Cross-compilation

Klaus-Peter Zauner

COMP2215: Computer Systems II
Topic for this and next week:

From Source to Transistor

This week:

▶ From source code to memory

Next week:

▶ From memory to hardware
Topic for this and next week:

From Source to Transistor

This week:

▶ From source code to memory

Next week:

▶ From memory to hardware
Topic for this and next week:

From Source to Transistor

This week:

- From source code to memory
  - Important for understanding C
  - Useful for interpreting error messages

Next week:

- From memory to hardware
Compilation

Source code → Compiler → Executable

Cross-compilation (If a compiler is cross-compiled the meaning of host and target are different.)
Compilation

Source code → Compiler → Executable

Cross-compilation

Source code → Compiler → Executable

(If a compiler is cross-compiled the meaning of host and target are different.)
Cross-compilation

...used when compilation on target is impossible or impracticable:

▶ First compiler on new hardware or new OS
▶ Low capability targets $\rightarrow$ Embedded systems

Architectures of host and target may be very different:

▶ Memory Architecture (von Neumann/Harvard)
▶ Word size (64 bit/8bit)
▶ Order of bytes in word (endianness)
Cross-compilation

... used when compilation on target is impossible or impracticable:

- First compiler on new hardware or new OS
- Low capability targets → Embedded systems

Architectures of host and target may be very different:

- Memory Architecture (von Neumann/Harvard)
- Word size (64 bit/8bit)
- Order of bytes in word (endianness)

...
Build Process

These are cross-compiler, and also cross-linker and cross-assembler.
Build Process: Errors and Options

This previous is still an abstraction—but it is the right level for our purpose:

► Interpreting error messages
► Considering configuration options

There are hundreds of options to steer the build process, the ones of most immediate interest for us are:

► Compiler Optimization
► Compiler Warnings
► Selection of libraries (paths and variants)
Build Process: Intermediate Files

See the Universal Makefile.
Linker Output: .elf object file

Object File Types

- Relocateable (.o)
- Executable (a.out/exe)
- Shared object (.so/dll)

- ELF ➔ Executable and Linkable Format
- Platform independent binary interface for object files
- Contains layout information and sections
Linker Output: .elf object file

Object File Types

- Relocateable (.o)
- Executable (a.out/exe)
- Shared object (.so/dll)

- ELF ➔ Executable and Linkable Format
- Platform independent binary interface for object files
- Contains layout information and sections
Linker Output: .elf object file

Object File Types

- Relocateable (.o)
- Executable (a.out/exe)
- Shared object (.so/dll)

- ELF → Executable and Linkable Format
- Platform independent binary interface for object files
- Contains layout information and sections
## Sections

Memory segments laid out with virtual addresses.

<table>
<thead>
<tr>
<th>Section</th>
<th>Use</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>Instructions</td>
<td>Flash Memory</td>
</tr>
<tr>
<td>.rodata</td>
<td>Read-only data</td>
<td>Flash Memory</td>
</tr>
<tr>
<td>.data</td>
<td>initialised global vars.</td>
<td>Flash copied to RAM</td>
</tr>
<tr>
<td>.bss</td>
<td>Uninitialised global vars.</td>
<td>RAM zeroed out</td>
</tr>
</tbody>
</table>

There are 24 section types defined in ELF, but the ones above are sufficient for us.
Sections II

- **Compiler** assigns program and data to sections
  - Takes into account scope
  - Takes into account qualifier’s like `const`
- **Linker** places the sections to virtual memory
- **Loader** places the sections in physical memory
Translation Units

- C-Source code has three scopes
  1. program wide scope
  2. file scope
  3. block scope (C89: only function scope)

- Programs are organized in files
- Files can be compiled independently (.c → .o)
Use of the ELF sections

11/11
# Use of the ELF sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Use</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>Instructions</td>
<td>Flash Memory</td>
</tr>
<tr>
<td>.rodata</td>
<td>Read-only data</td>
<td>Flash Memory</td>
</tr>
<tr>
<td>.data</td>
<td>initialised global vars.</td>
<td>Flash copied to RAM</td>
</tr>
<tr>
<td>.bss</td>
<td>Uninitialised global vars.</td>
<td>RAM zeroed out</td>
</tr>
</tbody>
</table>