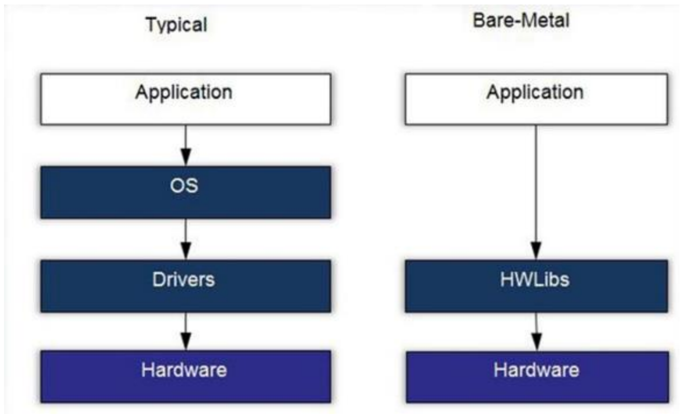
Bare metal Embedded SW

are metal embedded software referring to programming in embedded systems without an operating system (OS) or middleware layers.

This approach involves writing low-level code that interacts directly with the hardware, often in resource-constrained environments.



tool-chain

A **toolchain** in embedded systems development is a set of programming tools used to develop, compile, and debug software for embedded devices.

It typically includes a compiler, assembler, linker, and debugger, along with other utilities.

Native Toolchain

Native toolchain refers to a set of development tools used to compile and build software for the same architecture and operating system on which the development is performed. In other words, the toolchain is designed for the host machine's architecture.

Components:

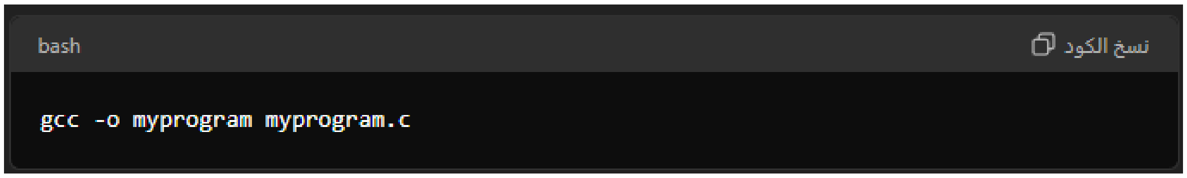
- **Compiler:** Compiles source code into machine code for the host system. For example, gcc for Linux or cl for Windows.
- **Assembler:** Converts assembly language code into machine code.
- **Linker:** Links object files into executables or libraries for the host system.
- **Debugger:** Debugs the application running on the host system.
- **Libraries:** Standard libraries and runtime for the host system.

Example Use Case:

- Developing applications on a Linux PC that runs on an x86 architecture. The toolchain will compile code for the same x86 architecture and Linux OS.

Example Toolchain:

- **GCC (GNU Compiler Collection):** Provides a native compiler for various operating systems and architectures.



```bash
gcc -o myprogram myprogram.c
```

Cross Compiling Toolchain

Cross-compiling toolchain is used to compile code on a host system for a target system with a different architecture or operating system. The toolchain generates binaries for a target platform different from the one used for development.

Components:

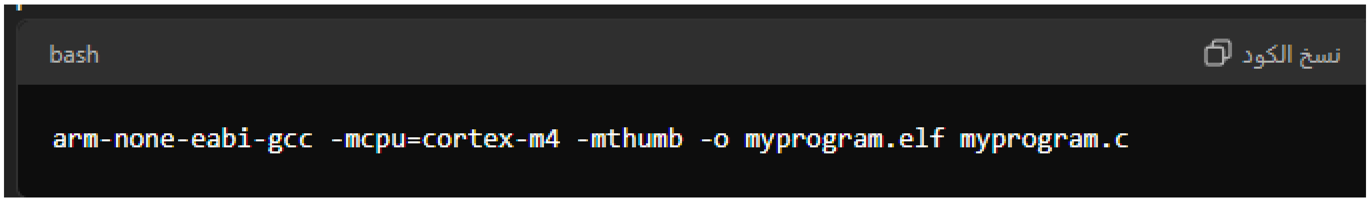
- **Cross-Compiler:** A compiler that generates code for the target architecture. For example, arm-none-eabi-gcc for ARM Cortex-M microcontrollers.
- **Cross-Assembler:** Assembles code for the target architecture.
- **Cross-Linker:** Links object files to create executables for the target system.
- **Cross-Debugger:** Debugs applications running on the target system. For example, gdb with a remote connection.

Example Use Case:

- Developing firmware for an ARM microcontroller on a Linux PC. The cross-compiling toolchain will generate code for the ARM architecture, not x86.

Example Toolchain:

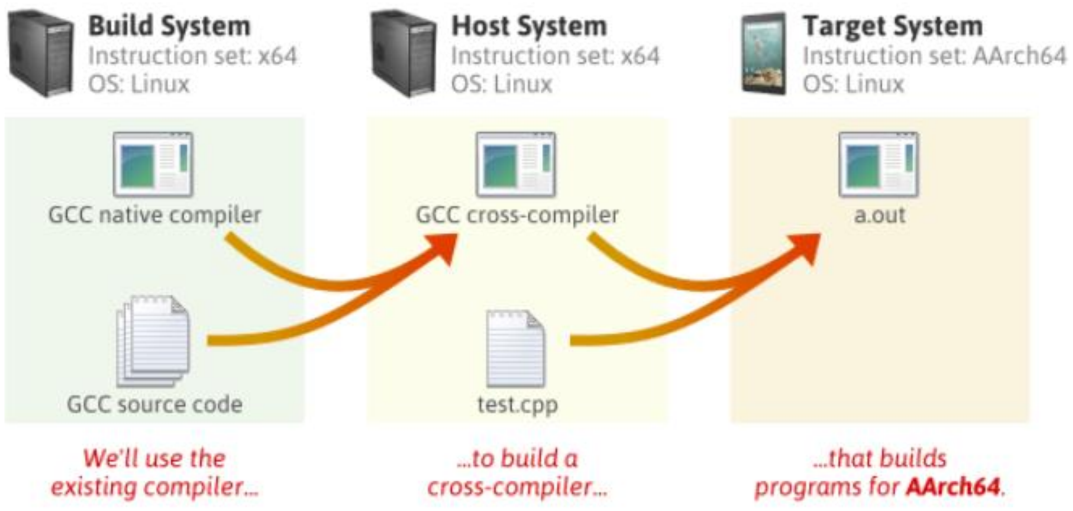
- **GNU Arm Embedded Toolchain:** A cross-compiling toolchain for ARM Cortex-M and Cortex-R processors.



```bash
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -o myprogram.elf myprogram.c
```

Definition

- The build machine, where the toolchain is built.
- The host machine, where the toolchain will be executed.
- The target machine, where the binaries created by the toolchain are executed



Components – GCC

GCC: The GNU Compiler Collection (Table Summary)

Feature	Description	Example
Basic Compilation	Compiles source code (e.g., main.c) into an executable by default named a.out.	gcc main.c
Output Name	Specifies the name of the generated executable file.	gcc main.c -o test (executable named test)
Include Paths	Tells GCC where to search for header files used by the source code.	gcc main.c -I/usr/share/include -I. -I./inc/ -I../../inc
Enabling Warnings	Enables all compiler warnings during the compilation process.	gcc -Wall main.c -o test
Warnings as Errors	Treats all warnings as errors, causing compilation to fail unless warnings are fixed.	gcc -Wall -Werror main.c -o test
Options File	Allows specifying compilation options in a separate text file.	gcc main.c @options-file (options in options-file)

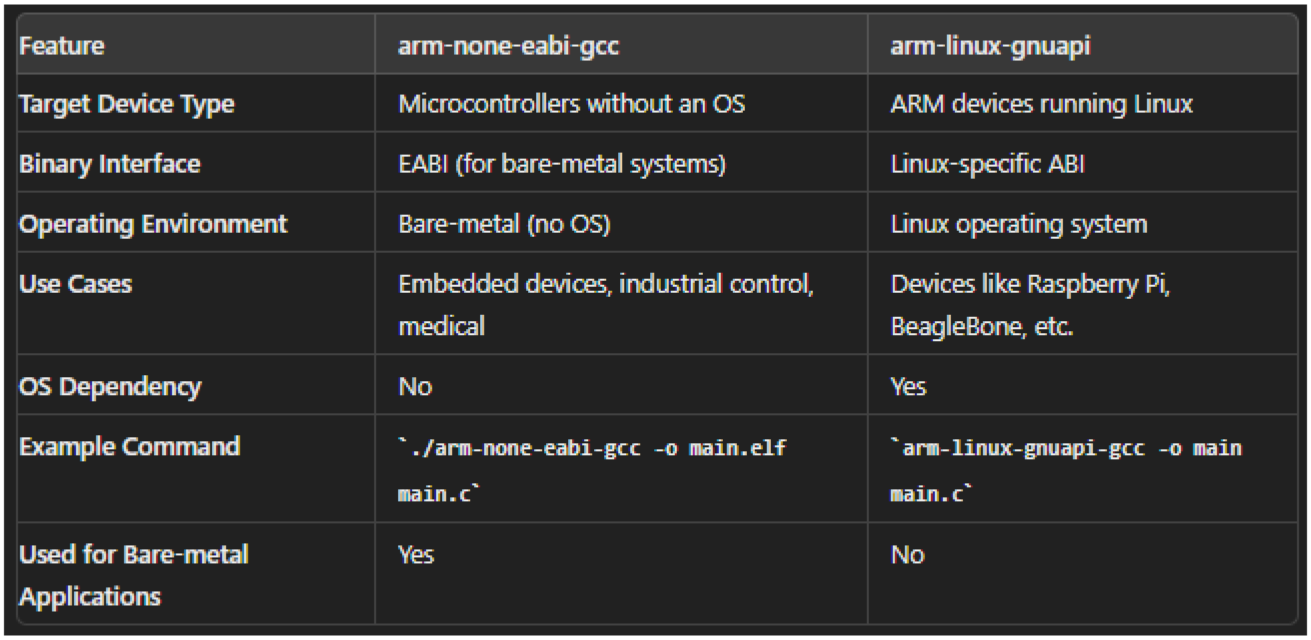
Creating a Static Library (ar Command)

Command (Options)	Description	Example
ar [rcs] library object_files	Creates a new static library (library) or adds object files (object_files) to an existing one.	ar rcs libmylib.a file1.o file2.o (creates libmylib.a from file1.o and file2.o)
ar r library object_files	Adds object files (object_files) to an existing static library (library).	ar r libmylib.a file3.o (adds file3.o to libmylib.a)
ar c library object_files	Creates new members (object files) in an existing static library (library).	(Same as ar r)
ar d library object_files	Deletes members (object files) from a static library (library).	ar d libmylib.a file3.o (removes file3.o from libmylib.a)
ar t library	Displays a table of contents for a static library (library), listing member names.	ar t libmylib.a (shows object files in libmylib.a)
ar x library	Extracts members (object files) from a static library (library) into the current directory.	ar x libmylib.a (extracts all files from libmylib.a)

Key Points:

- r: Creates a new archive or replaces existing members.
- c: Creates new members in an existing archive (same as r).
- s: Creates an archive with symbol table information (often used with r).
- d: Deletes members from an existing archive.
- t: Displays a table of contents for the archive.
- x: Extracts members from the archive.

arm-none-eabi-gcc vs. arm-linux-gnuapi



Feature	arm-none-eabi-gcc	arm-linux-gnuapi
Target Device Type	Microcontrollers without an OS	ARM devices running Linux
Binary Interface	EABI (for bare-metal systems)	Linux-specific ABI
Operating Environment	Bare-metal (no OS)	Linux operating system
Use Cases	Embedded devices, industrial control, medical	Devices like Raspberry Pi, BeagleBone, etc.
OS Dependency	No	Yes
Example Command	`./arm-none-eabi-gcc -o main.elf main.c`	`arm-linux-gnuapi-gcc -o main main.c`
Used for Bare-metal Applications	Yes	No

the meanings of each part of the texts you mentioned:

arm-none-eabi-gcc

- **arm**: Indicates that the compiler is targeted for ARM architecture processors.
- **none**: Signifies that the compiler is intended for bare-metal development, meaning software runs directly on hardware without an operating system.
- **eabi**: Stands for **Embedded Application Binary Interface**, a standard for organizing software on embedded systems to coordinate code and data.
- **gcc**: Stands for GNU Compiler Collection, a suite of compilers supporting various programming languages.

arm-linux-gnuapi

- **arm**: Refers to ARM architecture processors.
- **linux**: Indicates that the compiler is targeted for software running on the Linux operating system.
- **gnu**: Refers to tools that follow the GNU Project, such as compilers and libraries.
- **api**: Stands for **Application Programming Interface**, though in this context, it might be an attempt to indicate GNU-specific API standards. The correct term usually is "gnueabi" rather than "gnuapi."

In summary, arm-none-eabi-gcc is a compiler for developing software for ARM processors in a bare-metal environment, while arm-linux-gnuapi appears to be a somewhat incorrect or incomplete reference to tools for developing software for ARM processors running Linux with GNU tools.