List of gcc flags with their meanings

---

1. Compilation Stages Control

Flag	Meaning
`-E`	Preprocess only (expand macros, includes)
`-S`	Compile to assembly code only
`-c`	Compile to object file (`.o`), no linking
(no flag)	Compile + link (default behavior)

```
gcc -E file.c
gcc -S file.c
gcc -c file.c
```

---

2. Output & File Control

Flag	Meaning
`-o file`	Specify output file name
`-x language`	Specify language explicitly
`-pipe`	Use pipes instead of temp files

```
gcc main.c -o main
```

---

3. Warning & Error Flags (VERY IMPORTANT)

Flag	Meaning
`-Wall`	Enable most common warnings
`-Wextra`	Enable additional warnings

Flag	Meaning
`-Werror`	Treat warnings as errors
`-pedantic`	Enforce strict ISO C/C++
`-Wshadow`	Warn about variable shadowing
`-Wconversion`	Warn on implicit type conversions

```
gcc -Wall -Wextra -Werror main.c
```

---

4. Debugging Flags

Flag	Meaning
`-g`	Generate debug symbols
`-ggdb`	Debug symbols optimized for GDB
`-fno-omit-frame-pointer`	Better stack traces

```
gcc -g main.c
gdb ./a.out
```

---

5. Optimization Flags

Flag	Meaning
`-O0`	No optimization (default for debugging)
`-O1`	Basic optimization
`-O2`	Standard optimization
`-O3`	Aggressive optimization
`-Os`	Optimize for size (embedded)
`-Ofast`	Unsafe fast optimizations

```
gcc -O2 main.c
gcc -Os firmware.c
```

---

6. Architecture & CPU Flags (Embedded / Performance)

Flag	Meaning
`-march=`	Target specific CPU architecture
`-mcpu=`	Target specific CPU
`-mtune=`	Optimize for CPU without breaking compatibility
`-m32` / `-m64`	Generate 32-bit or 64-bit code

```
gcc -mcpu=cortex-m4 -mthumb main.c
```

---

7. Preprocessor Flags

Flag	Meaning
`-DNAME`	Define macro
`-DNAME=value`	Define macro with value
`-UNAME`	Undefine macro
`-Ipath`	Add include directory

```
gcc -DDEBUG -Iinc main.c
```

---

8. Linking Flags

Flag	Meaning
`-l<lib>`	Link against library
`-L<path>`	Add library search path
`-static`	Static linking
`-shared`	Create shared library
`-Wl,<opt>`	Pass option to linker

```
gcc main.c -lm
gcc -shared -o libtest.so test.c
```

---

9. Security Hardening Flags

Flag	Meaning
`-fstack-protector`	Stack smashing protection
`-fstack-protector-strong`	Stronger protection
`-D_FORTIFY_SOURCE=2`	Buffer overflow checks
`-fPIE`	Position-independent executable
`-pie`	Enable PIE at link time

```
gcc -fstack-protector-strong -D_FORTIFY_SOURCE=2 main.c
```

---

10. Sanitizers (Debugging Bugs 🔥)

Flag	Detects
`-fsanitize=address`	Memory corruption
`-fsanitize=undefined`	Undefined behavior
`-fsanitize=leak`	Memory leaks
`-fsanitize=thread`	Data races

```
gcc -fsanitize=address -g main.c
```

---

11. Language Standards

C

Flag	Standard
`-std=c89`	ANSI C

Flag	Standard
`-std=c99`	C99
`-std=c11`	C11
`-std=c17`	C17

C++

Flag	Standard
`-std=c++11`	C++11
`-std=c++14`	C++14
`-std=c++17`	C++17
`-std=c++20`	C++20

---

12. Code Generation & Behavior

Flag	Meaning
`-fPIC`	Position-independent code
`-ffreestanding`	Embedded / no OS
`-nostdlib`	Do not use standard library
`-nostartfiles`	Skip startup files
`-fno-builtin`	Disable compiler built-ins

---

13. Size & Analysis

Flag	Meaning
`-s`	Strip symbols
`-fdata-sections`	Place data in separate sections
`-ffunction-sections`	Place functions in separate sections
`--gc-sections`	Remove unused code (linker)

```
gcc -ffunction-sections -fdata-sections main.c -Wl,--gc-sections
```

---

14. Useful Inspection Flags

Flag	Meaning
`-v`	Verbose output
`-Q --help=optimizers`	List optimizations
`-dM -E`	List predefined macros
`-ftime-report`	Compilation time report

---

15. Common Flag Combos (Practical)

Debug Build

```
gcc -g -O0 -Wall -Wextra main.c
```

Release Build

```
gcc -O2 -Wall main.c
```

Embedded Firmware

```
gcc -mcpu=cortex-m4 -mthumb -Os -ffreestanding -nostdlib
```

Bug Hunting

```
gcc -fsanitize=address -g main.c
```

GCC Flags Used in Makefiles (Practical Guide)

In Makefiles, GCC flags are **grouped into variables** so they can be reused, modified, and switched easily (debug vs release, host vs embedded).

---

1. Core Makefile Variables (MOST IMPORTANT)

```
CC      = gcc
CFLAGS  =
CPPFLAGS=
LDFLAGS =
LDLIBS  =
```

What each one means 👇

Variable	Used For
`CC`	Compiler
`CFLAGS`	Compilation flags (optimization, warnings)
`CPPFLAGS`	Preprocessor flags (`-D` , `-I`)
`LDFLAGS`	Linker options (`-L` , `-Wl,`)
`LDLIBS`	Libraries (`-lm` , `-lpthread`)

---

2. Typical `CFLAGS` (Compilation Flags)

```
CFLAGS = -Wall -Wextra -Werror -O2 -g
```

Flag	Meaning
`-Wall`	Common warnings
`-Wextra`	Extra warnings
`-Werror`	Warnings → errors
`-O2`	Optimization
`-g`	Debug symbols

📌 **Rule:**

`CFLAGS` → affects `.c → .o`

---

3. `CPPFLAGS` (Preprocessor Flags)

```
CPPFLAGS = -Iinclude -DDEBUG -DVERSION=2
```

Flag	Meaning
`-Iinclude`	Header search path
`-DDEBUG`	Define macro
`-DVERSION=2`	Define macro with value

📌 Used **before compilation**.

---

4. `LDFLAGS` (Linker Options)

```
LDFLAGS = -L/usr/local/lib -Wl,--gc-sections
```

Flag	Meaning
`-Lpath`	Library search path
`-Wl,opt`	Pass option to linker
`--gc-sections`	Remove unused code

📌 Does **NOT** include `-lxxx`

---

5. `LDLIBS` (Libraries)

```
LDLIBS = -lm -lpthread
```

Flag	Meaning
`-lm`	Math library

Flag	Meaning
`-lpthread`	POSIX threads

📌 Linked **last** (order matters).

---

6. Full Minimal Makefile (REALISTIC)

```
CC = gcc

CFLAGS   = -Wall -Wextra -O2
CPPFLAGS = -Iinclude -DDEBUG
LDFLAGS  = -Wl,--gc-sections
LDLIBS   = -lm

TARGET = app
SRC = main.c utils.c
OBJ = $(SRC:.c=.o)

$(TARGET): $(OBJ)
    $(CC) $(OBJ) $(LDFLAGS) $(LDLIBS) -o $@

%.o: %.c
    $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(OBJ) $(TARGET)
```

---

7. Debug vs Release Builds (VERY COMMON)

```
DEBUG ?= 0

ifeq ($(DEBUG),1)
    CFLAGS += -O0 -g
else
    CFLAGS += -O2
endif
```

Build like:

```
make DEBUG=1
make
```

---

8. Embedded / Cross-Compile Makefile Flags

```
CC = arm-none-eabi-gcc

CFLAGS = -mcpu=cortex-m4 -mthumb -Os \
         -ffunction-sections -fdata-sections \
         -Wall

LDFLAGS = -Wl,--gc-sections -T linker.ld
```

📌 **Common embedded-specific flags**

Flag	Meaning
`-mcpu=cortex-m4`	Target MCU
`-mthumb`	Thumb instruction set
`-ffunction-sections`	Per-function section
`-T linker.ld`	Custom linker script

---

9. Automatic Variables Used with Flags

Variable	Meaning
`$@`	Target
`$<`	First dependency
`$^`	All dependencies

```
$(CC) $(CFLAGS) -c $< -o $@
```

---

10. Common Makefile Mistakes (Exam Gold)

❌ Putting `-lm` in `CFLAGS`
❌ Putting `-O2` in `LDFLAGS`
❌ Hardcoding flags inside rules
❌ Wrong order: `-lm` before objects

✅ Correct:

```
$(CC) obj.o -lm
```

---

11. Real Industry Flag Sets

Linux-style

```
CFLAGS = -Wall -Wextra -Wstrict-prototypes -Werror
```

Firmware

```
CFLAGS = -Os -ffreestanding -nostdlib
```

Bug hunting

```
CFLAGS = -fsanitize=address -g
```

---

12. One-Line Mental Model 🧠

```
CPPFLAGS → preprocess
CFLAGS   → compile
LDFLAGS  → link options
LDLIBS   → libraries
```
