X	T	R	O	T							
4												F	O	X	T	R	O	T						
5																		F	O	X	T	R	O	T

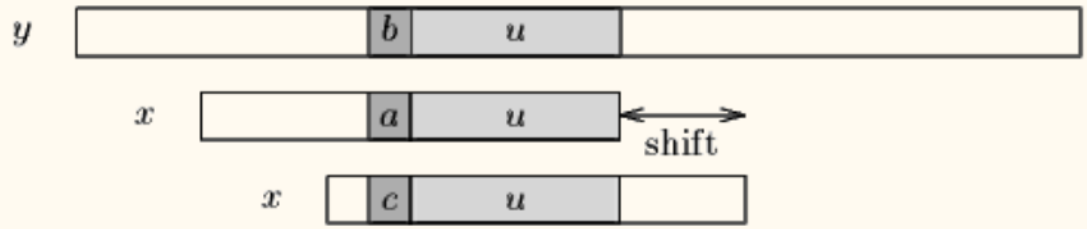


Figure 13.1. The good-suffix shift, u re-occurs preceded by a character c different from a.

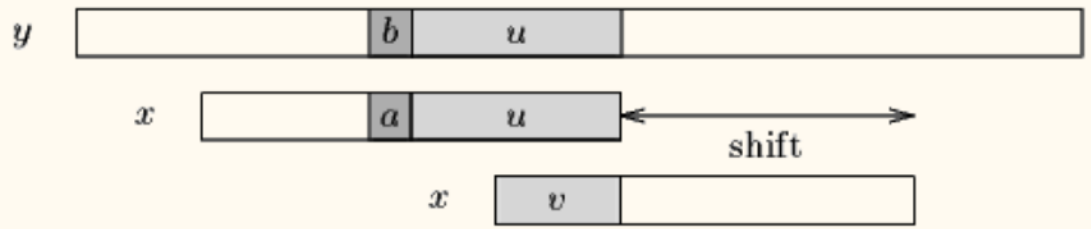


Figure 13.2. The good-suffix shift, only a suffix of u re-occurs in x.

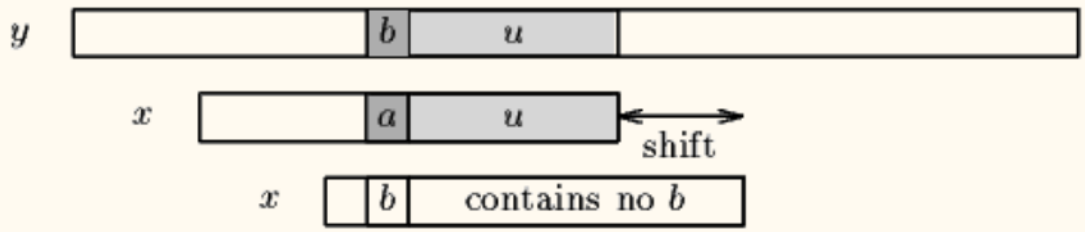


Figure 13.3. The bad-character shift, a occurs in x.

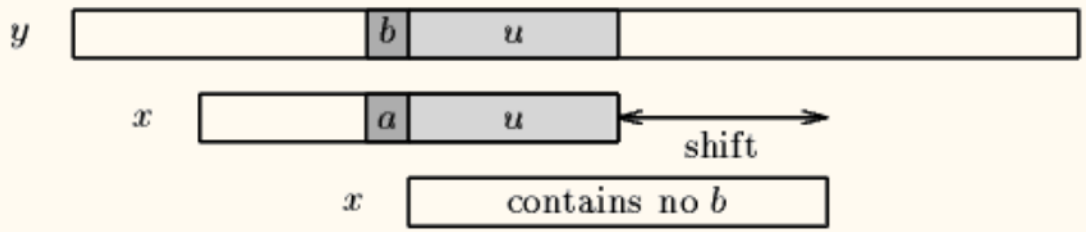
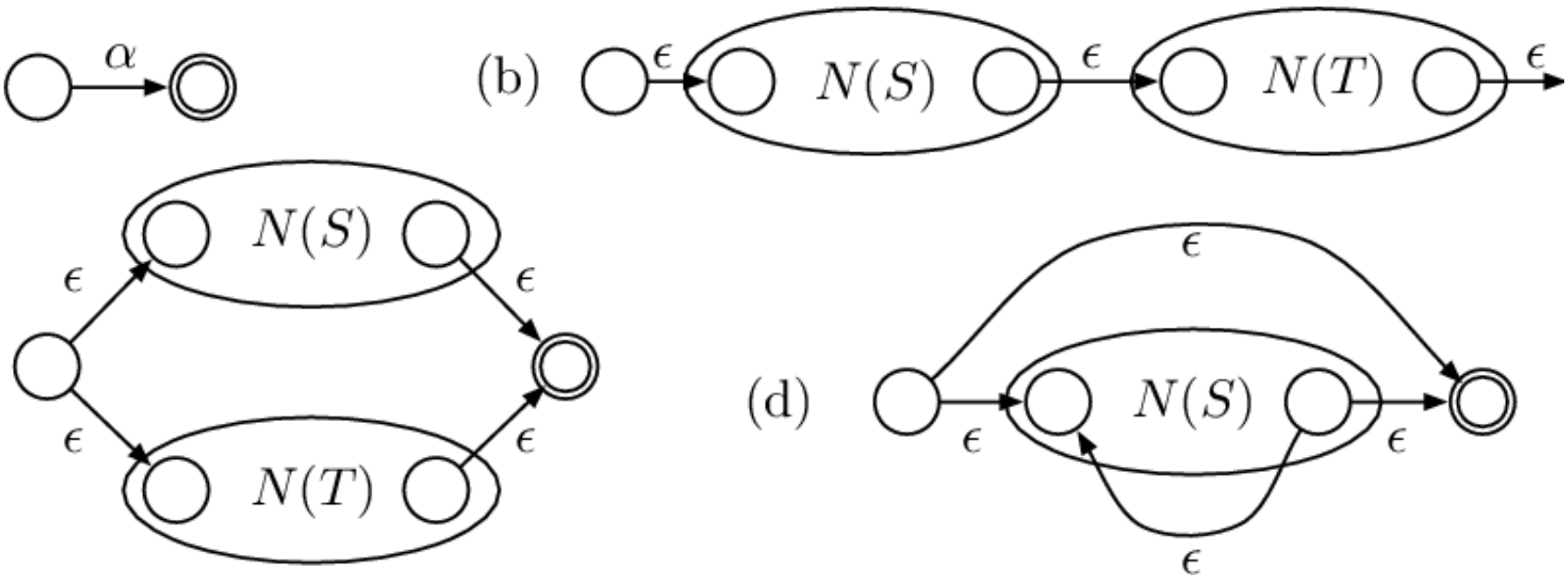


Figure 13.4. The bad-character shift, b does not occur in x.

Vyhľadávanie regulárnych výrazov


α
(b)
ϵ
N(S)
ϵ
N(T)
ϵ
N(S)
ϵ
ϵ
ϵ
ϵ
N(T)
(d)
ϵ
ϵ
N(S)
ϵ
ϵ

text: aaaaa
pattern: a?a?a?a?a?aaaaa

https://swtch.com/~rsc/regexp/regexp1.html

```
import re
n = 20
t = n*"a"
p = n*"a?" + t
print "text: ", t
print "pattern: ", p
re.match(p, t).group(0)
```

```
echo "grep -G:"
grep -G "a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?aaaaaaaaaaaaaaaaaaaaaaaaaaaaa" text
echo $?
echo "grep -P:"
grep -P "a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?aaaaaaaaaaaaaaaaaaaaaaaaaaaaa" text
echo $?
echo "grep -E:"
grep -E "a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?a?aaaaaaaaaaaaaaaaaaaaaaaaaaaaa" text
echo $?
```

Hešovanie

```
string r = "";
FOREACH(it, vec) r += *it;
```

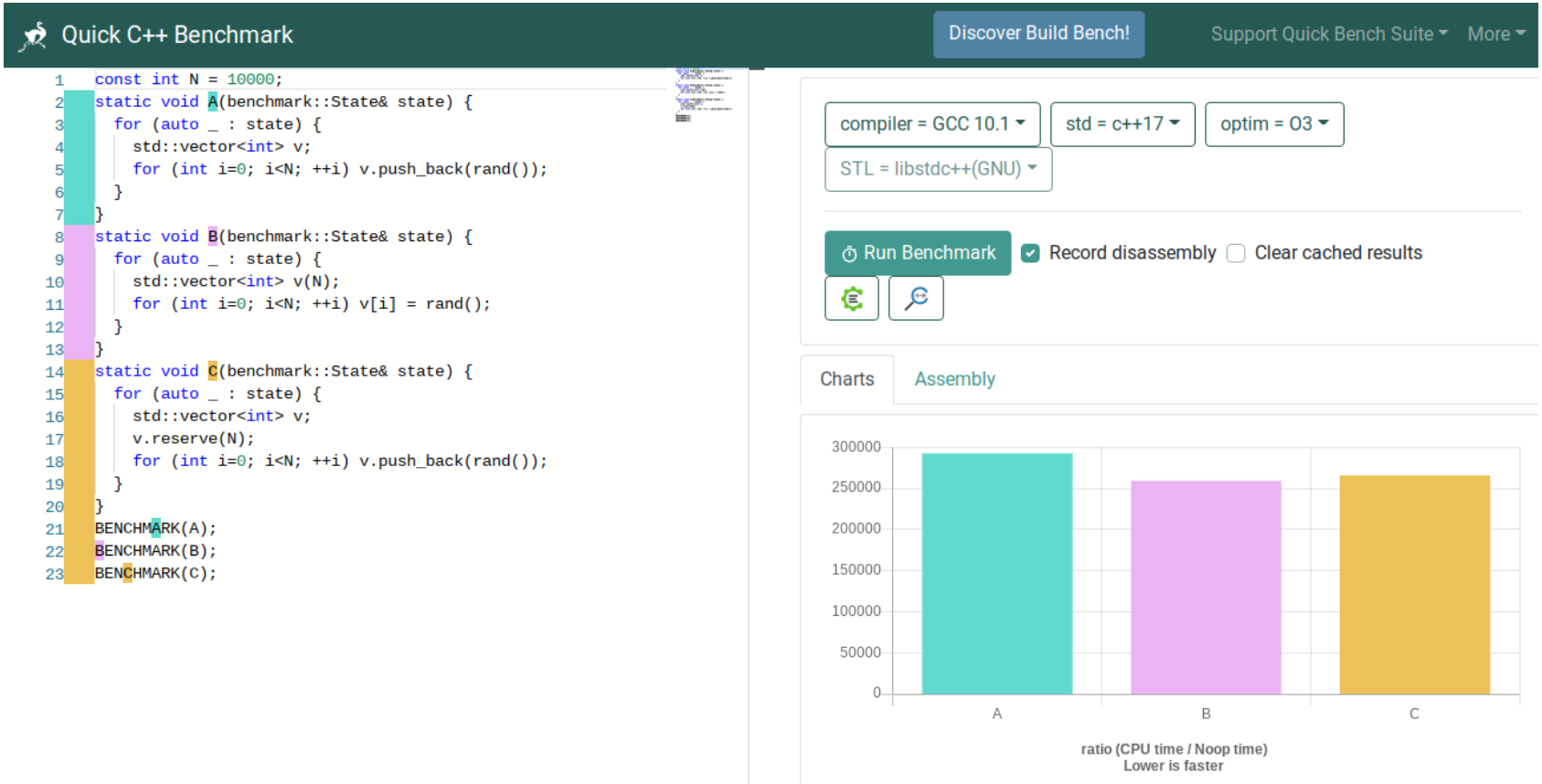
```
string r = boost::algorithm::join(vec, "")
```

Tip #3: Robte menej

```
std::vector<int> v;
for (int i=0; i<N; ++i)
  v.push_back(rand());
```

```cpp
std::vector<int> v;
v.reserve(N);
for (int i=0; i<N; ++i)
  v.push_back(rand());
```

```
std::vector<int> v(N);
for (int i=0; i<N; ++i)
  v[i] = rand();
```




```
1  const int N = 10000;
2  static void A(benchmark::State& state) {
3    for (auto _ : state) {
4      std::vector<int> v;
5      for (int i=0; i<N; ++i) v.push_back(rand());
6    }
7  }
8  static void B(benchmark::State& state) {
9    for (auto _ : state) {
10     std::vector<int> v(N);
11     for (int i=0; i<N; ++i) v[i] = rand();
12   }
13 }
14 static void C(benchmark::State& state) {
15   for (auto _ : state) {
16     std::vector<int> v;
17     v.reserve(N);
18     for (int i=0; i<N; ++i) v.push_back(rand());
19   }
20 }
21 BENCHMARK(A);
22 BENCHMARK(B);
23 BENCHMARK(C);
```

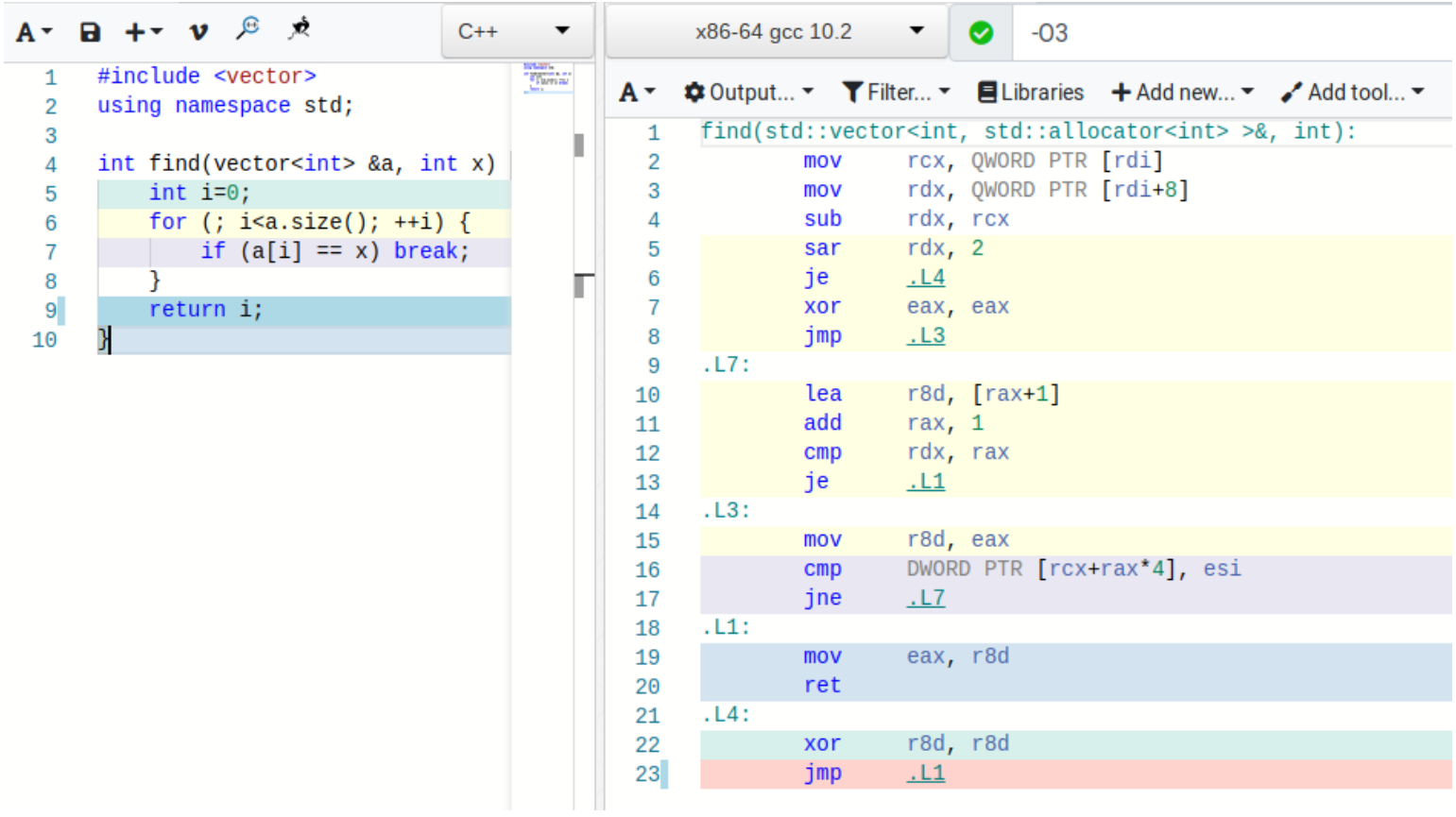
https://quick-bench.com/q/wKD9W3fhG4IE0wUQDA64rbds4jc

```cpp
#include<iostream>
using namespace std;

int main(int argc, char** argv) {
  for (int i=0; i<1000000; ++i)
    cout << "Hello World!" << endl;
  return 0;
}
```

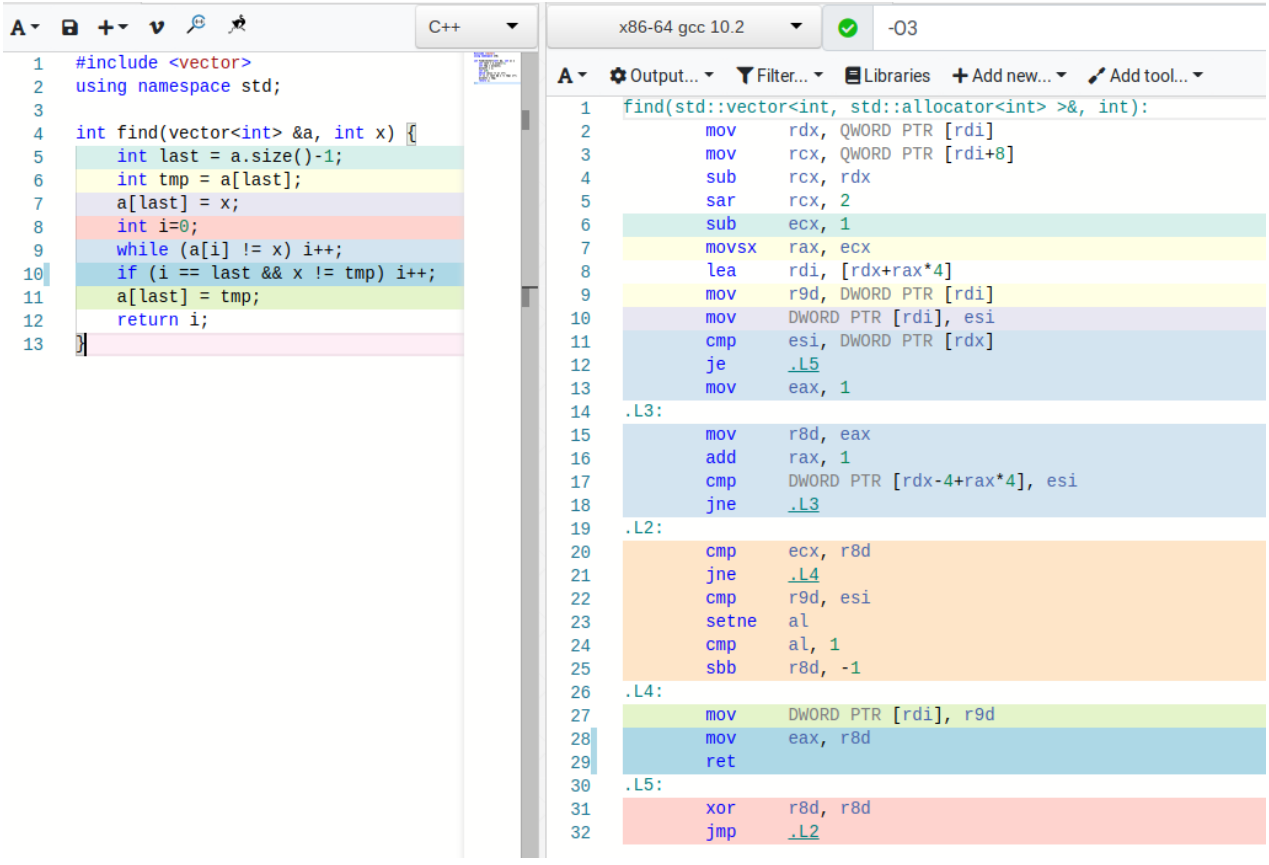
```cpp
#include<iostream>
using namespace std;

int main(int argc, char** argv) {
  for (int i=0; i<1000000; ++i)
    cout << "Hello World!\n" << flush;
  return 0;
}
```

```cpp
#include<iostream>
using namespace std;

int main(int argc, char** argv) {
  for (int i=0; i<1000000; ++i)
    cout << "Hello World!\n";
  return 0;
}
```

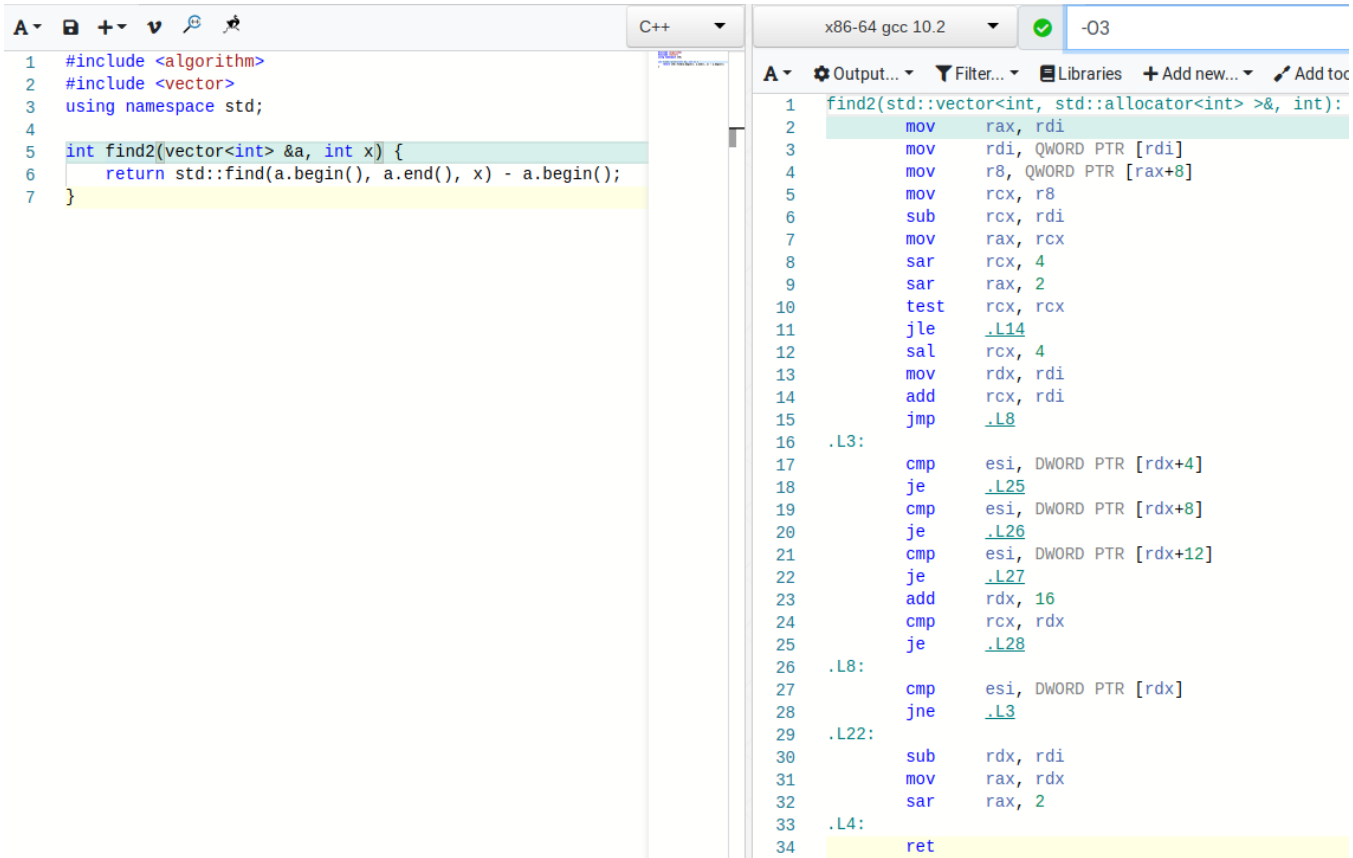
- lineárny find



https://godbolt.org/z/Y9o9bW



https://godbolt.org/z/rbh6d8



```
C++

#include <algorithm>
#include <vector>
using namespace std;

int find2(vector<int> &a, int x) {
    return std::find(a.begin(), a.end(), x) - a.begin();
}
```

```
x86-64 gcc 10.2    -O3

Output...  Filter...  Libraries  + Add new...  Add toc

find2(std::vector<int, std::allocator<int> >&, int):
        mov     rax, rdi
        mov     rdi, QWORD PTR [rdi]
        mov     r8, QWORD PTR [rax+8]
        mov     rcx, r8
        sub     rcx, rdi
        mov     rax, rcx
        sar     rcx, 4
        sar     rax, 2
        test    rcx, rcx
        jle     .L14
        sal     rcx, 4
        mov     rdx, rdi
        add     rcx, rdi
        jmp     .L8
.L3:
        cmp     esi, DWORD PTR [rdx+4]
        je      .L25
        cmp     esi, DWORD PTR [rdx+8]
        je      .L26
        cmp     esi, DWORD PTR [rdx+12]
        je      .L27
        add     rdx, 16
        cmp     rcx, rdx
        je      .L28
.L8:
        cmp     esi, DWORD PTR [rdx]
        jne     .L3
.L22:
        sub     rdx, rdi
        mov     rax, rdx
        sar     rax, 2
.L4:
        ret
```

Tip #4: Používajte zarážky

Tip #5: Zrýchlite častý/priemerný prípad

Príklad 1: UTF8

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

```
    while (s[i]) {
        if ((s[i] & 0xc0) != 0x80) j++;
        i++;
    }
    return j;
```

https://godbolt.org/z/8vbssd

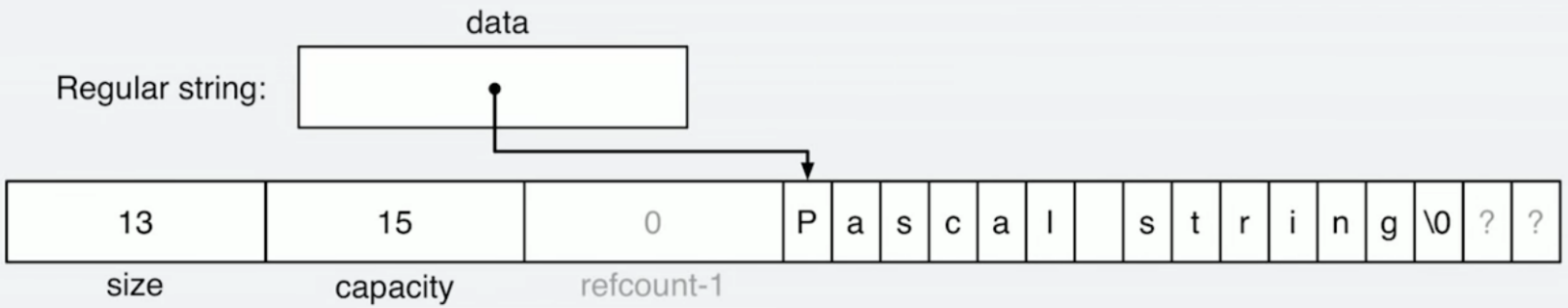
```
        while (s[i] > 0) {
ascii:
            i++;
        }

        count += i - iBefore;

        while (s[i]) {
            if (s[i] > 0) {
                iBefore = i;
                goto ascii;
            } else {
                switch (0xF0 & s[i]) {
                case 0xE0:
                    i += 3;
                    break;
                case 0xF0:
                    i += 4;
                    break;
                default:
                    i += 2;
                    break;
                }
            }
            count++;
        }
```

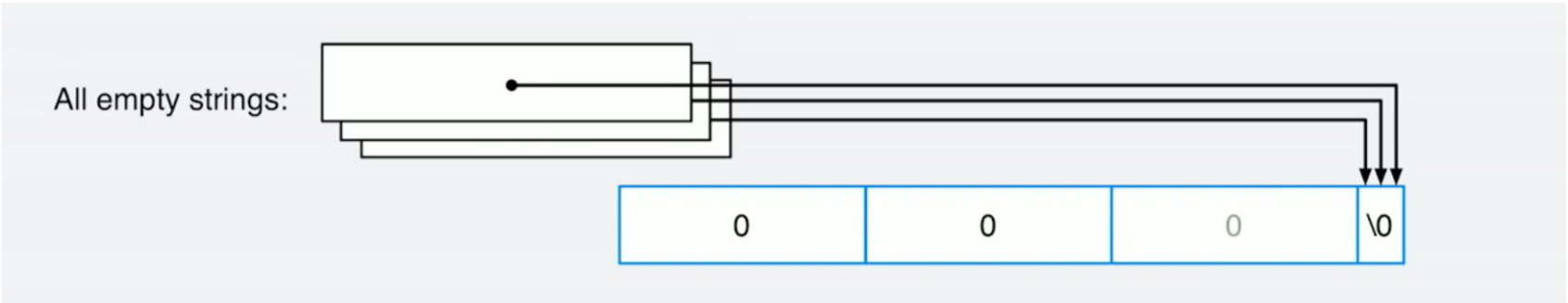
https://godbolt.org/z/1dhKq5

Príklad 2: Ako funguje string?

```
struct string {
    int size;
    int capacity;
    char * data;
};
```

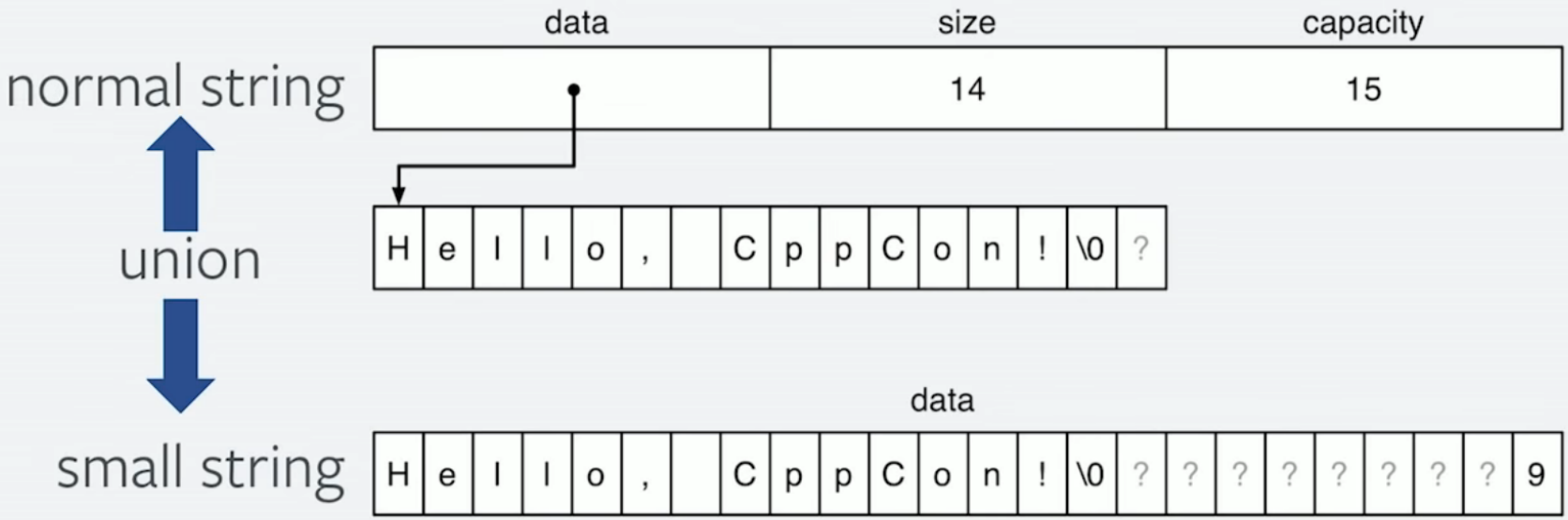
gcc string (version <5)



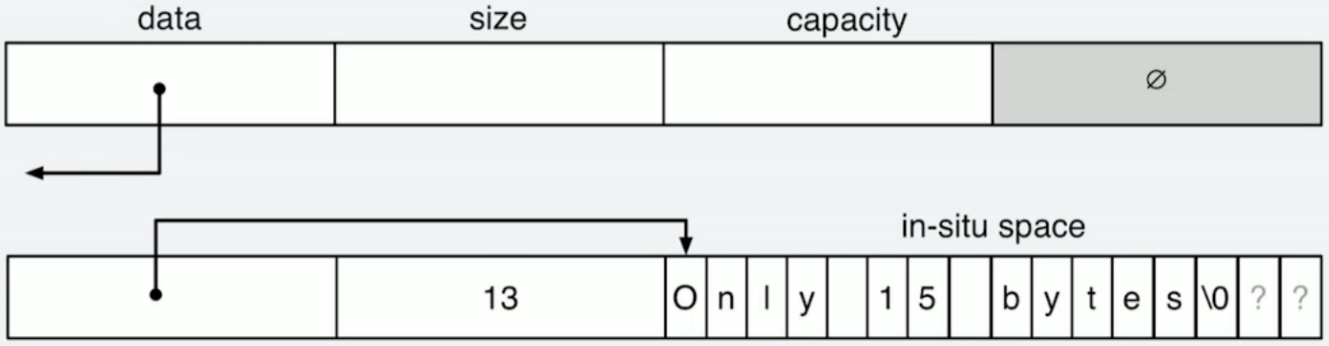

All empty strings:
0
0
0
\0

fbstring

@author Andrei Alexandrescu



gcc string (version >=5)



+Has SSO

+`data()`, `size()` very fast

+Size, 32, is power of 2

−Only 15-byte capacity

−Move is no longer memcopy

−Size is 33% larger than fbstring

1. NEoptimalizujte, nerobte prácu kompilátora
2. Zvoľte správny algoritmus a DŠ
3. Robte menej
4. Používajte zarážky
5. Zrýchlite častý/priemerný prípad

2. časť: Rovnaká práca rýchlejšie

CPUS HAVE A HIERARCHICAL CACHE SYSTEM

One cycle on a 3 GHz processor	1	ns			
L1 cache reference	0.5	ns			
Branch mispredict	5	ns			
L2 cache reference	7	ns			14x L1 cache
Mutex lock/unlock	25	ns			
Main memory reference	100	ns			20x L2, 200x L1
Compress 1K bytes with Snappy	3,000	ns			
Send 1K bytes over 1 Gbps network	10,000	ns	0.01	ms	
Read 4K randomly from SSD*	150,000	ns	0.15	ms	
Read 1 MB sequentially from memory	250,000	ns	0.25	ms	
Round trip within same datacenter	500,000	ns	0.5	ms	
Read 1 MB sequentially from SSD*	1,000,000	ns	1	ms	4X memory
Disk seek	10,000,000	ns	10	ms	20x datacenter RT
Read 1 MB sequentially from disk	20,000,000	ns	20	ms	80x memory, 20X SSD
Send packet CA->Netherlands->CA	150,000,000	ns	150	ms	

Tip #6: Využívajte efektívne cache

- prechod 2D poľa po riadkoch vs. po stĺpcoch vs. náhodne
- vector vs. list
- qsort vs. heapsort
- d-árna vs. binárna halda
- flat_hash_set vs. unordered_set
- podielové vs. bloom filtre

Tip #7: Bitová a vektorová mágia

Subsets of $0, \ldots, N-1$

a subset	{ 0,			3,		5 }		
good/bad numbers		0,	1,	2,	3,	4,	5	
yes/no bits		1	0	0	1	0	1	<- binary!
the number	2^0+			2^3+		2^5		= 41

Bitwise operations

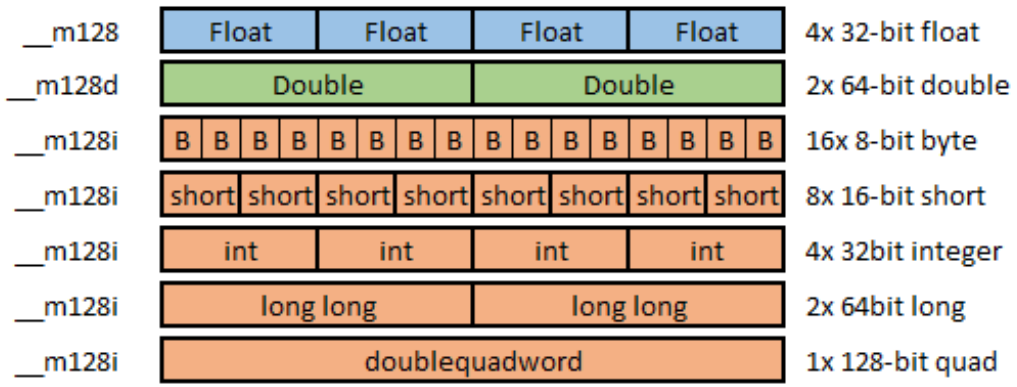
union: bitwise or

intersection: bitwise and

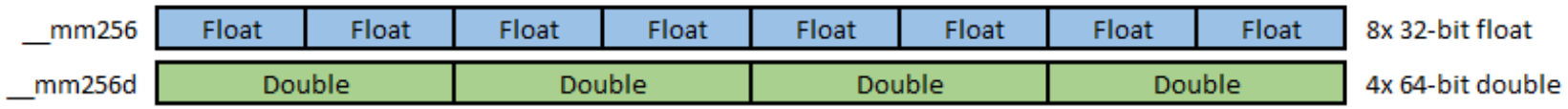
invert mask: bitwise xor

set $\{i\}$: bitwise shifts: `1 << i`

SSE Data Types (16 XMM Registers)

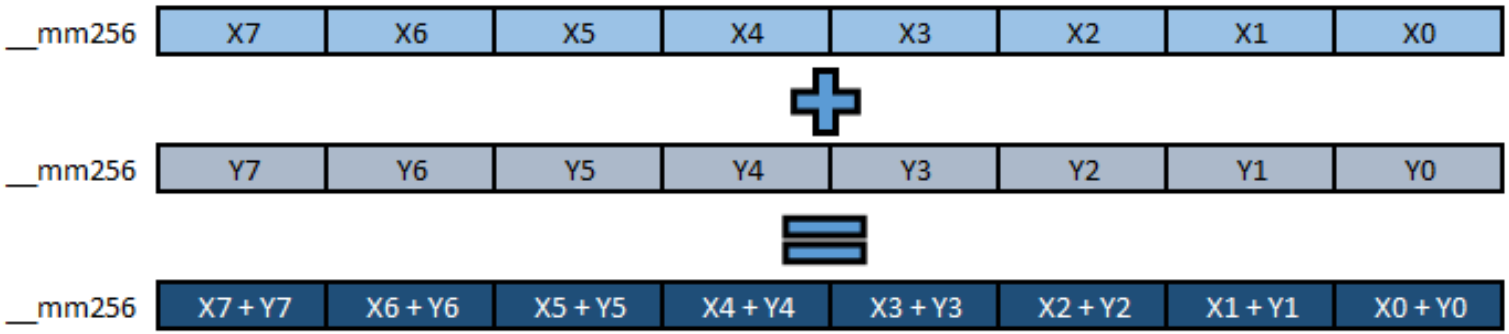


AVX Data Types (16 YMM Registers)



__mm256i *256-bit Integer registers. It behaves similarly to __m128i.Out of scope in AVX, useful on AVX2*

AVX Operation



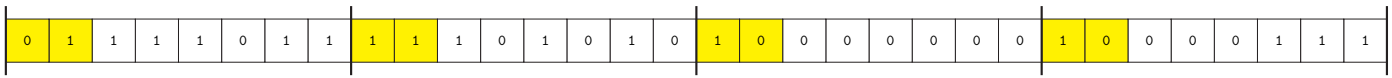
Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

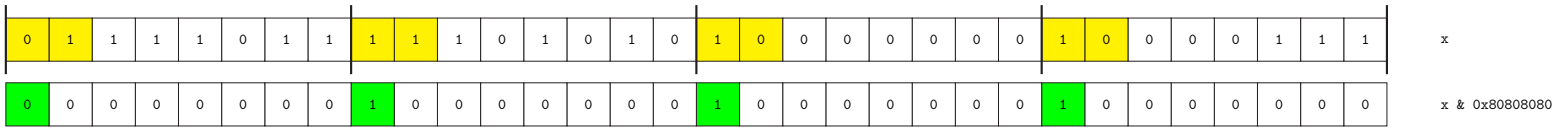
```
    while (s[i]) {
        if ((s[i] & 0xc0) != 0x80) j++;
        i++;
    }
    return j;
```

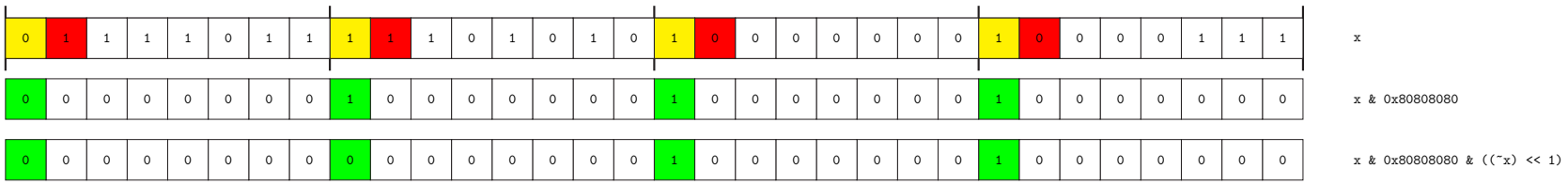
```
/* Handle complete blocks. */
for (; ; s += sizeof(size_t)) {
        /* Grab 4 or 8 bytes of UTF-8 data. */
        u = *(size_t *)(s);

        /* Exit the loop if there are any zero bytes. */
        if ((u - ONEMASK) & (~u) & HIMASK) break;

        /* Count bytes which are NOT the first byte of a character. */
        count += __builtin_popcountll(u & HIMASK & ((~u) << 1));
}
```

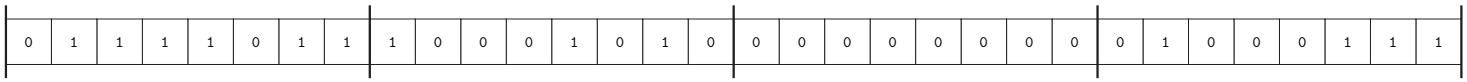
https://godbolt.org/z/Pn7xTb

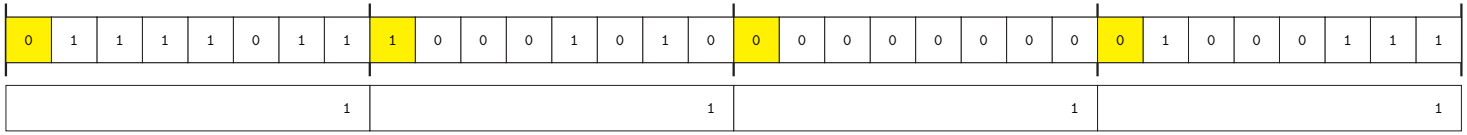
http://www.daemonology.net/blog/2008-06-05-faster-utf8-strlen.html

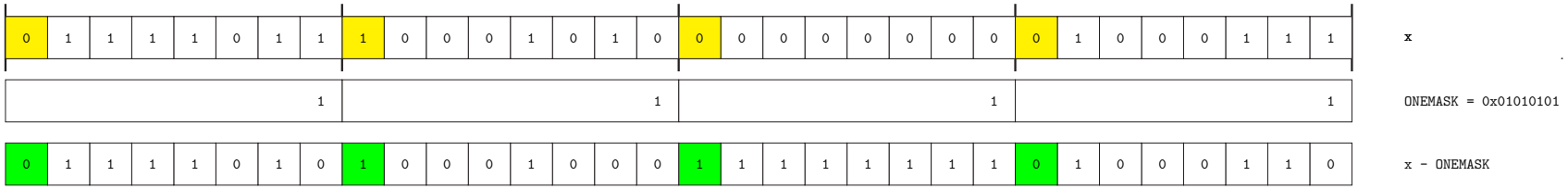

0 1 1 1 1 0 1 1
1 1 1 0 1 0 1 0
1 0 0 0 0 0 0 0
1 0 0 0 0 1 1 1
x

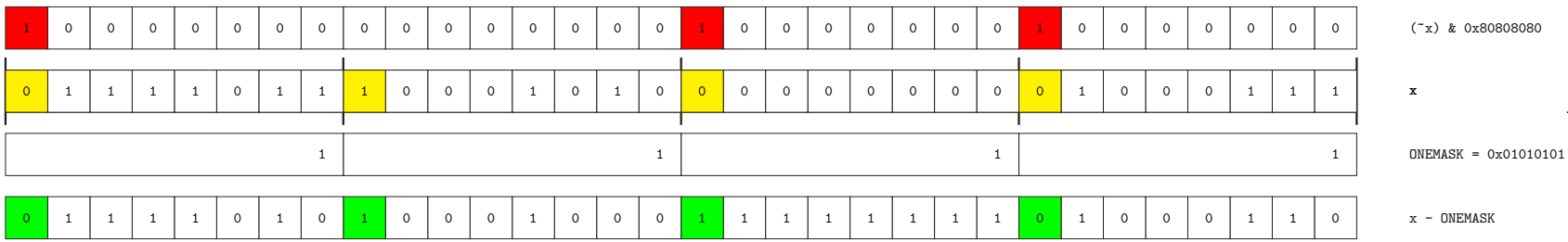

0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1
x
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
x & 0x80808080


0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1
x
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
x & 0x80808080
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
x & 0x80808080 & ((~x) << 1)


0 1 1 1 1 0 1 1
1 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0
0 1 0 0 0 1 1 1
x


0 1 1 1 1 0 1 1
1 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0
0 1 0 0 0 1 1 1
x
1
1
1
1
ONEMASK = 0x01010101


0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1
x
1 1 1 1
ONEMASK = 0x01010101
0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0
x - ONEMASK


1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
(~x) & 0x80808080
0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1
x
1 1 1 1
ONEMASK = 0x01010101
0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 0
x - ONEMASK

```
/* Handle complete blocks. */
for (; ; s += sizeof(size_t)) {
        /* Prefetch 256 bytes ahead. */
        __builtin_prefetch(&s[256], 0, 0);

        /* Grab 4 or 8 bytes of UTF-8 data. */
        u = *(size_t *)(s);

        /* Exit the loop if there are any zero bytes. */
        if ((u - ONEMASK) & (~u) & HIMASK) break;

        /* Count bytes which are NOT the first byte of a character. */
        count += __builtin_popcountll(u & HIMASK & ((~u) << 1));
}
```

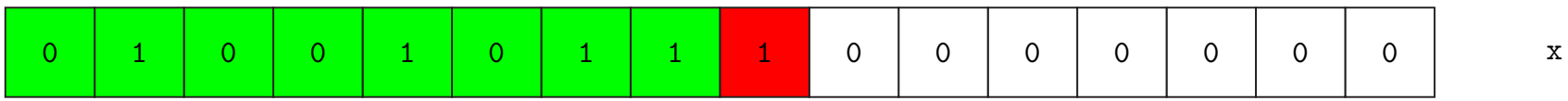
https://godbolt.org/z/Pn7xTb

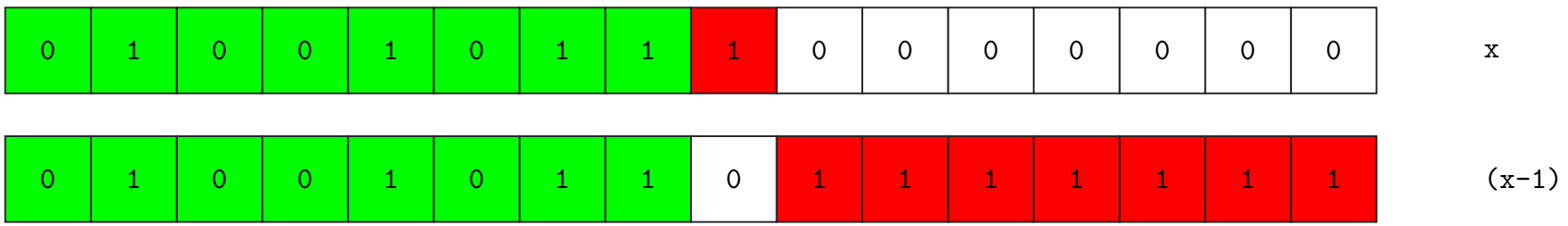
```
constexpr size_t ONEMASK = ((size_t)(-1) / 0xFF); // 0x0101010101010101
constexpr size_t HIMASK = ONEMASK * 0x80;         // 0x8080808080808080

static size_t cp_strlen_utf8(const char * _s) {
        const char * s;
        size_t count = 0;
        size_t u;
        unsigned char b;

        /* Handle any initial misaligned bytes. */
        for (s = _s; (uintptr_t)(s) & (sizeof(size_t) - 1); s++) {
                b = *s;

                /* Exit if we hit a zero byte. */
                if (b == '\0') goto done;

                /* Is this byte NOT the first byte of a character? */
                count += (b >> 7) & ((~b) >> 6);
        }

        /* Handle complete blocks. */
        for (; ; s += sizeof(size_t)) {
                /* Prefetch 256 bytes ahead. */
                __builtin_prefetch(&s[256], 0, 0);

                /* Grab 4 or 8 bytes of UTF-8 data. */
                u = *(size_t *)(s);

                /* Exit the loop if there are any zero bytes. */
                if ((u - ONEMASK) & (~u) & HIMASK) break;

                /* Count bytes which are NOT the first byte of a character. */
                count += __builtin_popcountll(u & HIMASK & ((~u) << 1));
        }

        /* Take care of any left-over bytes. */
        for (; ; s++) {
                b = *s;
                if (b == '\0') break;
                count += (b >> 7) & ((~b) >> 6);
        }

done:
        return ((s - _s) - count);
}
```

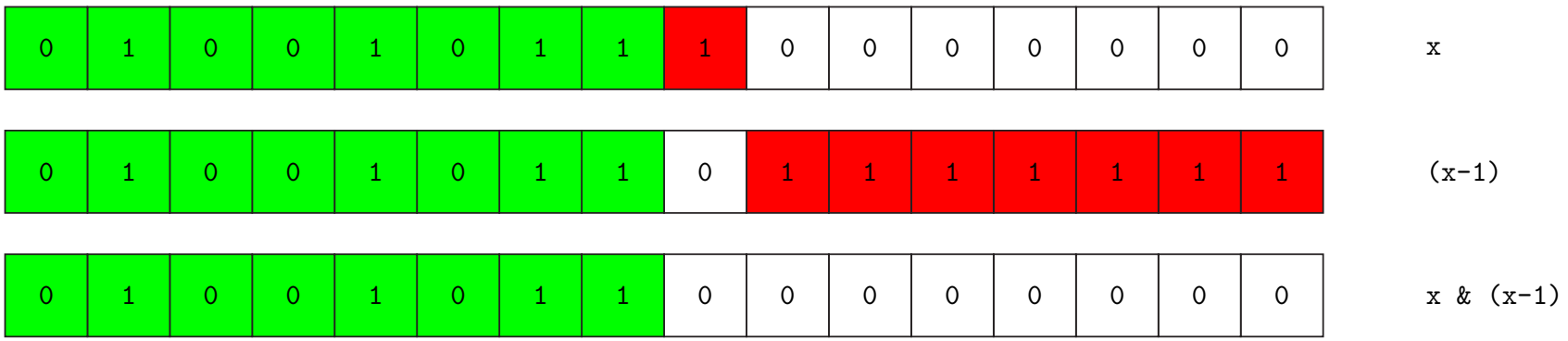
https://godbolt.org/z/Pn7xTb

```
__m128i ZERO = _mm_set1_epi8(0);
__m128i SND = _mm_set1_epi8(0x40);

for (; ; s += sizeof(__m128i)) {
        /* Prefetch 2048 bytes ahead. */
        __builtin_prefetch(&s[2048], 0, 0);

        u = *(__m128i *)(s);
        if (_mm_movemask_epi8(_mm_cmpeq_epi8(u, ZERO))) break;

        __m128i snd = _mm_andnot_si128(u, SND);
        __m128i res = _mm_and_si128(_mm_add_epi8(snd, snd), u);
        count += __builtin_popcountll(_mm_movemask_epi8(res));
}
```

Popcount

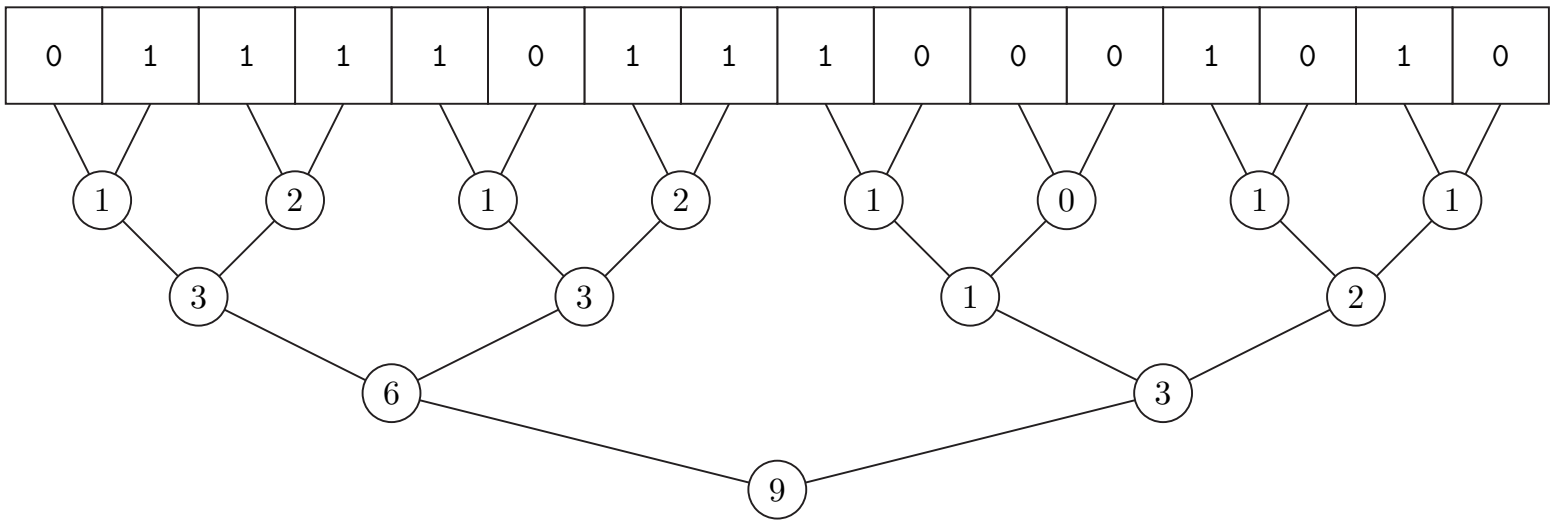

0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0
x

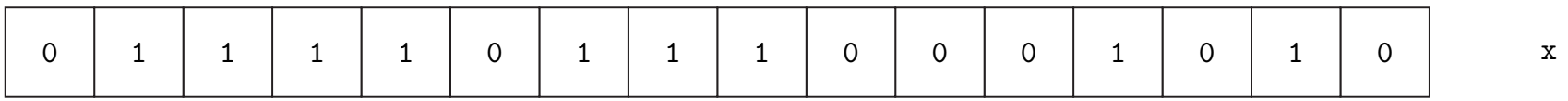

0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0
x
0 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1
(x-1)


0 1 0 0 1 0 1 1 1 0 0 0 0 0 0 0
x
0 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1
(x-1)
0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0
x & (x-1)

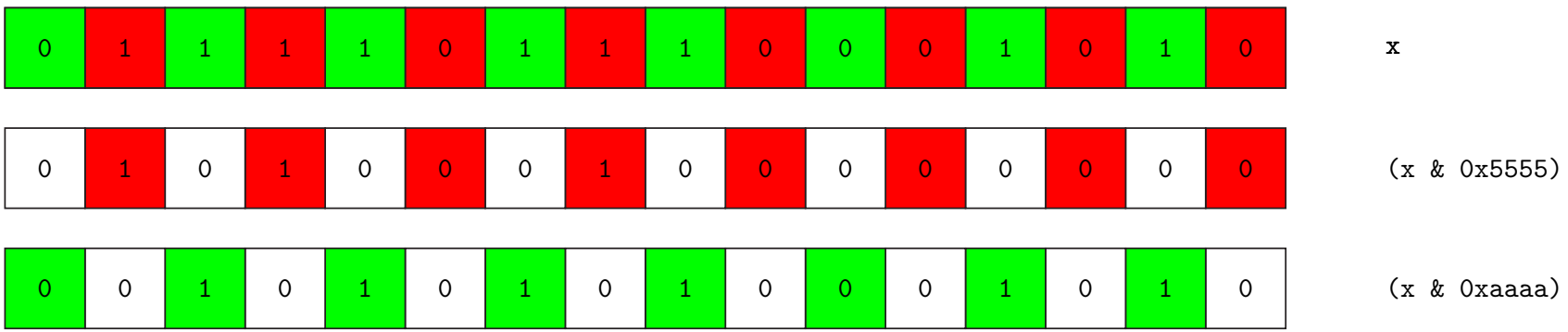
```c
int count(uint64_t x) {
  int v = 0;
  while(x != 0) {
    x &= x - 1;
    v++;
  }
  return v;
}
```

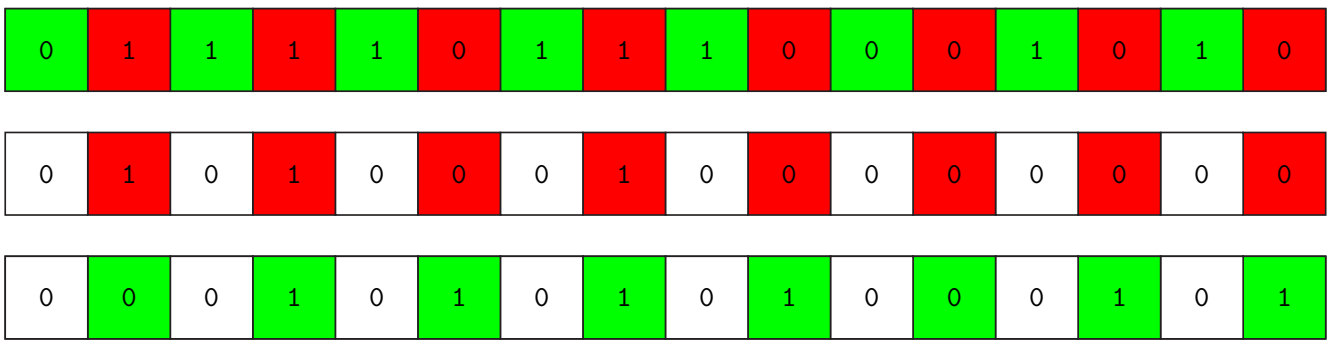
FIGURE 4. The Wegner function in C.

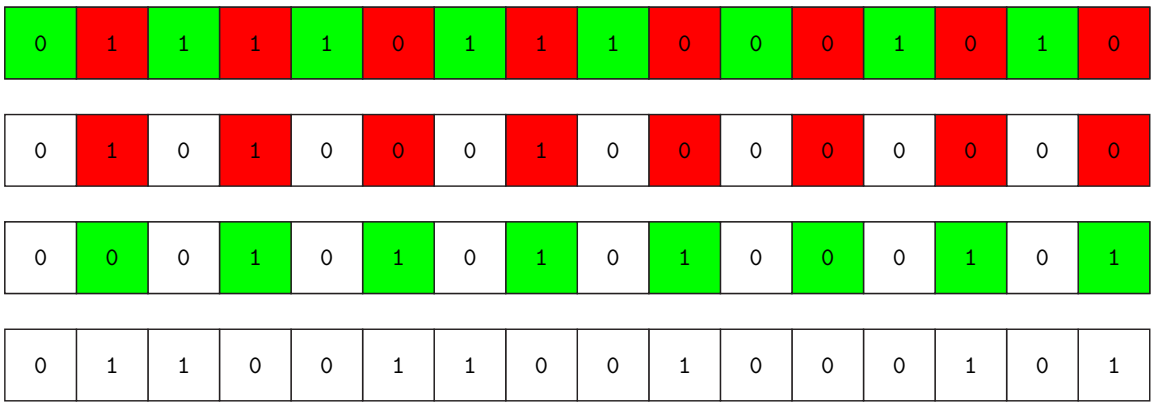

0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x
1 2 1 2 1 0 1 1
3 3 1 2
6 3
9

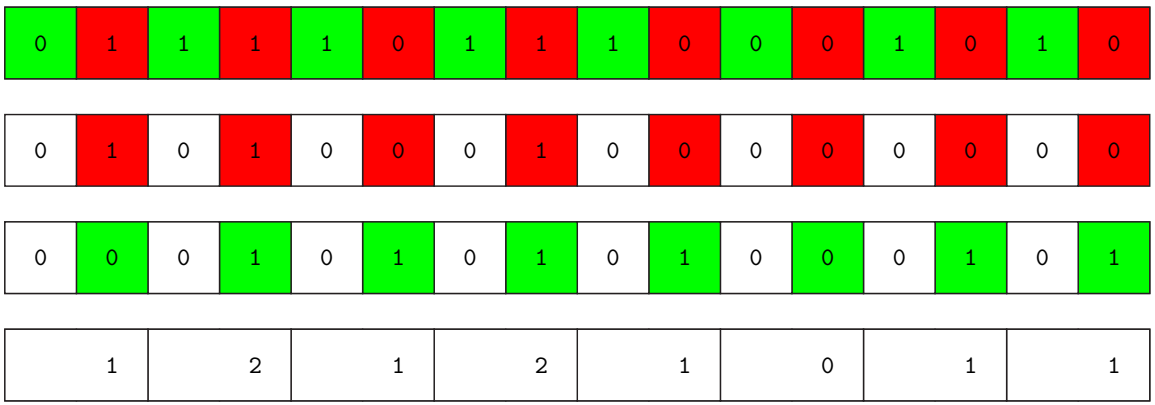

0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x


0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x

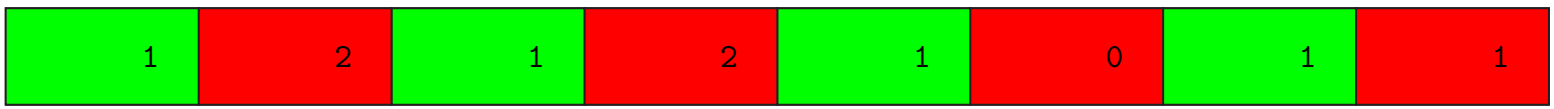

0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x
0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0
(x & 0x5555)
0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0
(x & 0xaaaa)

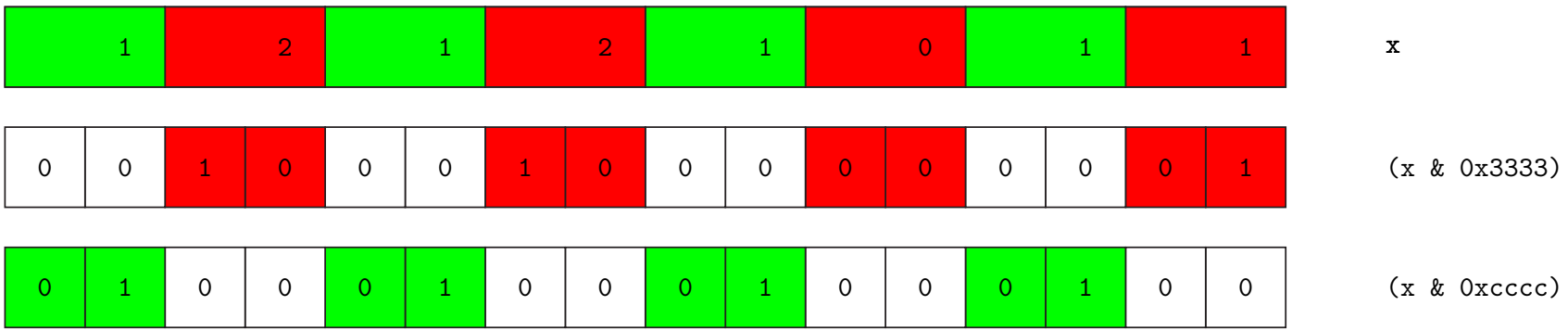

0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x
0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0
(x & 0x5555)
0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1
(x & 0xaaaa) >> 1

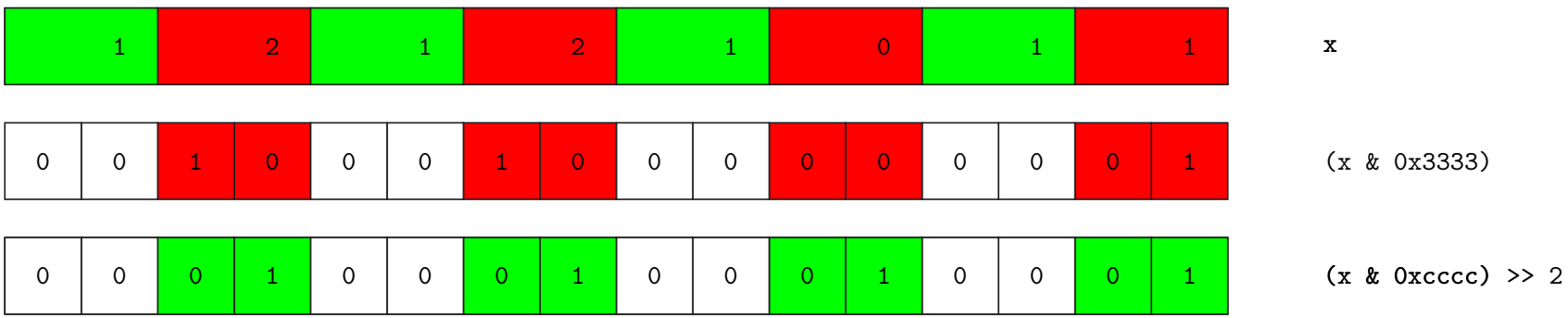

0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x
0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0
(x & 0x5555)
0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1
(x & 0xaaaa) >> 1
0 1 1 0 0 1 1 0 0 1 0 0 0 1 0 1
(x & 0x5555) + ((x & 0xaaaa) >> 1)

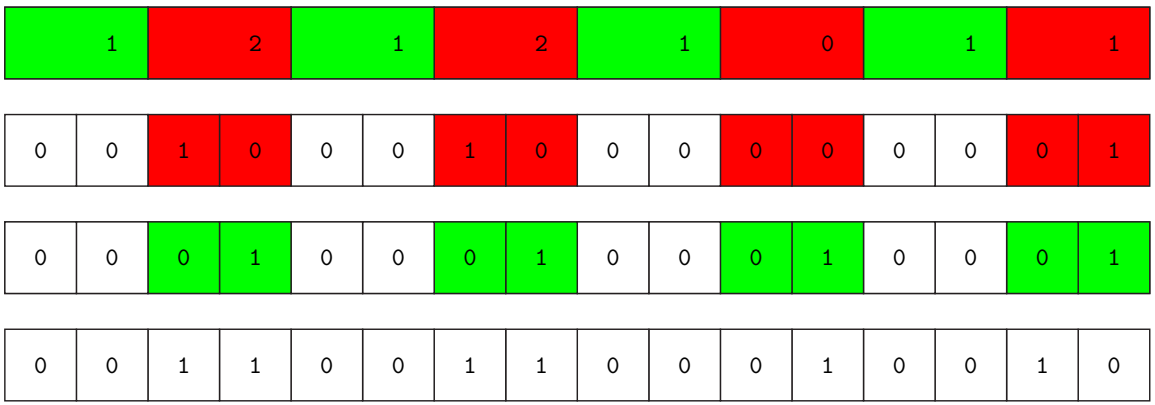

0 1 1 1 1 0 1 1 1 0 0 0 1 0 1 0
x
0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0
(x & 0x5555)
0 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1
(x & 0xaaaa) >> 1
1 2 1 2 1 0 1 1
(x & 0x5555) + ((x & 0xaaaa) >> 1)

1	2	1	2	1	0	1	1	x


1
2
1
2
1
0
1
1
x


1 2 1 2 1 0 1 1
x
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
(x & 0x3333)
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
(x & 0xcccc)


1 2 1 2 1 0 1 1
x
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
(x & 0x3333)
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
(x & 0xcccc) >> 2

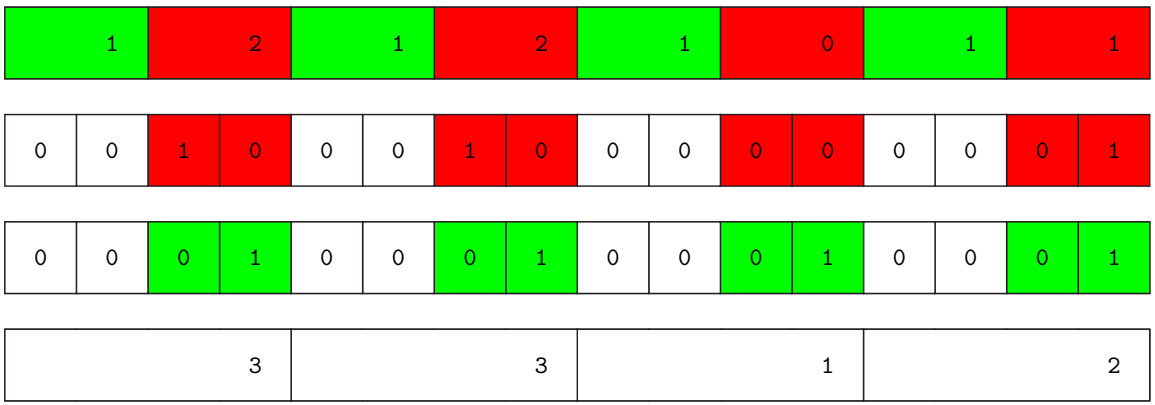


x

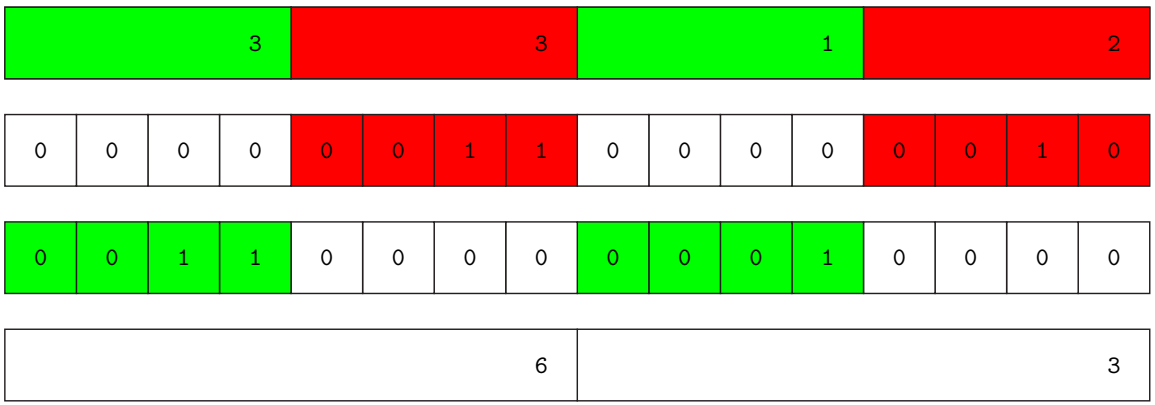
(x & 0x3333)

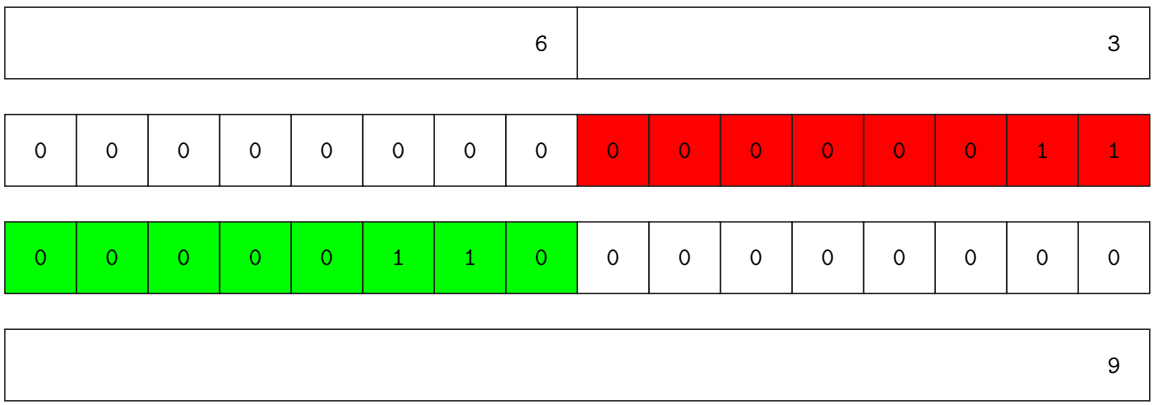
(x & 0xcccc) >> 2

(x & 0x3333) + ((x & 0xcccc) >> 2)


1 2 1 2 1 0 1 1
x
0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1
(x & 0x3333)
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
(x & 0xcccc) >> 2
3 3 1 2
(x & 0x3333) + ((x & 0xcccc) >> 2)

3	3	1	2	x

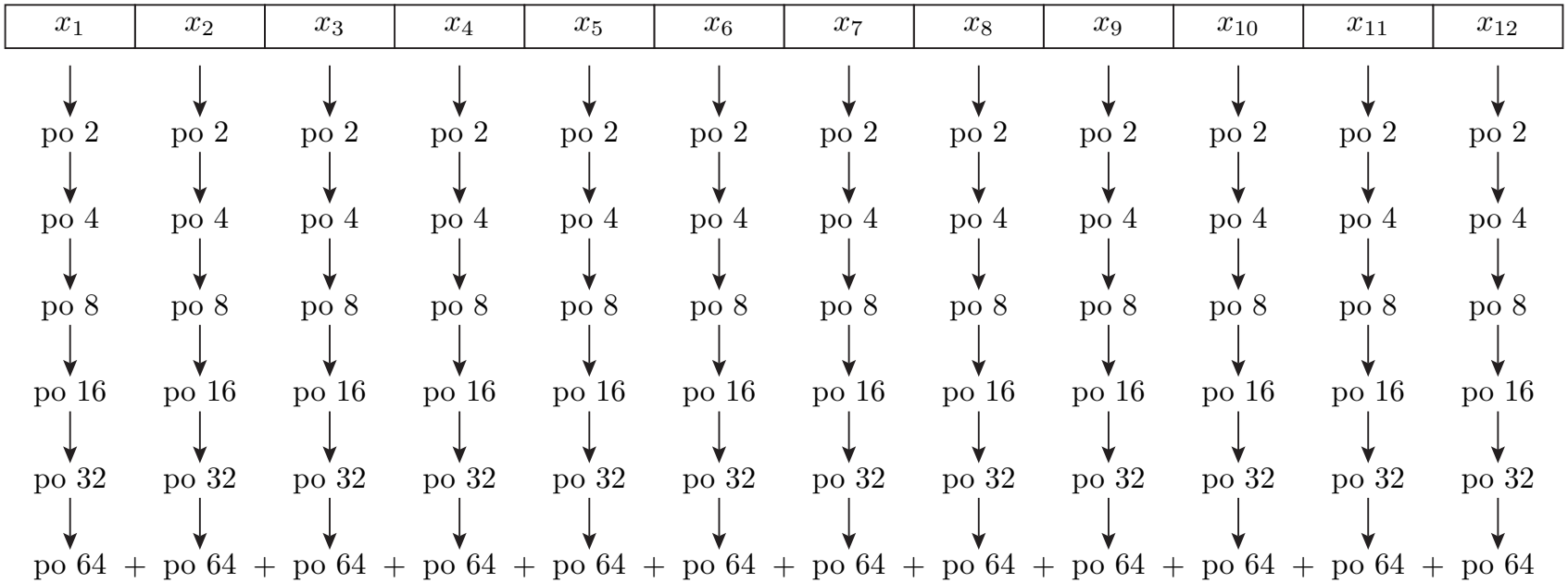

3
3
1
2
x
0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0
(x & 0x0f0f)
0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0
(x & 0xf0f0)
6
3
(x & 0x0f0f) + ((x & 0xf0f0) >> 4)

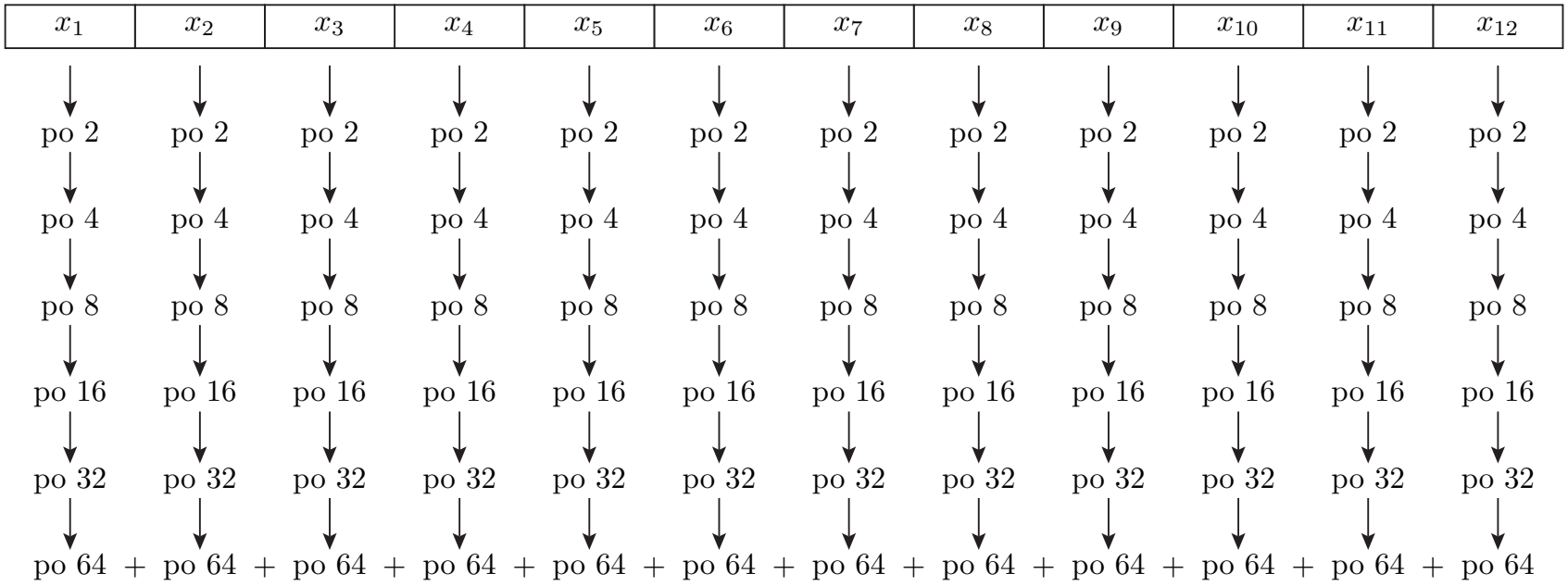

6
3
x
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
(x & 0x00ff)
0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
(x & 0xff00)
9
(x & 0x00ff) + ((x & 0xff00) >> 8)

```
uint64_t c1  = UINT64_C(0x5555555555555555);
uint64_t c2  = UINT64_C(0x3333333333333333);
uint64_t c4  = UINT64_C(0x0F0F0F0F0F0F0F0F);
uint64_t c8  = UINT64_C(0x00FF00FF00FF00FF);
uint64_t c16 = UINT64_C(0x0000FFFF0000FFFF);
uint64_t c32 = UINT64_C(0x00000000FFFFFFFF);

uint64_t count(uint64_t x) {
  x = (x & c1) + ((x >> 1) & c1);
  x = (x & c2) + ((x >> 2) & c2);
  x = (x & c4) + ((x >> 4) & c4);
  x = (x & c8) + ((x >>  8)  & c8);
  x = (x & c16)+ ((x >> 16)) & c16);
  return (x & c32) + ((x >> 32) & c32);
}
```

```
uint64_t c1  = UINT64_C(0x5555555555555555);
uint64_t c2  = UINT64_C(0x3333333333333333);
uint64_t c4  = UINT64_C(0x0F0F0F0F0F0F0F0F);

uint64_t count(uint64_t x) {
  x -= (x >> 1) & c1;
  x = (( x >> 2) & c2) + (x & c2);
  x = ( x  + (x >> 4) ) & c4;
  x *= UINT64_C(0x0101010101010101);
  return x >> 56;
}
```

FIGURE 3. The Wilkes-Wheeler-Gill function in C


x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
po 2
po 4
po 8
po 16
po 32
po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64


x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
po 2
po 4
po 8
po 16
po 32
po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64 + po 64

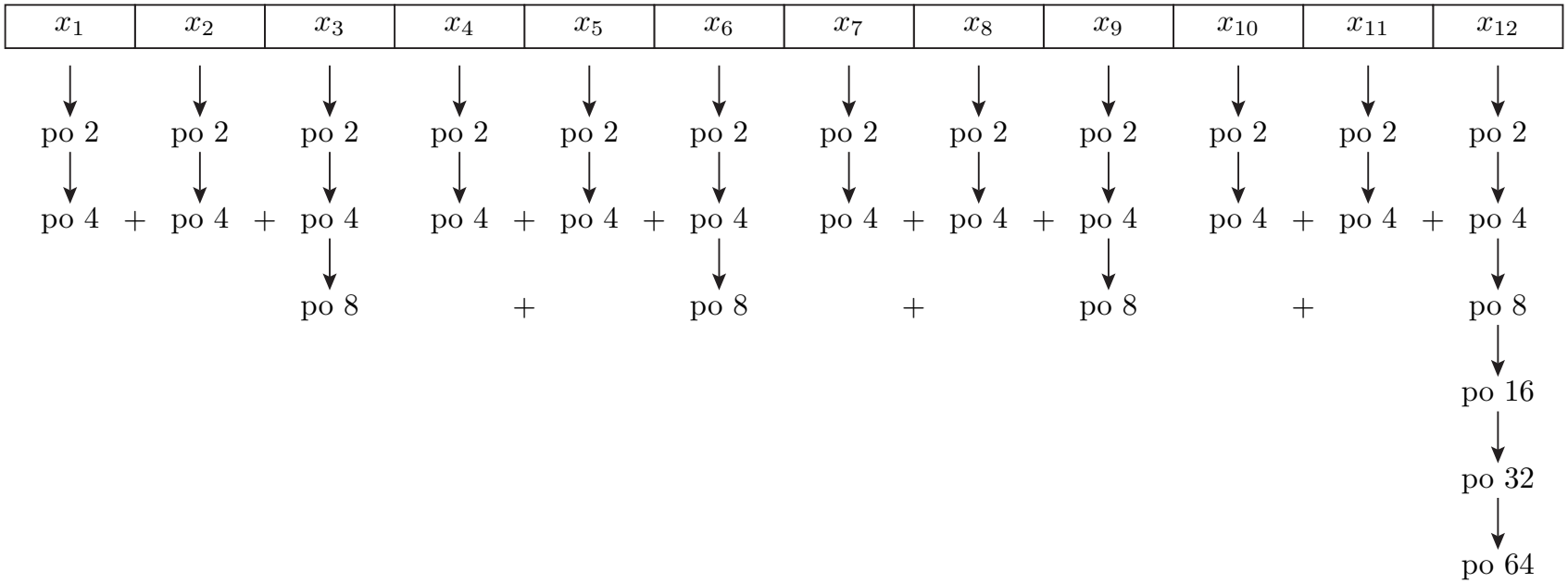
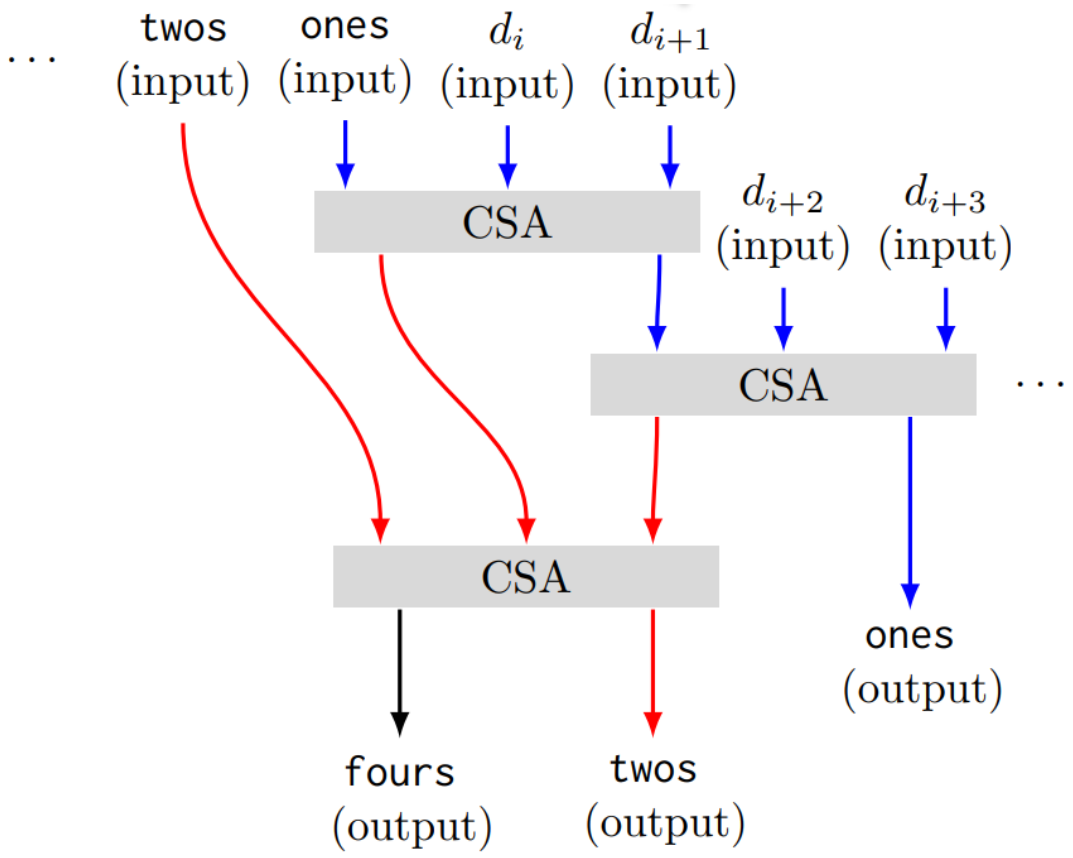

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}
po 2
po 4
po 8
po 16
po 32
po 64
+

TABLE 1. Sum of three bits $a+b+c$. We use $\oplus$ for XOR, $\wedge$ for AND and $\vee$ for OR.

a	b	c	$a+b+c$	$(a \oplus b) \oplus c$	$(a \wedge b) \vee ((a \oplus b) \wedge c)$
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	1	0
1	0	0	1	1	0
0	1	1	2	0	1
1	0	1	2	0	1
1	1	0	2	0	1
1	1	1	3	1	1


twos
(input)
ones
(input)
d_i
(input)
d_{i+1}
(input)
CSA
d_{i+2}
(input)
d_{i+3}
(input)
CSA
CSA
ones
(output)
fours
(output)
twos
(output)

1. NEoptimalizujte, nerobte prácu kompilátora
2. Zvoľte správny algoritmus a DŠ
3. Robte menej
4. Používajte zarážky
5. Zrýchlite častý/priemerný prípad
6. Využívajte efektívne cache
7. Bitová a vektorová mágia