Motivation:

- lack of overflow flag means checking for overflow wastes gas and storage on pointless comparisons
- lack of carry flag makes >256 bit arithmetic slow because it has to be reimplemented via comparisons

If these two flags were to be added there's no reason not to add two already existing flags - sign (is the result negative or not) and zero; useful for optimizations in hll compilation in some instances (especially checking if a result of an arithmetic operation is negative or zero).

Performance impact: positive with jit, negligible with interpreter. Every arithmetic operation onx86/amd64 already sets these flags, they only have to be made available. Arm has these flags too and can update them for bitwise/arithmetic instructions except div/mul. In jit cpu flags can be accessed directly, making them available in evm means less pointless comparisons, resulting in less cpu instructions.

How:

- make every arithmetic&bitwise evm op change the value of a byte register with four one-bit flags: carry, zero, sign, overflow.
- add a PUSHF opcode that pushes the flags byte.

This is enough to make multiple-precision arithmetic and overflow checking faster & easier.

Small extension 1:

Add an 'overflow_throw' flag into the flags registers + a POPF opcode to set it. When the overflow_throws flag is set arithmetic opcodes call revert (or throw if some exception handling was added in the future to evm) on overflow. That would directly emulate current assert on overflow behavior.

Larger extension 2:

Add cmp + conditional jump instructions.

Why - lack of flag-setting comparison and flag-conditional jumps makes repeated comparisons wasteful, consider this example:

if(a == b) { s1; } else if(a < b) { s2; } else { s3; }

which currently translates to something like this:

mload b

mload a

dup2

dup2

eq

push s1

jumpi

lt

push s2

jumpi

;; s3 here

with cmp+jcc instructions like on a x86/arm cpu:

mload b

mload a

cmp ;;compares and sets flags

jz s1

```
jb s2

;; s3 here
```