Half-precision in LLVM libc

Nicolas Celik – Google Summer of Code 2024

LLVM libc

- C standard library implementation part of the LLVM Project
- Written in C++
- Supports x86-64, AArch64, 32-bit ARM, 32-bit and 64-bit RISC-V
- Also supports AMD and NVIDIA GPUs
- Learn more at https://libc.llvm.org/

Half-precision

- C23 defines new _FloatN types
- Among them, _Float16
  - Corresponds to IEEE 754’s binary16 format
  - Also known as half-precision or FP16
  - Use case examples: neural networks, graphics
- C23 also defines new _FloatN-typed math functions accordingly
  - float fabsf(float x); → _FloatN fabsfN(_FloatN x);
  - Example: _Float16 fabsf16(_Float16 x);

Half-precision in LLVM libc

- This Google Summer of Code project aimed to implement C23 _Float16 math functions in LLVM libc
- A step toward C23 support in LLVM libc
- Makes LLVM libc is the first known libc to implement C23 _Float16 math functions

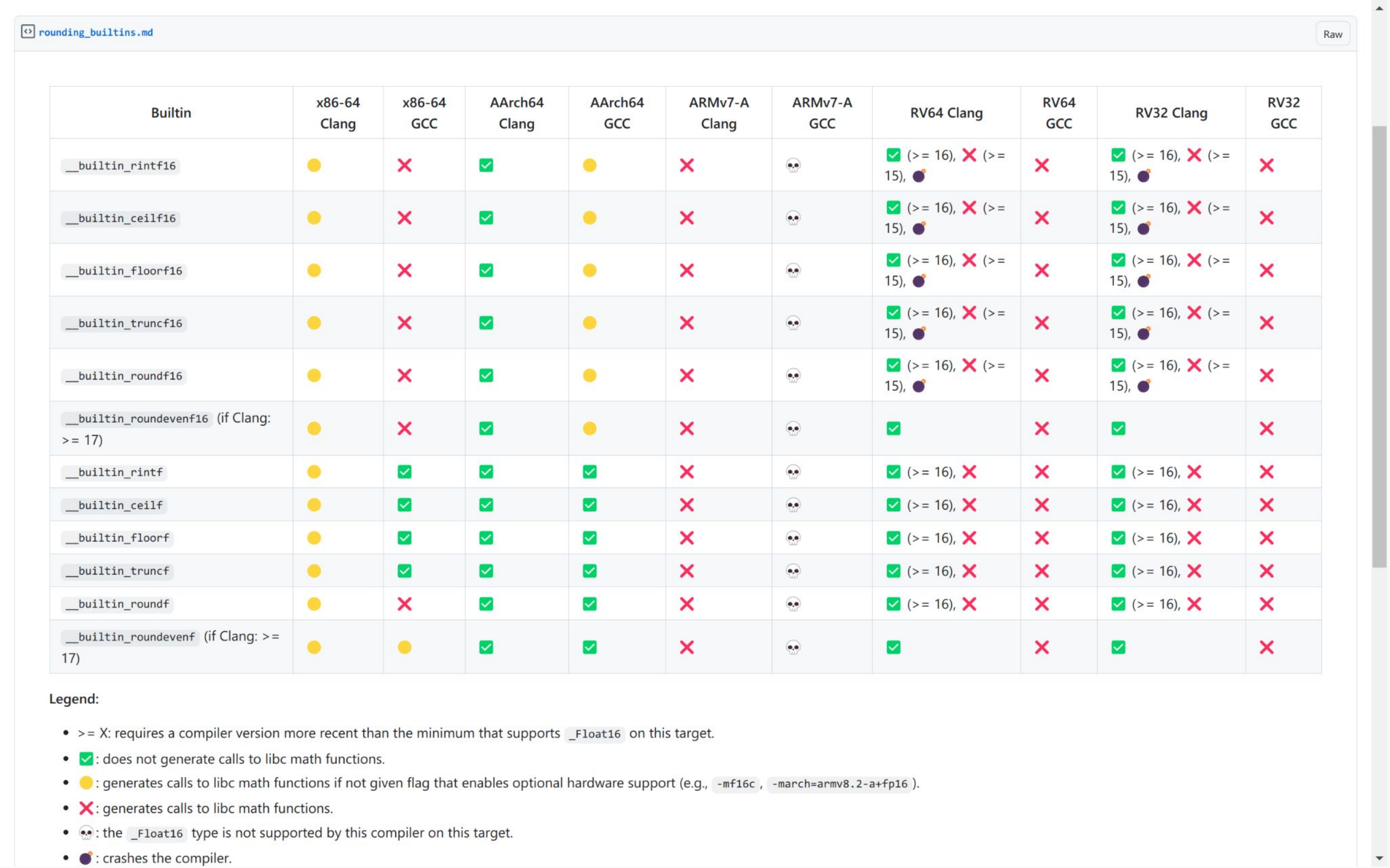
Project Goals

1. Basic operations

- Examples: fabsf16, roundf16, fmaximumf16, ufromfpf16, f16addf128
- Implemented using simple bit-manipulation algorithms
- All 70 planned _Float16 basic operations have been implemented
- https://github.com/llvm/llvm-project/issues/93566

2. Optimizations

- Optimize certain basic operations using compiler builtins
- We wanted to avoid using inline assembly and target-specific intrinsics
- The _Float16, float and double variants of the following functions have been optimized:
  - ceil, floor, rint, round, roundeven, trunc
  - copysign
  - fmax, fmin, fmaximum, fminimum, fmaximum_num, fminimum_num



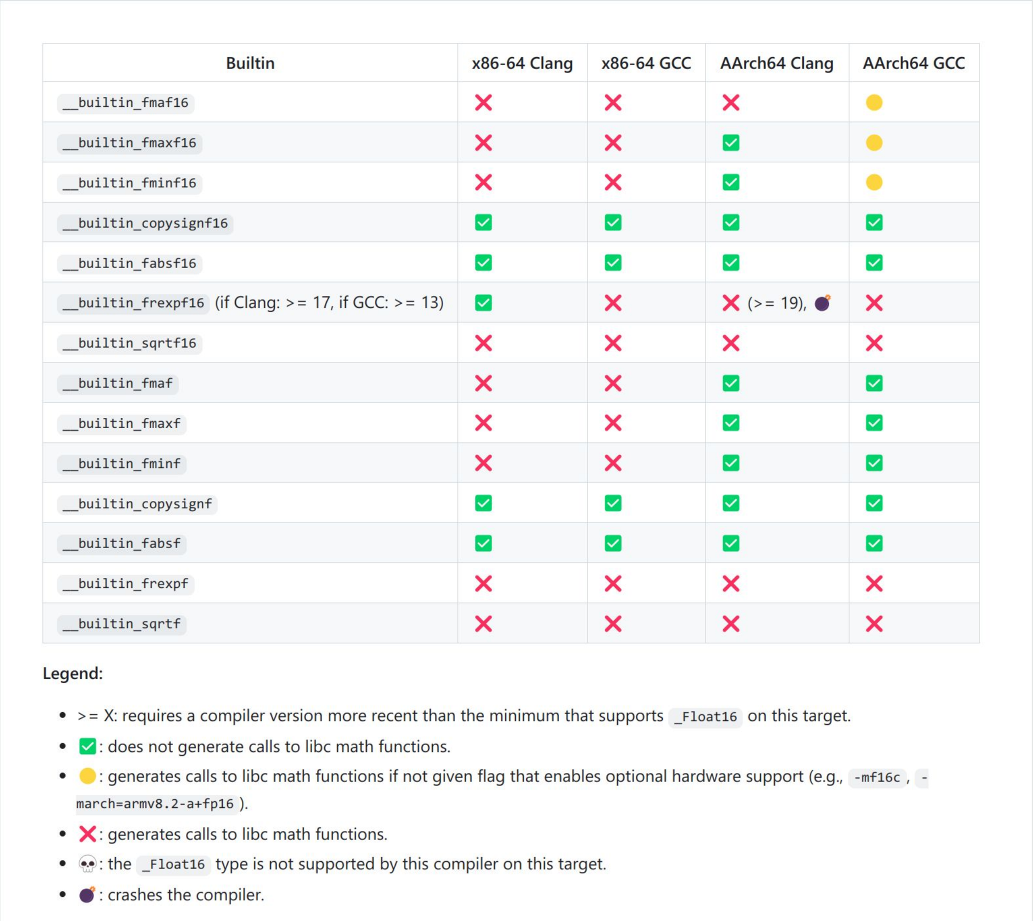
rounding_builtins.md

Builtin	x86-64 Clang	x86-64 GCC	AArch64 Clang	AArch64 GCC	ARMv7-A Clang	ARMv7-A GCC	RV64 Clang	RV64 GCC	RV32 Clang	RV32 GCC
`__builtin_rintf16`	🟡	❌	✅	🟡	❌	💀	✅ (>= 16), ❌ (>= 15), 💣	❌	✅ (>= 16), ❌ (>= 15), 💣	❌
`__builtin_ceilf16`	🟡	❌	✅	🟡	❌	💀	✅ (>= 16), ❌ (>= 15), 💣	❌	✅ (>= 16), ❌ (>= 15), 💣	❌
`__builtin_floorf16`	🟡	❌	✅	🟡	❌	💀	✅ (>= 16), ❌ (>= 15), 💣	❌	✅ (>= 16), ❌ (>= 15), 💣	❌
`__builtin_truncf16`	🟡	❌	✅	🟡	❌	💀	✅ (>= 16), ❌ (>= 15), 💣	❌	✅ (>= 16), ❌ (>= 15), 💣	❌
`__builtin_roundf16`	🟡	❌	✅	🟡	❌	💀	✅ (>= 16), ❌ (>= 15), 💣	❌	✅ (>= 16), ❌ (>= 15), 💣	❌
`__builtin_roundevenf16` (if Clang: >= 17)	🟡	❌	✅	🟡	❌	💀	✅	❌	✅	❌
`__builtin_rintf`	🟡	✅	✅	✅	❌	💀	✅ (>= 16), ❌	❌	✅ (>= 16), ❌	❌
`__builtin_ceilf`	🟡	✅	✅	✅	❌	💀	✅ (>= 16), ❌	❌	✅ (>= 16), ❌	❌
`__builtin_floorf`	🟡	✅	✅	✅	❌	💀	✅ (>= 16), ❌	❌	✅ (>= 16), ❌	❌
`__builtin_truncf`	🟡	✅	✅	✅	❌	💀	✅ (>= 16), ❌	❌	✅ (>= 16), ❌	❌
`__builtin_roundf`	🟡	❌	✅	✅	❌	💀	✅ (>= 16), ❌	❌	✅ (>= 16), ❌	❌
`__builtin_roundevenf` (if Clang: >= 17)	🟡	🟡	✅	✅	❌	💀	✅	❌	✅	❌

Legend:

- >= X: requires a compiler version more recent than the minimum that supports `_Float16` on this target.
- ✅: does not generate calls to libc math functions.
- 🟡: generates calls to libc math functions if not given flag that enables optional hardware support (e.g., `-mf16c`, `-march=armv8.2-a+fp16`).
- ❌: generates calls to libc math functions.
- 💀: the `_Float16` type is not supported by this compiler on this target.
- 💣: crashes the compiler.

https://gist.github.com/overmighty/9c8e17f9580594cc00cce82f83caef60



Builtin	x86-64 Clang	x86-64 GCC	AArch64 Clang	AArch64 GCC
`__builtin_fmaf16`	❌	❌	❌	🟡
`__builtin_fmaxf16`	❌	❌	✅	🟡
`__builtin_fminf16`	❌	❌	✅	🟡
`__builtin_copysignf16`	✅	✅	✅	✅
`__builtin_fabsf16`	✅	✅	✅	✅
`__builtin_frexpf16` (if Clang: >= 17, if GCC: >= 13)	✅	❌	❌ (>= 19), 💣	❌
`__builtin_sqrtf16`	❌	❌	❌	❌
`__builtin_fmaf`	❌	❌	✅	✅
`__builtin_fmaxf`	❌	❌	✅	✅
`__builtin_fminf`	❌	❌	✅	✅
`__builtin_copysignf`	✅	✅	✅	✅
`__builtin_fabsf`	✅	✅	✅	✅
`__builtin_frexpf`	❌	❌	❌	❌
`__builtin_sqrtf`	❌	❌	❌	❌

Legend:

- >= X: requires a compiler version more recent than the minimum that supports `_Float16` on this target.
- ✅: does not generate calls to libc math functions.
- 🟡: generates calls to libc math functions if not given flag that enables optional hardware support (e.g., `-mf16c`, `-march=armv8.2-a+fp16`).
- ❌: generates calls to libc math functions.
- 💀: the `_Float16` type is not supported by this compiler on this target.
- 💣: crashes the compiler.

https://gist.github.com/overmighty/a9a9de847eb11c667ba6b257375afe83

Performance

- ceilf16 on Google Tensor G3 (Pixel 8) (Clang 17):
  - Generic implementation: 1.38 - 8.92 ns
  - Builtin-based implementation: 0.70 - 0.79 ns
- fmaxf16 on Intel Core i7-13700H (without F16C) (Clang 18):
  - Generic implementation: 7.19 - 133.4 ns
  - Builtin-based implementation: 3.81 ns
- fmaxf16 on Intel Core i7-13700H (with F16C) (Clang 18):
  - Generic implementation: 6.17 ns
  - Builtin-based implementation: 3.81 ns

3. Higher math functions

- Examples: expf16, exp2m1f16, logf16, coshf16, sinhf16
- Require more math to implement, e.g., polynomial approximations
- We knew we couldn’t implement them all during Google Summer of Code
- 17 out of the 54 planned higher math functions have been implemented
- https://github.com/llvm/llvm-project/issues/95250

Issues encountered

Compiler bugs: old Clang crashes

- Clang 11 is still supported by LLVM libc and used in post-merge CI
- Crashes when compiling some of the _Float16 code for AArch64

```
fatal error: error in backend: Cannot select: 0x367e6b60: f16 = fp_round 0x3693a720,
TargetConstant:i64<0>, llvm-project/libc/src/__support/FPUtil/generic/FMA.h:191:33
  0x3693a720: f128,ch,glue = CopyFromReg 0x3693d2a0, Register:f128 $q0, 0x3693d2a0:1,
llvm-project/libc/src/__support/FPUtil/generic/FMA.h:191:39
    0x3693ca80: f128 = Register $q0
…
```

Compiler bugs: current Clang miscompiles

- Targets may not have full hardware support for half-precision
- They may only have conversion instructions or no hardware support at all
- Current versions of Clang may generate conversions that result in incorrect behavior

Compiler bugs: current Clang miscompiles

Clang 16:

```
__llvm_libc_20_0_0_git::fabsf16(_Float16):
        push    rbp
        mov     rbp, rsp
        vpextrw eax, xmm0, 0
        and     eax, 32767
        vpinsrw xmm0, xmm0, eax, 0
        pop     rbp
        ret
```

Clang 19:

```
.LCPI0_0:
        .long   0x7fffffff
__llvm_libc_20_0_0_git::fabsf16(_Float16):
        push    rbp
        mov     rbp, rsp
        call    __extendhfsf2@PLT
        vbroadcastss    xmm1, dword ptr [rip
+ .LCPI0_0]
        vandps  xmm0, xmm0, xmm1
        call    __truncsfhf2@PLT
        pop     rbp
        ret
```

Suboptimal codegen

GCC 14 (1.33-1.64 ns on i7-13700H):

```
foo(_Float16):
        vpxor   xmm1, xmm1, xmm1
        vpblendw        xmm0, xmm1, xmm0, 1
        vcvtph2ps       xmm0, xmm0
        vroundss        xmm0, xmm0, xmm0, 10
        vinsertps       xmm0, xmm0, xmm0, 0xe
        vcvtps2ph       xmm0, xmm0, 4
        ret
```

Clang 18 (~9.12 ns on i7-13700H):

```
foo(_Float16):
        push    rbp
        mov     rbp, rsp
        vpextrw eax, xmm0, 0
        vmovd   xmm0, eax
        vcvtph2ps       xmm0, xmm0
        vroundss        xmm0, xmm0, xmm0, 10
        vcvtps2ph       xmm0, xmm0, 4
        vmovd   eax, xmm0
        vpinsrw xmm0, xmm0, eax, 0
        pop     rbp
        ret
```

Compiler runtime issues: missing builtins

- The libgcc versions used on 32-bit Arm and RISC-V post-merge CI are missing builtins to convert from/to _Float16
- compiler-rt is missing builtins to convert between _Float16 and x86 long double

Compiler runtime issues: incorrect behavior

- libgcc implements _Float16 conversion builtins differently for 32-bit Arm: always rounds to nearest, ties to even (ignores actual CPU rounding mode)
- compiler-rt uses the same implementation on all targets, always rounds to nearest, ties to even
- Before LLVM 19, compiler-rt may incorrectly use GPRs instead of vector/FP registers for _Float16, depending on how it was built

Lessons learned

- Issues that you can encounter when implementing functions for a new floating-point type in a library:
  - Compiler bugs
  - Suboptimal codegen
  - Compiler runtime bugs
- The less hardware support for the type a target has, the more likely you are to run into issues with it
- Targets without hardware support for the new FP type may not be a priority for hardware vendors' compiler teams

Easier addition of new FP types in libraries

- Implement fully working soft-float operations in compilers from the beginning
- Implement optimized codegen for targets with hardware support later
- Save libraries from tangled conditions based on compiler version and target to enable support for new types
- Would require convincing hardware vendors to change their priorities

Conclusion

- LLVM libc is the first known libc to implement C23 _Float16 math functions
- Not all _Float16 math functions are implemented yet
- All of them are supported on x86-64
- Some were temporarily disabled on AArch64 and GPUs
- All were temporarily disabled on 32-bit Arm and on RISC-V due to compiler runtime issues
    - We’re working on enabling them back