Platforms and Architectures

This lesson discusses the support of Go on multiple platforms and architectures.

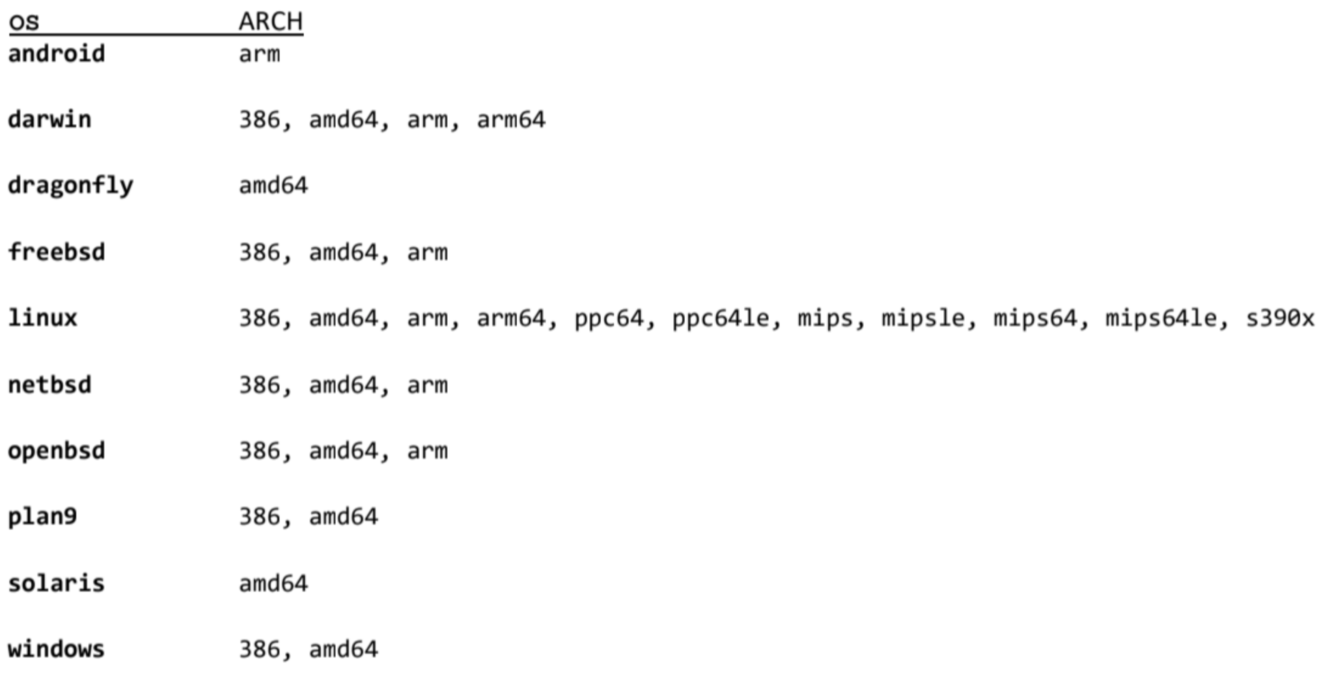
WE'LL COVER THE FOLLOWING

Compatible operating systems

Some architectures were considered to design the compilers for Go according to their supportive OS. The Go-team and the community developed compilers for the following operating systems (OS):

- Linux 2.6.23 or later
- FreeBSD 10.3 or later
- Mac OS (also named Darwin) 10.10 or later
- Windows 7, Server 2008 R2 or later
- ChromeOS
- NetBSD
- Solaris
- Plan9

The following OS-processor combinations (instruction sets) are released:



OS	ARCH
android	arm
darwin	386, amd64, arm, arm64
dragonfly	amd64
freebsd	386, amd64, arm
linux	386, amd64, arm, arm64, ppc64, ppc64le, mips, mipsle, mips64, mips64le, s390x
netbsd	386, amd64, arm
openbsd	386, amd64, arm
plan9	386, amd64
solaris	amd64
windows	386, amd64

OS-Architecture Combinations

The latest Go release at the time of writing was Go 1.11 (released Aug 2018), and work on Go 2.0 is underway.

Flexibility provided by Go

The *portability* of Go-code between OS's is excellent. Assuming you only use pure Go-code, no cgo, inotify, or any low-level packages, then you can copy the source code to the other OS and compile, but you can also cross-compile Go sources, which you will learn in the next lesson. To present a simpler, more unified and OS-independent interface to the developer from Go1 onwards, the compiler, linker, make-scripts, and so on were all brought together in one tool-chain: the **go command**, which is *written in Go* and works in an *OS-independent manner*.

Go compilers

The Go compilers and linkers are written in *C* and produce *native code*; there is no Go bootstrapping or self-hosting involved. It means a different compiler (instruction set) is required for every combination of processor-architecture (32 bit and 64 bit) and OS. The compilers produce state of the art native code and are not linkable with gcc. Furthermore, they work single-pass, non-compacting, and parallel compacting, and parallel.

The sources of the compilers and linkers can be found under:

```
$GOROOT/src/cmd
```

Modifying the Go language itself is done in C code, not Go. The lexer/parser is generated with GNU bison. An overview of the build process can be seen by examining:

```
$GOROOT/src/make.bash
```

Most of the directories contain a file **doc.go**, that provides more information.

File extensions and packages

The extension for Go source code files is, not surprisingly, **.go**. C files end in **.c** and assembly files in **.s**, where source code-files are organized in packages. Compressed files for packages containing executable code have the extension **.a** (AR archive). Linker (object) files can have the extension **.o**. An executable program has the extension **.exe** on Windows by default.

Remember, when creating directories for working with Go or Go-tools, *never use spaces* in the directory name, you can replace space by '_'.

Compilers for Go differ for different platforms to provide full support and maximum flexibility. In the next lesson, you will learn how to install Go on your local machine, if the OS of your device has a Go compiler designed for it.