# C Optimizations

## Code Alternatives

- XINT, Y, Z - Represent integers (either a literal or a variable)

| From | To | Notes |
|---|---|---|
| `(XINT/8)*16` | `XINT*2` *or* `XINT<<1` | Apply math properties (like simplification and distribution); apply bit-shifting when possible |
| `(XINT/Y)>Z` | `XINT>(Y*Z)` | Multiplication is faster than division |
| `char` *and* `short` | `int` | Integers are processed more quickly than "char" or "short" |
| `for(i=0; i<10;i++)` | `for(i=10;--i;)` | For-loops counting down to zero are faster than other for-loops |
| `Global variable` | `Local variable` | Local variables are more quickly accessed than global variables |
| `if-else` | `switch-case` | Switch-constructs are faster than if-else-constructs |
| `signed_int/signed_int` | `unsigned_int/unsigned_int` | Unsigned-division is faster than signed-division |
| `TYPE VAR;` | `const TYPE VAR;` | If a variable's value will not be changed, declare it as a constant |
| `XINT*8` | `XINT<<3` | Bit-shifting is less intensive than multiplication; this tip only works when multiplying by a power-of-2 |
| `XINT/10` | `XINT*0.1` | Multiplication is faster than division |
| `XINT/16` | `XINT>>4` | Bit-shifting is faster than division; this tip only works when dividing by a power-of-2; 16 = 2^4 |
| `XINT%16` | `XINT&0x000f` | Hex integers and ANDing are processed more quickly |
| `XINT%32` | `XINT&31` | ANDing is less intensive than modulus; this tip only works when using a power-of-2 against the integer |
| `i++` | `++i` | Pre-increment is usually faster |

**NOTE:** Some compilers may apply these optimizations themselves.

## GNU-GCC Optimizations Flags

| | |
|---|---|
| `-fdata-sections` | Place Data items in separate section; improves reference locality in the instruction-space on some systems; executable may be larger; linker may have better dead code removal; may prevent gprof and debugging; do not use on static libraries |
| `-ffast-math` | Sets the options -fno-math-errno, -funsafe-math-optimizations, -ffinite-math-only, -fno-rounding-math, -fno-signaling-nans and -fcx-limited-range. Breaks IEEE and ISO rules. |
| `-ffunction-sections` | Place functions in separate section; improves reference locality in the instruction-space on some systems; executable may be larger; linker may have better dead code removal; may prevent gprof and debugging; do not use on static libraries |
| `-fgcse-lm` | Move loads out of loops |
| `-fgcse-sm` | Move stores out of loops |
| `-fstack-protector` | Provide extra code for checking for buffer-overflows and stack smashing attacks |
| `-fstack-protector-all` | Provide extra code for checking for buffer-overflows; protection added to all functions |
| `-floop-nest-optimize` | Enable the isl based loop nest optimizer. This is a generic loop nest optimizer based on the Pluto optimization algorithms. |
| `-flto -fuse-linker-plugin` | Enable the link-time optimizer; do not use with -fwhole-program |
| `-fmerge-all-constants` | Attempt to merge identical constants and identical variables |
| `-fmodulo-sched` | Perform swing modulo scheduling |
| `-fno-exceptions` | Disables exception handling |
| `-fno-sanitize=all` | Disable sanitizers |
| `-fno-stack-protector` | Disable stack protectors |
| `-fselective-scheduling -fselective-scheduling2 -fsel-sched-pipelining -fsel-sched-pipelining-outer-loops` | Pipeline inner and outer loops |
| `-funroll-all-loops` | Unroll all loops (even with unknown number of iterations); may cause the executable to run less quickly and increase size |
| `-funroll-loops` | Unroll loops with known number of iterations at compile-time; may increase executable size |
| `-funsafe-math-optimizations` | Float-point optimizations; breaks IEEE and ISO rules. |
| `-funswitch-loops` | Move branches with loop-invariant-conditions out of the loop |
| `-g0` | Do not add debugging info |
| `-gtoggle` | Turn off generation of debugging info |
| `-march=*` | Compile and optimize code for using the special features of the specified CPU |
| `-minline-all-stringops` | (x86) Allows extra string inlining; speeds up code that uses memcpy, memset, and strlen |
| `-mlong-double-128` | (x86) Set the size of "long double" to 128 bits |
| `-mmmx and -msse4` | (x86) Enable use of MMX instructions and SSE4, respectively |
| `-msse2avx` | (x86) Encode SSE instructions with VEX |
| `-O3` | Apply level-3 optimizations |
| `-s` | Remove all symbol table and relocation info |
| `-Wl,--gc-sections` | Enable garbage collection of unused input sections |
| `-Wl,-O3` | Use level-3 linker optimizations |
| `-Wl,-s` | Strip all symbols during link-time |
| `-Wl,-S` | Strip debugging symbols during link-time |
| `-Wl,--no-whole-archive` | Only use needed symbols from archive files |
| `-Wl,-x` | Strip local symbols during link-time |
| `-Wl,-X` | Strip temporary local symbols during link-time |
| `-Wl,-z,relro,-z,now` | non-PLT GOT and GOT are read-only |

**NOTE:** See https://gcc.gnu.org/onlinedocs/gcc/

## Strip

| | |
|---|---|
| `--discard-all` | Remove non-global symbols |
| `--discard-locals` | Remove compiler-generated local symbols |
| `--only-keep-debug` | Remove all symbols that would not be removed with "--strip-debug" |
| `--remove-section=.comment` | Remove ".comment" section; ".comment" stores compiler version information |
| `--remove-section=.note` | Remove ".note" section; ".note" stores linker version information |
| `--remove-section=*` | Remove the specified section |
| `--strip-all` | Remove all symbols |
| `--strip-debug` | Remove debugging symbols |
| `--strip-dwo` | Remove DWARF .dwo sections |
| `--strip-unneeded` | Remove unused symbols that are not need for relocation processing |
| `--verbose` | List discarded symbols in the terminal |

**NOTE:** See https://sourceware.org/binutils/docs/binutils/strip.html

Created by Devyn Collier Johnson <DevynCJohnson@Gmail.com> (2016) More cheatsheets at DCJTech.info