

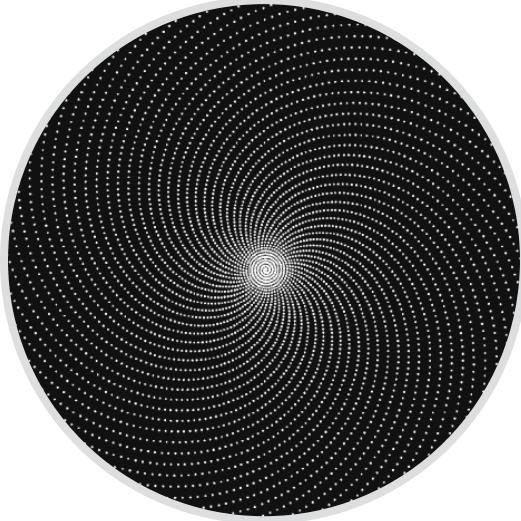
Branchless Programming

"How to make your code faster and less readable."

```
public static T CountBitsSetTo1<T>(T v)
{
    v = v - ((v >> 1) & (T)~(T)0/3);                                // temp
    v = (v & (T)~(T)0/15*3) + ((v >> 2) & (T)~(T)0/15*3);          // temp
    v = (v + (v >> 4)) & (T)~(T)0/255*15;                          // temp
    return (T)(v * ((T)~(T)0/255)) >> (sizeof(T) - 1) * CHAR_BIT; // count
}
```

example taken from "Bit Twiddling Hacks" by Sean Eron Anderson

Sooo... who is this guy?



frederik-hoeft

.NET guy studying computer science.
Interested in low level stuff, cryptography
and cyber security.

 frederik-hoeft

 @frederik.hoeft

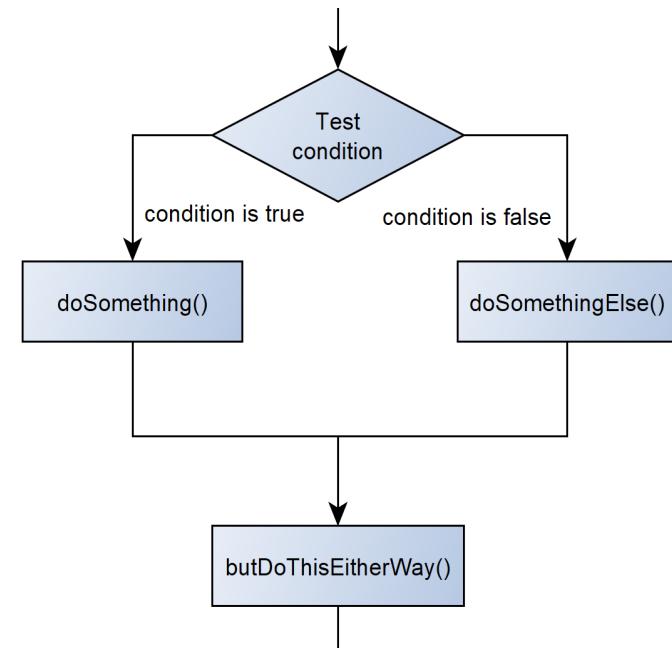
- 😐 thinks he's funny...
- 🎓 5th semester CS student @FHDW Hannover
- 💼 full-stack mobile / web dev @WKG Software GmbH
- 💼 freelancing since 2020
- 🎓 since 2018

Structure

- What are branches and why should I care?
 - An "everyday" example
- Branchless programming
 - The CPU, x86 assembly and pipelining
 - Bits, bytes and binary arithmetic
- Going branchless
- Some benchmarks
- Branchless is just the beginning
- Pitfalls, Do's and Dont's
- Questions

What are branches?

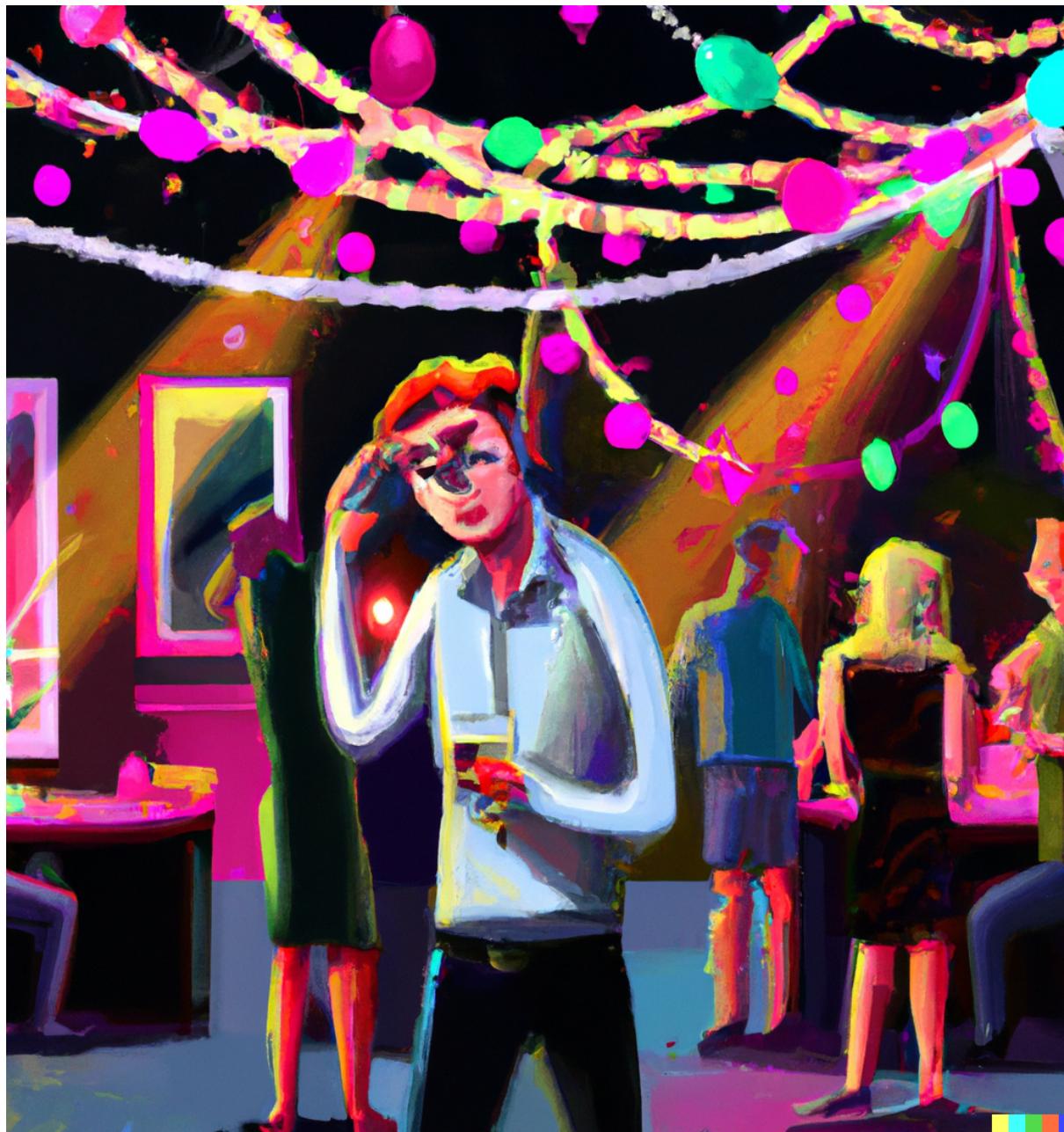
```
if (condition)
{
    doSomething();
}
else
{
    doSomethingElse();
}
butDoThisEitherWay();
```



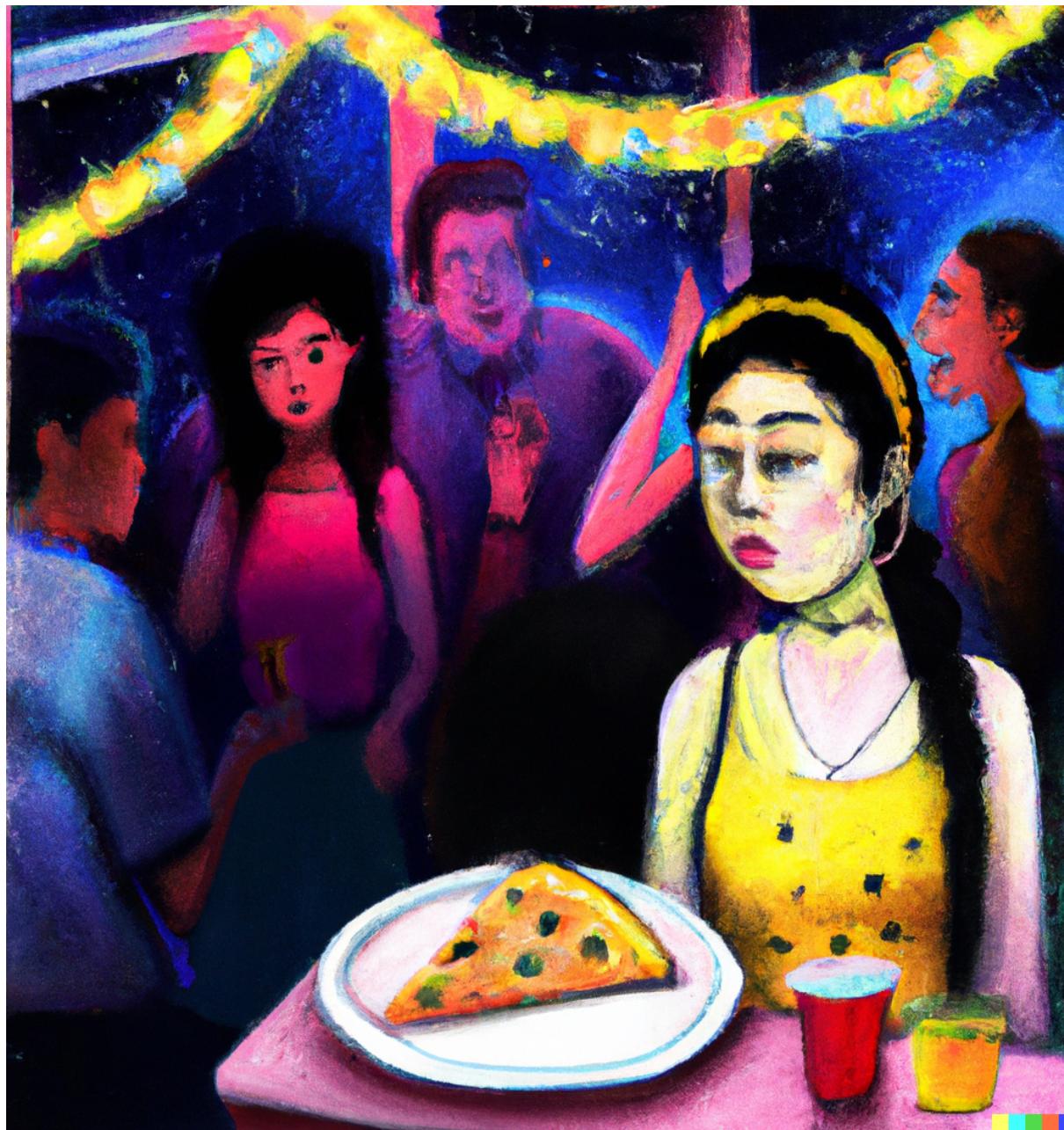
Cool... So what's wrong with it?
An introductory example



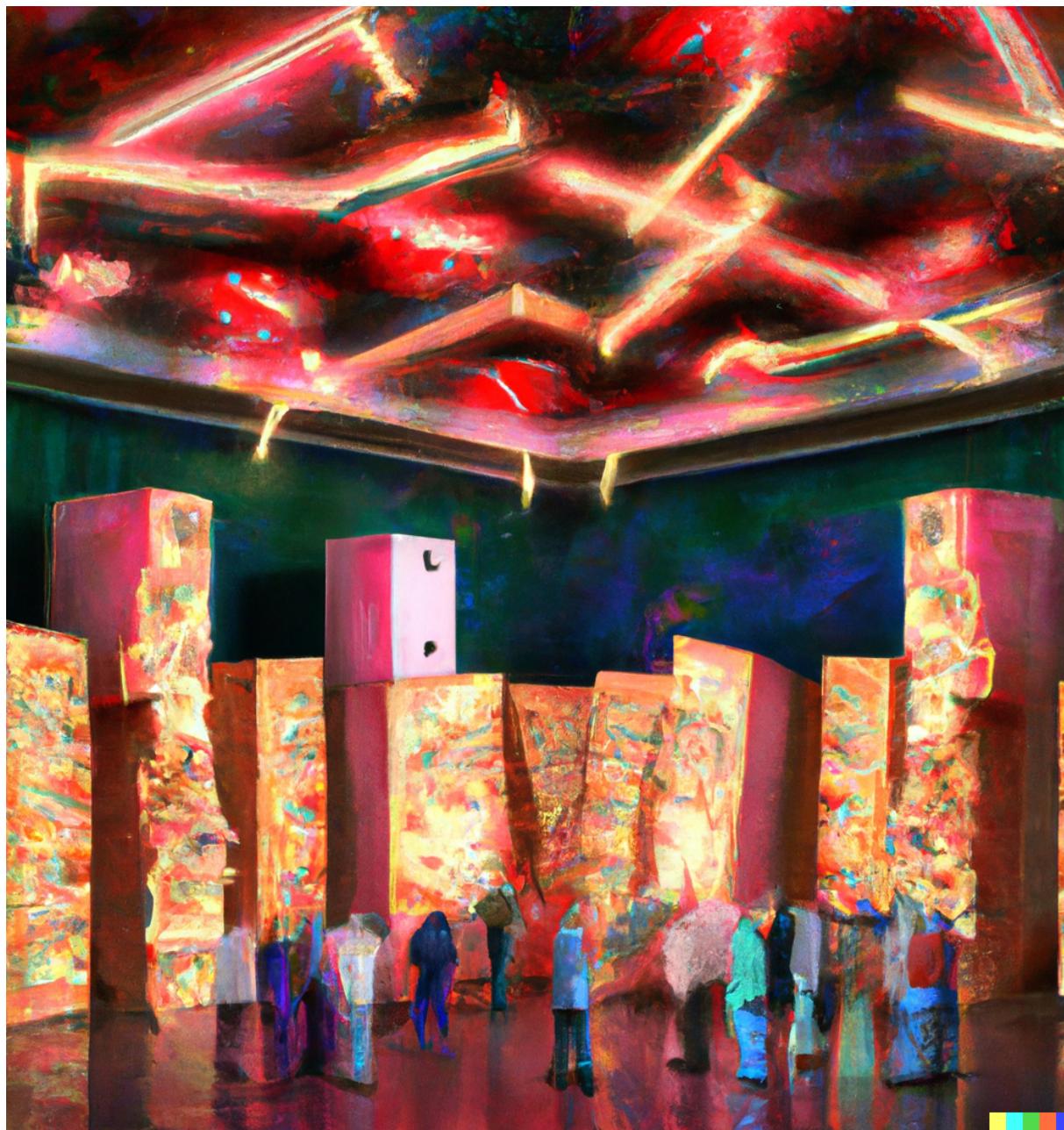
"It's Friday. This is what you could be doing right now." by DALL·E (OpenAI)



"Waiting for branch evaluation." by DALL·E (OpenAI)



"Branch evaluated: friendship assertion failed. Reason: 'pineapple pizza'." by DALL·E (OpenAI)



"Eliminate branches: just order everything. Also: wth is happening?" by DALL·E (OpenAI)



"Consequences of (over-) using branchless programming." by DALL·E (OpenAI)

For real: what *is* "branchless programming"?

- advanced optimization technique for low-level code
 - "*making things faster*"
- Eliminate branches in if, switch statements.
 - *not loops (while, for, ...)* \Rightarrow "loop unrolling"

Let's see why branches can be slow...

How does it work?

Our code:

```
int AbsoluteOf(int number)
{
    if (number >= 0) // condition
    {
        return number; // option 1
    }
    else
    {
        return number * (-1); // option 2
    }
}
```

What the CPU sees:

```
int AbsoluteOf(int number)
{
    if (number >= 0) // condition
    {
        return number; // option 1
    }
    else
    {
        return number * (-1); // option 2
    }
}
```

```
// result is eax
// number is edx
AbsoluteOf(Int32)
L0000: test edx, edx // condition
L0002: jl short L0007
L0004: mov eax, edx // option 1
L0006: ret
L0007: mov eax, edx // option 2
L0009: neg eax
L000b: ret
```

⇒ instructions: test, jl, mov, ret, mov, neg, ret

What the CPU does

You won't believe these 4 simple steps to execute an instruction:

1. Instruction Fetch (IF) "Give me an instruction"
 2. Instruction Decode (ID) "K thx. So what do I need to do?"
 3. Execute (EX) "Ah okay. I'm doing it."
 4. Write back (WB) "Yay I did it. Here's the result." 😊
- Repeat...

time	IF	→	ID	→	EX	→	WB
1t	mov						
2t			mov				
3t					mov		
4t						mov	

Pipelining

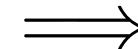
Just process multiple instructions at once:

instructions: mov, add, mov, sub, ...

time	IF	→	ID	→	EX	→	WB
1t	mov						
2t		add		mov			
3t			mov		add		mov
4t				sub		mov	
						add	
							mov

Back to our example

```
AbsoluteOf(Int32)
L0000: test edx, edx // condition
L0002: jl short L0007
L0004: mov eax, edx // option 1
L0006: ret
L0007: mov eax, edx // option 2
L0009: neg eax
L000b: ret
```



time	IF	→	ID	→	EX	→	WB
1t	test						
2t	j1		test				
3t			j1		test		
4t					j1		test

⇒ Pipeline stall: wait for branch evaluation!

Branch elimination

Branching

```
int AbsoluteOf(int number)
{
    if (number >= 0) // condition
    {
        return number; // option 1
    }
    else
    {
        return number * (-1); // option 2
    }
}
```

Branchless

```
Int32 AbsoluteOf(Int32 number)
{
    Int32 mask = (number >> 31);
    return number * (mask | 1);
}
```

```
Int32 AbsoluteOf(Int32 number)
{
    Int32 mask = (number >> 31);

    return (number + mask) ^ mask;
}
```

"Yeah, cool... But what is happening here?"

Two's complement

How are negative numbers stored?

positive decimal	positive binary	negative binary	negative decimal
10^0	$2^3 2^2 2^1 2^0$	$-2^3 2^2 2^1 2^0$	-10^0
0	0 0 0 0	0 0 0 0	- 0
1	0 0 0 1	1 1 1 1	- 1
2	0 0 1 0	1 1 1 0	- 2
3	0 0 1 1	1 1 0 1	- 3
4	0 1 0 0	1 1 0 0	- 4
5	0 1 0 1	1 0 1 1	- 5
6	0 1 1 0	1 0 1 0	- 6
7	0 1 1 1	1 0 0 1	- 7
8	-----	1 0 0 0	- 8

$\implies x \cdot (-1) = x - 1$, then invert bits.

Binary arithmetic

or: |

1001

| 0011

= 1011

and: &

1001

& 0011

= 0001

xor: ^

1001

^ 0011

= 1010

flip bits: ~

~ 1001

= 0110

signed shift right by x: >>x

1001 >>2

= 1110

Back to our example

```
Int32 AbsoluteOf(Int32 number)
{
    Int32 mask = (number >> 31);
    return number * (mask | 1);
}
```

- `Int32` \implies 32 bit signed integer
- `>> 31` \implies signed bit-shift right by 31 bits.
- `| 1` \implies "or 1"

Let's test this.

What if number = 5?

- $\text{number} = 5 = 0 \dots 0101\text{b}$
- $\text{mask} = 0 \dots 0\text{b} = 0$
- $(\text{mask} | 1) = 0 \dots 0\text{b} | 0 \dots 1\text{b} = 1$
- $\text{number} * (\text{mask} | 1) = 5 * 1 = 5$

What if number = -3?

- $\text{number} = -3 \implies 1 \dots 1101\text{b}$
- $\text{mask} = 1 \dots 1\text{b} = -1$
- $(\text{mask} | 1) = 1 \dots 1\text{b} | 0 \dots 1\text{b} = 1 \dots 1\text{b} = -1$
- $\text{number} * (\text{mask} | 1) = -3 * (-1) = 3$

Aha, so it is calculating

- $\text{number} * (0 | 1) = \text{number} * 1$, if number is positive
- $\text{number} * (-1 | 1) = \text{number} * -1$, if number is negative

What about the other version?

```
Int32 AbsoluteOf(Int32 number)
{
    Int32 mask = (number >> 31);
    return (number + mask) ^ mask;
}
```

- `Int32` \implies 32 bit signed integer
- `>> 31` \implies copy sign bit 31 times.
- `^` \implies "xor"
- `mask` will be 0 (all 0) or -1 (all 1).

Let's test this.

What if number = 5?

- 5 is positive, so mask will be 0.
- $\text{number} + \text{mask} = 5 + 0 = 5$
- $5 \wedge \text{mask} = 5 \wedge 0$
- xor all 0s does nothing
- $\implies 5 \wedge 0 = 5$

What if number = -3?

- -3 is negative, so mask will be -1 (all bits set to 1).
- $\text{number} + \text{mask} = -3 + (-1) = -4$
- $-4 \wedge \text{mask} = -4 \wedge (-1)$
- xor all 1s inverts all bits
- $\implies -4 \wedge (-1) = \sim(-4) = \sim1 \dots 100b = 0 \dots 011b = 3$

Aha, so if number is negative it returns number - 1, then inverts all bits

\implies Two's complement!

So is branchless really faster?

Let's do a benchmark!

```
int Abs(int number)
{
    if (number >= 0) // condition
    {
        return number; // option 1
    }
    else
    {
        return number * (-1); // option 2
    }
}
```

```
int AbsBranchlessMul(int number)
{
    int mask = (number >> 31);
    return number * (mask | 1);
}
```

```
int AbsBranchlessXor(int number)
{
    int mask = (number >> 31);
    return (number + mask) ^ mask;
}
```

Benchmark results

Method	Mean	Error	StdDev	Ratio
Abs	16.498 us	0.0588 us	0.0491 us	1.00
AbsBranchlessXor	6.044 us	0.0152 us	0.0127 us	0.37
AbsBranchlessMul	6.089 us	0.0399 us	0.0374 us	0.37

⇒ The branchless versions are >2.5 times faster (and more stable) than Abs() with branches!

Also the XOR version seems to be faster

Branchless is just the beginning

Branchless programming translates conditional logic into arithmetic!

Can be further optimized using more advanced techniques:

- SIMD (Single instruction, multiple data) / vector arithmetic
- ILP (Instruction-Level-parallelism)
- GP-GPU (General purpose GPU computation)

Pitfalls, Do's and Dont's

Do

- **Do** always benchmark your changes
- **Do** consider using branchless code in low-level scenarios, such as
 - cryptography
 - audio / video processing
 - game development (i.e. physics simulations)
- **Do** consider eliminating branches in
 - tight loops
 - functions that are called very frequently
- **Do** always write comments!

Don't

- **Don't** just start optimizing code without a baseline benchmark.
- **Don't** be the guy writing branchless code in high-level scenarios, such as
 - database management
 - web request handling
 - a couple of microseconds won't matter...
- **Don't** write branchless code in places that must be maintained by multiple people or are changed frequently. Everyone will hate you...
- **Don't** assume your code is translated 1:1 into assembly. Always check what your compiler generates!
- **Don't** write branchless code in interpreted languages (JavaScript, Python, ...) it won't matter.

Useful in 1% of the cases

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
private static bool TryParseMpegHeader(void* pFrame)
{
    const uint FALSE = 0x0;
    const uint TRUE = ~FALSE;
    // extract frame header and inline buffer from pFrame
    Mp3FrameHeader* self = &((Mp3Frame*)pFrame)->Header;
    byte* pHeaderBytes = (byte*)pFrame;
    // assume success if nothing fails: (~0x0)
    int success = unchecked((int)TRUE);
    // load the MPEG version from the header
    MpegVersion mpegVersion = (MpegVersion)((pHeaderBytes[1] & 0b0001_1000u) >> 3);
    // if the mpegVersion is reserved (invalid) set success to FALSE (0x0)
    // use sign extension to convert to either 0x0 or ~0x0 (FALSE or TRUE)
    success &= (-(int)(mpegVersion ^ MpegVersion.Reserved)) >> 31;
    // determine the version index of the BitRates array using the mpegVersion.
    // if the mpegVersion is Version1 then the version index should be 0 otherwise it is 1.
    // Enum member MpegVersion.Version1 has the value 3 (11 in binary).
    // any other possible value is 1 or 2 (01 or 10 in binary).
    // -> if both bits are equal (1 and 1) then the version index is 0 otherwise it is 1.
    // -> first bit XOR second bit
    // if both bits are zero (0) then the success flag will be set to FALSE already and
    int versionIndex = ((int)mpegVersion & 0b01) ^ (((int)mpegVersion & 0b10) >> 1);
    // load the MPEG layer information from the header
```

```
MpegLayer mpegLayer = (MpegLayer)((pHeaderBytes[1] & 0b0000_0110u) >> 1);
```

```
... .
```

Questions?

... or a few moments of awkward silence. It's up to you 

```
// secured my job :)

internal static bool IsValidMp3Header(void* pFrame) =>
    (unchecked((int)TRUE) & ((-(int)((MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3
    ) ^ MpegVersion.Reserved)) >> 31) & ((-(int)(MpegLayer)((((byte*)pFrame)[1] &
    0x06) >> 1)) >> 31) & (-(((byte*)pFrame)[2] & 0xF0) >> 4 ^ 15) >> 31) &
    (-BitRates[((int)(MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3) & 0b01) ^
    ((int)(MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3) & 0b10) >> 1], (int)
    (MpegLayer)((((byte*)pFrame)[1] & 0x06) >> 1), (((byte*)pFrame)[2] & 0xF0) >> 4
    ] >> 31) & (-(((byte*)pFrame)[2] & 0xF0) >> 4 ^ 3) >> 31) & (~((-int)
    ((ChannelMode)((((byte*)pFrame)[3] & 0xC0) >> 6) ^ ChannelMode.JointStereo) >>
    31) & (-(((byte*)pFrame)[3] & 0x30) >> 4) >> 31))) & (~((MAX_FRAME_SIZE -
    (((SamplesPerFrame[((int)(MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3) &
    0b01) ^ (((int)(MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3) & 0x2) >> 1),
    (int)(MpegLayer)((((byte*)pFrame)[1] & 0x06) >> 1)] >> 3) * BitRates[((int)
    (MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3) & 0x1) ^ (((int)(MpegVersion)
    (((byte*)pFrame)[1] & 0x18) >> 3) & 0b10) >> 1], (int)(MpegLayer)((((byte*)
    pFrame)[1] & 0x06) >> 1), (((byte*)pFrame)[2] & 0xF0) >> 4] / SampleRates
    [(int)(MpegVersion)((((byte*)pFrame)[1] & 0x18) >> 3), (((byte*)pFrame)[2]
    & 0xC0) >> 2]) + (((byte*)pFrame)[2] & 0x2) >> 1)) << (~(-int)((MpegLayer)
    (((byte*)pFrame)[1] & 0x06) >> 1) ^ MpegLayer.Layer1) >> 31) & 2))) >> 31)))
== unchecked((int)TRUE);
```

"Documentation? Just read the code. The code: ...  "