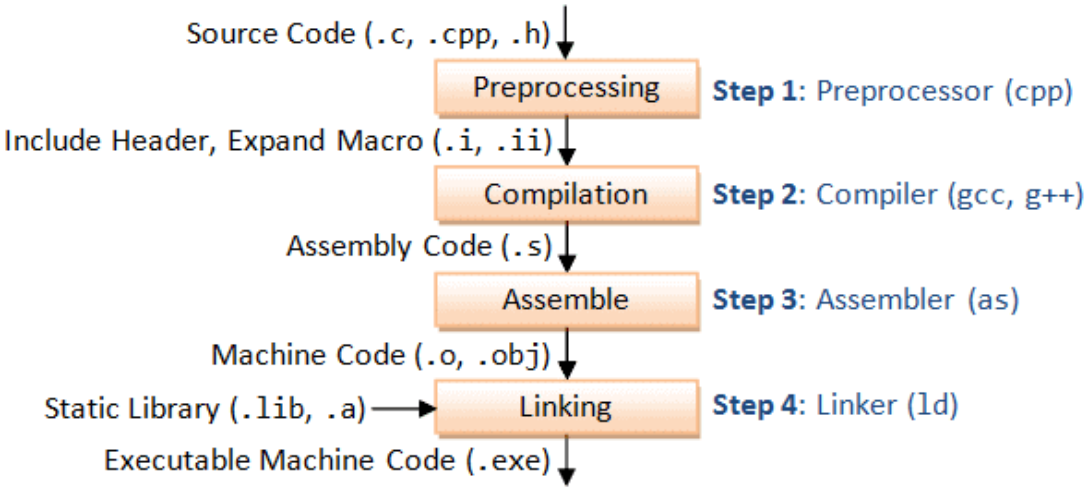
Compilation Process



Preprocessing:

- The preprocessor handles directives like #include, #define, and conditional compilation.
- It expands macros, includes the contents of header files, and processes conditional compilation instructions.
- The output is a preprocessed source file with the .i extension (in C) or .ii extension (in C++).

Example command:

```bash
gcc -E source.c -o source.i
```

Compilation:

- The compiler translates the preprocessed source code into assembly code for the target architecture.
- Syntax and semantic analysis are performed during this stage.
- The output is an assembly file with the .s extension.

Example command:

```bash
gcc -S source.i -o source.s
```

Assembly:

- The assembler converts the assembly code into machine code, generating an object file.
- The output is an object file with the .o (or .obj on Windows) extension.

Example command:

```bash
gcc -c source.s -o source.o
```

Relocatable object files

contain processor architecture-specific machine code with no absolute addresses, allowing for flexible placement in memory during the linking process. They enable the generation of executable files by allowing the linker to adjust addresses and resolve symbols according to the final memory layout of the program.

Linking:

- The linker combines object files and libraries into a single executable.
- It resolves symbol references, assigning addresses to various code and data sections.
- The output is an executable file.

Example command:

```bash
gcc source.o -o executable
```

Additional Details

- **Intermediate Files**: During the compilation process, various intermediate files are generated:
  - Preprocessed source file (.i or .ii).
  - Assembly file (.s).
  - Object file (.o or .obj).

- **Linking Libraries**: The linker can link against standard libraries (like the C standard library) and custom libraries.
  - Static libraries have the .a (Unix/Linux) or .lib (Windows) extension.
  - Dynamic libraries have the .so (Unix/Linux) or .dll (Windows) extension.

Compiler Flags

Compiler Flags: Various flags can be used to control the compilation process:

- -I to specify include directories for header files.
- -L to specify library directories.
- -D to define macros.
- -O to control optimization levels.
- -g to include debugging information.
- -Wall to enable all compiler's warning messages.

table of key gcc flags :

Flag	Description	Result
`-E`	Runs only the preprocessing stage.	Produces a file with the source code after processing preprocessor directives.
`-S`	Runs the compilation stage to generate assembly code.	Produces a file with the assembly code (e.g., `.s` file).
`-c`	Runs preprocessing, compilation, and assembly, but not linking.	Produces an object file (e.g., `.o` or `.obj` file).
`-o`	Specifies the name of the output file.	Determines the name of the file where the compiled code will be stored.
`-l`	Links against specified libraries.	Links the program with the specified library (e.g., `-lm` for the math library).
`-I`	Adds directories to the search path for header files.	Allows the compiler to find header files in the specified directories.
`-L`	Adds directories to the search path for libraries.	Allows the linker to find libraries in the specified directories.
`-D`	Defines macros for the preprocessor.	Defines a macro for the compiler as if it were specified in the source code (e.g., `-DDEBUG`).
`-g`	Includes debugging information in the output file.	Facilitates debugging with tools like `gdb`.
`-O`	Sets the optimization level for the compiler.	Controls the level of optimizations applied to the compiled code (e.g., `-O2`, `-O3`).
`-Wall`	Enables most compiler warning messages.	Helps identify potential issues and warnings in the code.
`-Werror`	Treats warnings as errors.	Causes the compilation to fail if there are warnings.

Compile Time Binding

- is a concept related to how addresses are assigned to code and data during the compilation process.
- if memory location is fixed and known at compile time, absolute code with absolute address can be generated.
- Must Recompile Code if starting location changed.

The. map

The .map file gives a complete listing of all code and data addresses for the final software image.

It provides information similar to the contents of the linker script described earlier.

However, these are results rather than instructions and therefore include the actual lengths of the sections and the names and locations of the public symbols found in the relocatable program.

Link with `.map` File Generation:

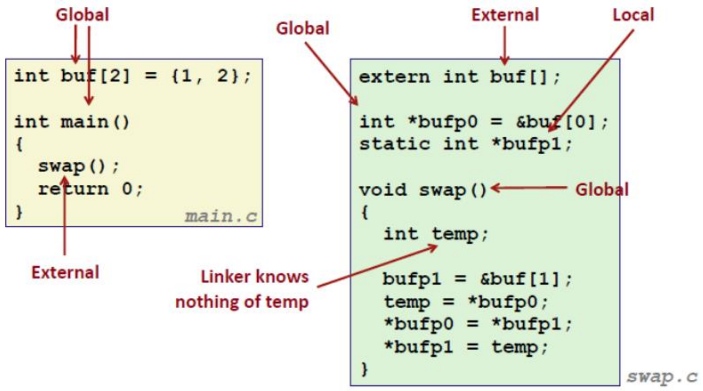
When linking the object files, use the `-Map` option to create a `.map` file. In this example, we will produce an executable file named `main.elf` and a `.map` file named `main.map`:

bash نسخ الكود

```
./arm-none-eabi-gcc main.o -o main.elf -Wl,-Map=main.map
```

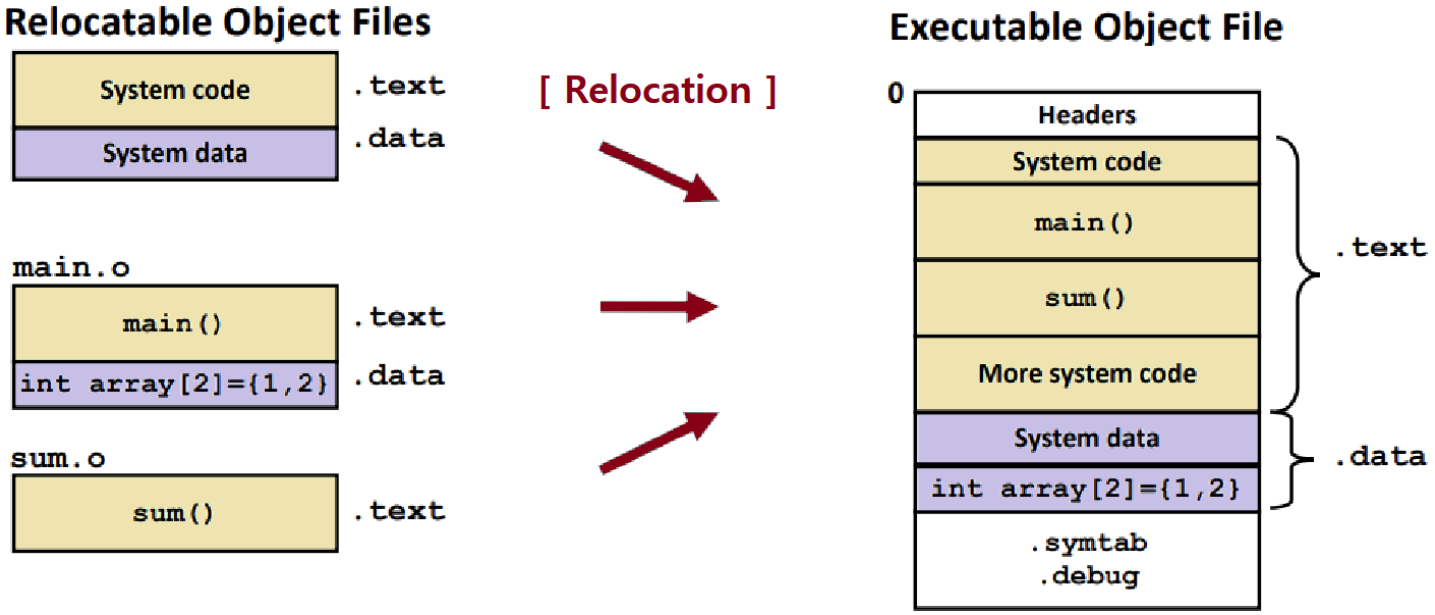
Resolving Symbols During Linking

Resolving Symbols During Linking is the process where the linker connects symbolic references in object files to their actual addresses in the final executable. This involves reading symbol tables, matching references to definitions, and adjusting addresses. This step ensures that all code and data references are correctly linked and that the final program is executable.



Relocating Code and Data During Linking

Relocating Code and Data During Linking involves adjusting addresses in object files so that they correctly fit into the final memory layout of an executable or shared library. This process includes combining object files, assigning final memory addresses, updating references based on relocation entries, and producing the final output with all symbols correctly resolved.



Tool	Purpose	Example Usage
gcc (GNU C Compiler)	Compiles C/C++ (and other languages) source code into object files and links them into executables.	gcc -o myprogram myprogram.c
ld (Linker)	Links object files generated by compilers into executable programs.	ld -o myprogram myprogram.o
make	Reads Makefiles to automate the compilation, linking, and building process of software projects.	make
ar (Archiver)	Creates and manages static libraries by archiving multiple object files into a single library file.	ar rcs libmylib.a file1.o file2.o
readelf	Analyzes and displays information about ELF format executable and object files.	readelf -a myprogram
objdump	Disassembles machine code instructions within ELF format object and executable files.	objdump -d myprogram
nm	Lists symbols (functions, variables) defined within object and executable files.	nm myprogram
strings	Extracts printable strings embedded within binary files.	strings myprogram
strip	Removes unnecessary sections from executables to reduce their file size.	strip myprogram
addr2line	Converts memory addresses from a running program back to the original source code line number.	addr2line -e myprogram 0x08000400
size	Displays the size of each section within an ELF format object or executable file, and the total file size.	size myprogram
gdb (GNU Debugger)	Powerful interactive debugger for analyzing program behavior, setting breakpoints, and inspecting variables.	gdb myprogram
cp (copy)	Copies files and directories from one location to another.	cp /source/file /destination/file
mv (move)	Moves or renames files and directories.	mv /source/file /destination/file
rm (remove)	Deletes files and directories (use with caution!).	rm /path/to/file (**Caution: Use with care!**)

mkdir (make directory)	Creates a new directory.	mkdir newdirectory
rmdir (remove directory)	Removes an empty directory.	rmdir emptydirectory
ls (list)	Lists the contents of a directory. ls -l provides a detailed listing with permissions, owner, and group.	ls or ls -l /path/to/directory
cd (change directory)	Changes the current working directory.	cd /path/to/directory
pwd (print working directory)	Prints the full path of the current working directory.	pwd
cat (concatenate)	Reads the contents of one or more files and displays them on the terminal.	cat /path/to/file
grep (global search regular expression)	Searches for lines in one or more files that match a specified pattern.	grep 'pattern' /path/to/file
less	Reads a file one page at a time, useful for viewing long files.	less /path/to/file
head	Displays the first few lines of a file.	head /path/to/file
tail	Displays the last few lines of a file.	tail /path/to/file
chmod (change mode)	Modifies file permissions to control read, write, and execute access for users, groups, and others.	chmod +x /path/to/file (makes file executable)