VOF v1 / v1.1 Layout Reference

2,188 words · ~10 min read

Cross-chapter quick-reference handout for CSA-101. Anchors: §6.8 + §6a.2-§6a.4.

Purpose: complete byte-level reference for the **Virtus simplified Object Format v1**, the file format the Ch 6 assembler emits and the Ch 6a linker consumes. Print and pin during Lab 6.3 (VOF emission), Lab 6.4 (assembler-vs-GNU reconciliation), Lab 6a.1-6a.5 (linker pass + relocations + Ghidra forward-pointer). All multi-byte fields are **little-endian** unless explicitly marked.

v1.0 → v1.1 spec migration (2026-04-27). Three coordinated extensions ship together in v1.1:

1. **`NAME_FIELD_BYTES` 16 → 32** (and `SYMBOL_ENTRY_SIZE` 24 → 40). Solves the 16-byte truncation cross-cut surfaced at R4: VM call-protocol names like `vm_<basename>_call_<func>_ret_<n>` always exceed 16 chars; per-file static names typically exceed 16; `Output.printString` is 18 chars. Names > 32 still raise (string-table indirection deferred to CSA-201's ELF migration).
2. **`SECTION_UNDEF = 0x0002`**: a third section value for "extern; defer to linker globals_table". Needed by the `la` pseudo when the assembler doesn't define the symbol locally and must hand it to the linker without inventing a placeholder section.
3. **`R_VIRTUS_LA_GP12 = 2`** relocation kind: 12-bit `gp`-relative offset patched into an `addi rd, gp, <off12>` instruction, pointing into a per-symbol pointer-table entry the linker materialises in `.data`. The CSA-101-specific carve-out for `la rd, sym` in absence of `auipc` / `lui` (see RV32I-Lite encoding card §The 9 pseudo-instructions).

Backward-compatibility direction. Forward-compatible only: every v1.0 symbol-table entry (16-byte names) trivially fits the v1.1 32-byte field. The reverse is not safe. A v1.1 file with names 17-32 chars cannot be rendered to v1.0. v1 readers should reject `version != 0x0001` regardless of the byte layout they encounter; the version field stays `0x0001` since v1.1 is a sub-spec extension, not a major-version bump.

CSA-201 retires VOF in favour of ELF. The conceptual organs (header, sections, symbol table, relocation table) carry over directly; ELF adds richer metadata, multiple section types, a separate string table, dozens of relocation kinds, and several decades of legacy compatibility shims. **VOF is small enough to learn whole; ELF is what every working tool the student will eventually encounter expects.**

At a glance

Property	Value
Magic	`"VOF1"` (4 bytes ASCII; `0x31464F56` little-endian)
Version	`0x0001` (the v1 spec)
Endianness	Little-endian for all multi-byte integers
Header size	**40 bytes** (fixed; unchanged in v1.1)
Symbol entry size	**v1.0:** 24 bytes / **v1.1:** 40 bytes (fixed; name-field grew 16 → 32)
Relocation entry size	**12 bytes** (fixed; unchanged in v1.1)
Sections	`.text` (raw bytes), `.data` (raw bytes), symbol table, relocation table; v1.1 adds `SECTION_UNDEF = 0x0002` for extern symbols
Section ordering	Conventional: `.text`, `.data`, symtab, reltab, but offsets in header are authoritative
Replaces	(nothing, VOF is the first format the student emits)
Replaced by	ELF in CSA-201

File layout

```
+0x00 ┌──────────────────────────────────────────────┐
      │ 40-byte HEADER                               │
+0x28 ├──────────────────────────────────────────────┤
      │ .text section (raw bytes; multiple of 4)     │
      │ └ size = header.text_size                    │
      ├──────────────────────────────────────────────┤
      │ .data section (raw bytes; multiple of 4)     │
      │ └ size = header.data_size                    │
      ├──────────────────────────────────────────────┤
      │ Symbol table (header.symtab_count × 24 bytes)│
      ├──────────────────────────────────────────────┤
      │ Relocation table (header.reltab_count × 12 bytes)
└─────┴──────────────────────────────────────────────┘
EOF
```

The header's `*_offset` fields are the authoritative byte positions of each section within the file. Assemblers may emit sections in any order; readers must use the offsets, not the conventional order.

Header layout (40 bytes)

Offset	Size	Field	Contents
`0x00`	4	`magic`	ASCII `"VOF1"` (`0x31 0x46 0x4F 0x56` byte-order on disk)
`0x04`	2	`version`	`0x0001` (the v1 spec)
`0x06`	2	`flags`	reserved; always `0x0000` in v1
`0x08`	4	`text_offset`	byte offset within file where `.text` begins
`0x0C`	4	`text_size`	size in bytes of `.text` (multiple of 4)
`0x10`	4	`data_offset`	byte offset within file where `.data` begins
`0x14`	4	`data_size`	size in bytes of `.data` (multiple of 4)
`0x18`	4	`symtab_offset`	byte offset within file where the symbol table begins
`0x1C`	4	`symtab_count`	number of entries in the symbol table
`0x20`	4	`reltab_offset`	byte offset within file where the relocation table begins
`0x24`	4	`reltab_count`	number of entries in the relocation table

Magic check. A reader's first job is to verify `magic == "VOF1"`. Any other value is a hard "this is not a VOF v1 file" error. The four-byte magic catches accidental pointing at the wrong file format (ELF, raw text, JSON parse trees, etc.) before any further parsing.

Version check. v1 readers must reject `version != 0x0001` with a clear error. CSA-201 introduces VOF v2 (or transitions directly to ELF. Under discussion); v1 readers do not attempt forward compatibility.

Symbol table entry. V1.0 (24 bytes) / v1.1 (40 bytes)

Each entry describes one label declared in the source file. **v1.1 expands the name field from 16 to 32 bytes; all other field positions shift accordingly.**

v1.1 layout (canonical post-2026-04-27)

Offset	Size	Field	Contents
`+0x00`	32	`name`	null-padded ASCII name; truncates at 32 chars in v1.1 (raises if exceeded)
`+0x20`	2	`section`	`0x0000` = `.text`; `0x0001` = `.data`; **`0x0002` = `SECTION_UNDEF`** (extern; defer to linker)
`+0x22`	2	`binding`	`0x0000` = local; `0x0001` = global
`+0x24`	4	`value`	symbol's offset within its section (ignored when `section == SECTION_UNDEF`)

v1.0 layout (legacy / Ch 6 worked example)

Offset	Size	Field	Contents
`+0x00`	16	`name`	null-padded ASCII name; truncates at 16 chars in v1.0
`+0x10`	2	`section`	`0x0000` = `.text`; `0x0001` = `.data`
`+0x12`	2	`binding`	`0x0000` = local; `0x0001` = global
`+0x14`	4	`value`	symbol's offset within its section

Why v1.1 widened the name field. The 16-byte limit was a deliberate v1.0 pedagogical feature. Small, hex-walkable, every entry the same size. R4.5 surfaced that real CSA-101 code routinely overflows it: VM call-protocol mangled names (`vm_<basename>_call_<func>_ret_<n>`) always exceed 16; per-file static names commonly do; even `Output.printString` is 18 chars. The 32-byte field accommodates every name CSA-101 emits without forcing premature string-table indirection. **CSA-201's ELF migration removes the fixed-size-name constraint entirely via `.strtab`.**

Section field encoding (v1.1). Three values:

- `0x0000` = `.text`
- `0x0001` = `.data`
- `0x0002` = `SECTION_UNDEF` (v1.1 only. Extern symbol; assembler doesn't define it locally; the linker resolves it via globals_table during link). The `la` pseudo emits `SECTION_UNDEF` symbols when the target lives in another translation unit.

Binding field encoding. `local` symbols are file-private; `global` symbols are eligible for cross-file resolution. The `.global SYMBOL` directive in the source promotes a symbol's binding flag in the assembler's symbol table; the VOF emission writes that flag into the binding field. The Ch 6a linker uses the global-binding flag to decide which symbols are eligible for cross-file resolution.

Value field. The symbol's offset *within its section*, not its absolute address. The Ch 6a linker computes absolute addresses by adding the section's base address (from the linker script) to each symbol's value at link time.

Relocation table entry (12 bytes)

Each entry describes one byte position in `.text` (or `.data`) that the linker must patch with a resolved address at link time.

Offset	Size	Field	Contents
`+0x00`	4	`offset`	byte offset within `.text` where the relocation applies
`+0x04`	4	`symbol_index`	index into the symbol table (the symbol whose address is needed)
`+0x08`	4	`kind`	relocation kind. See table below

Relocation kinds

`kind` value	Name (Ch 6a canonical)	Patch-target	Meaning	Spec
`0x00000000`	**`R_VIRTUS_BRANCH13`**	B-format `beq` / `bne` instruction	Patches the 13-bit signed PC-relative offset into the B-format immediate shuffle (bits 31, 30:25, 11:8, 7)	v1.0+
`0x00000001`	**`R_VIRTUS_32`**	Full 32-bit word in `.text` or `.data`	Patches the full 4-byte word with the symbol's final absolute address	v1.0+
`0x00000002`	**`R_VIRTUS_LA_GP12`**	I-format `addi rd, gp, imm12` instruction	Patches the 12-bit signed `gp`-relative immediate to point into the linker-materialised la-pointer-table entry for the symbol; the assembler emits `addi rd, gp, 0` + `lw rd, 0(rd)` and tags the `addi` with this reloc. CSA-101-only carve-out for the `la` pseudo (no `auipc`/`lui` available).	v1.1+

`R_VIRTUS_BRANCH13`, PC-relative branch. The most common relocation kind in CSA-101 single-file work (when used) and the dominant kind in CSA-201's multi-file world.

```
target_address     = global_syms[symbol_index].final_address
pc_relative_offset = target_address - (text_base + offset)        # bytes; signed
imm13              = pc_relative_offset >> 1                      # half-words

# Validate range (B-format reaches ±4 KiB)
if not (-4096 <= pc_relative_offset <= 4094):
    raise LinkerError(f"branch offset 0x{pc_relative_offset:x} exceeds ±4 KiB")

# Read the existing instruction word (assembler emitted with imm = 0)
instr = read_word(text_bytes, offset)

# Patch the B-format immediate shuffle
imm12   = (imm13 >> 12) & 0x1     # → instruction bit 31
imm11   = (imm13 >> 11) & 0x1     # → instruction bit 7   (isolated; most-missed)
imm10_5 = (imm13 >> 5)  & 0x3F    # → instruction bits 30:25
imm4_1  = (imm13 >> 1)  & 0xF     # → instruction bits 11:8

instr |= (imm12 << 31) | (imm10_5 << 25) | (imm4_1 << 8) | (imm11 << 7)
write_word(text_bytes, offset, instr)
```

`R_VIRTUS_32`. Absolute 32-bit address. Patches a full 4-byte word with the target's final address. Used for:

- Cross-file 32-bit address loads (the `.data`-resident pointer trick from Ch 7 §7.8 and Ch 8 §8.6, RV32I-Lite has no `lui`, so 32-bit constants live in `.data` and are loaded via `lw`).
- `.data` and `.bss` references that need an absolute address (e.g., `lw x1, 0(x2)` after `x2` is loaded with the absolute address of a `.data` symbol).
- Linker-materialised la-pointer-table entries: each `R_VIRTUS_LA_GP12` reloc induces one `R_VIRTUS_32` resolution against the underlying symbol when the linker writes the pointer-table word in `.data`.

`R_VIRTUS_LA_GP12`. `la` pseudo expansion (v1.1). The linker maintains a per-binary la-pointer-table in `.data` (one 32-bit slot per distinct la-target symbol). The assembler emits placeholder `addi rd, gp, 0` (tagged with `R_VIRTUS_LA_GP12`) + `lw rd, 0(rd)` for each `la rd, sym`. At link time:

```
ptr_slot_offset = la_pointer_table_offset(sym)         # signed 12-bit offset off
gp
                                                        # (relative to gp's
runtime value)
patched_addi    = encode_addi(rd, gp_reg, ptr_slot_offset)
text_bytes[reloc.offset:reloc.offset+4] = patched_addi.to_bytes(4, "little")
```

The `lw` immediately following the `addi` then dereferences the pointer slot, materialising the symbol's full 32-bit address in `rd`. **Two cycles, no `auipc`, no `lui`, the cost of CSA-101's tight 11-instruction core.** Range check: `ptr_slot_offset` must fit in 12 signed bits (±2 KiB off `gp`), the linker rejects la-pointer tables that would push past the limit, with a CSA-201 forward-pointer suggesting `auipc`+`addi` materialisation as the standard fix.

la-pointer-table / VM segment-pointer collision. la-pointer-table is allocated at `gp+0` by the v1.1 linker, which collides with the VM segment-pointer region (LCL/ARG/THIS/THAT/temp at `gp+0..gp+0x2F` per `cross-chapter-vm-segment-cheat-sheet.md`). The fix is to move the la-pointer-table past `0x40` (after segment pointers) or move la-pointer-table indirection off `gp` to a dedicated register (e.g., `tp`). Tracked in `linker.py` source comments.

```
target_address = global_syms[symbol_index].final_address
patched_word   = target_address                     # full 32-bit absolute
text_bytes[offset:offset+4] = patched_word.to_bytes(4, "little")
```

No bit-shuffle, no PC-relative arithmetic. Just a 4-byte memcpy. The simplest of the two relocation kinds; the workhorse of `la`-based address materialisation.

Note on Ch 6 prose's "I-format immediate" naming

The Ch 6 prose (§6.8.1, p.694) introduces the relocation kind as `0x1 = I-format immediate`. The Ch 6a prose (§6a.3, §6a.4.1-§6a.4.2) refines this to `R_VIRTUS_32` once the linker walks the relocation arithmetic in detail. **The values are identical (`0x00000001`);**

the canonical name is `R_VIRTUS_32`. The "I-format immediate" framing is a Ch 6 pedagogical simplification, the production-grade name applies once cross-file 32-bit address materialisation is the dominant use case, which is the Ch 6a perspective.

When the relocation table is mostly empty (CSA-101)

The CSA-101 single-file regime keeps the relocation table mostly empty: every label reference resolves within the same file, the assembler emits the final encoding directly, and the linker has nothing left to patch. **The relocation table becomes central in CSA-201**, where multi-file linking introduces real cross-module references the assembler cannot resolve and must defer to the linker. *We carry the relocation table in v1 anyway, so the student sees its shape and understands what CSA-201 will use it for.*

The Ch 6 Lab 6.4 fixtures exercise the relocation path explicitly with cross-file label references, simulating multi-file work; the Ch 6a labs build the linker that consumes those relocations.

Cross-format relocation cousin map

Virtus	ELF / RISC-V	Same-shape?	Patch arithmetic
`R_VIRTUS_32`	`R_RISCV_32`	✓ Yes	Identical. Full 32-bit absolute address
`R_VIRTUS_BRANCH13`	`R_RISCV_BRANCH`	✓ Yes	Identical, B-format PC-relative shuffle
`R_VIRTUS_LA_GP12` *(v1.1)*	(no exact ELF/ RISC-V cousin)	✗ No	CSA-101-only carve-out; full RV32I uses `R_RISCV_HI20` + `R_RISCV_LO12_I` (`auipc` + `addi`). The CSA-201 ELF migration retires this reloc entirely.

Two of the three v1.1 relocation kinds are bit-for-bit ELF cousins. The third (`R_VIRTUS_LA_GP12`) has no ELF cousin because it's a CSA-101-specific synthesis required by the absence of U-format. *CSA-201's ELF migration is mechanical for `R_VIRTUS_32` and `R_VIRTUS_BRANCH13`; `R_VIRTUS_LA_GP12` is dropped in favour of standard `auipc` + `addi`.*

Worked example: tiny 2-symbol VOF file (v1.0 layout. Pedagogical brevity)

This worked example uses the v1.0 24-byte symbol entries for compactness; the v1.1 layout is bit-identical except symbol entries grow from 24 to 40 bytes (16 → 32 byte name field), and a third section value `0x0002 = SECTION_UNDEF` is available. Lab 6.3 fixtures should track the v1.1 layout; the worked-example walkthrough below remains pedagogically sound for either spec.

Consider a minimal `.s` source with one global function and one local data label:

```
        .text
        .global _start

_start:
        addi x1, x0, 5          ; 0x00500093
        beq  x1, x0, done       ; B-format branch to done - relocation here
        addi x2, x0, 7          ; 0x00700113
done:
        ret                     ; jalr x0, x1, 0  →  0x00008067

        .data
n_value:
        .word 42
```

The assembler emits this VOF file (392 bytes total. Annotated hex dump):

```
=== HEADER (offset 0x00, 40 bytes) ===
0x00:  56 4F 46 31     magic       = "VOF1"
0x04:  01 00           version     = 1
0x06:  00 00           flags       = 0x0000
0x08:  28 00 00 00     text_offset = 0x0028
0x0C:  10 00 00 00     text_size   = 16 (4 instructions × 4 bytes)
0x10:  38 00 00 00     data_offset = 0x0038
0x14:  04 00 00 00     data_size   = 4 (one .word)
0x18:  3C 00 00 00     symtab_offset = 0x003C
0x1C:  03 00 00 00     symtab_count  = 3 (_start, done, n_value)
0x20:  84 00 00 00     reltab_offset = 0x0084
0x24:  01 00 00 00     reltab_count  = 1 (the beq → done branch)

=== .text (offset 0x28, 16 bytes) ===
0x28:  93 00 50 00     addi x1, x0, 5       ; 0x00500093
0x2C:  63 00 00 00     beq  x1, x0, ???     ; placeholder; reltab patches imm
0x30:  13 01 70 00     addi x2, x0, 7       ; 0x00700113
0x34:  67 80 00 00     jalr x0, x1, 0       ; 0x00008067 (= ret pseudo)

=== .data (offset 0x38, 4 bytes) ===
0x38:  2A 00 00 00     .word 42

=== Symbol table (offset 0x3C, 3 × 24 = 72 bytes) ===
[symbol 0] @ 0x003C
0x3C:  5F 73 74 61 72 74 00 00 00 00 00 00 00 00 00 00     name = "_start" (null-
padded to 16)
0x4C:  00 00                                               section = 0x0000 (.text)
0x4E:  01 00                                               binding = 0x0001
(global)
0x50:  00 00 00 00                                         value   = 0x00000000

[symbol 1] @ 0x0054
0x54:  64 6F 6E 65 00 00 00 00 00 00 00 00 00 00 00 00     name = "done"
0x64:  00 00                                               section = 0x0000 (.text)
0x66:  00 00                                               binding = 0x0000 (local)
0x68:  0C 00 00 00                                         value   = 0x0000000C
(offset 12 in .text)

[symbol 2] @ 0x006C
0x6C:  6E 5F 76 61 6C 75 65 00 00 00 00 00 00 00 00 00     name = "n_value"
0x7C:  01 00                                               section = 0x0001 (.data)
0x7E:  00 00                                               binding = 0x0000 (local)
0x80:  00 00 00 00                                         value   = 0x00000000

=== Relocation table (offset 0x84, 1 × 12 = 12 bytes) ===
[reloc 0] @ 0x0084
0x84:  04 00 00 00                                         offset       =
0x00000004 (the beq instruction in .text)
0x88:  01 00 00 00                                         symbol_index = 1 (done)
```

```
0x8C:  00 00 00 00                                        kind         =
0x00000000 (R_VIRTUS_BRANCH13)

=== EOF @ 0x0090 (total file size = 144 bytes - minimal-example smaller than full
sum_to_n.vof's 392 bytes) ===
```

What the linker does (Ch 6a):

1. Reads the header at offset 0x00; verifies magic + version.
2. Reads the symbol table; populates global namespace with `_start` (global, `.text`+0).
3. Reads the relocation table; finds reloc 0 at `.text+4`, kind=`R_VIRTUS_BRANCH13`, target=symbol[1] (`done`).
4. Looks up `done` → section=`.text`, value=12 → final address = `text_base + 12`.
5. Computes PC-relative offset: `final_address(done) - (text_base + 4)` = `12 - 4` = `+8` bytes.
6. Encodes `+8` into B-format immediate shuffle (`imm13 = 0b0000000001000`), patches the `beq` instruction at `.text+4`.
7. Emits the linked flat image, the student's `program.hex` ready for `$readmemh`.

The student's hand-decoded VOF dump should match the `vof_dump.py` output bit-for-bit. *That match is the chapter's central reproducibility check.*

`vof_dump.py` reference output

Lab 6.3 ships a `vof_dump.py` utility that reads a VOF file and prints a human-readable summary. For the chapter's canonical `sum_to_n.vof` (~392 bytes total):

```
$ python vof_dump.py sum_to_n.vof
=== VOF1 file: sum_to_n.vof (size 392 bytes) ===

Header:
  magic        : "VOF1"
  version      : 1
  flags        : 0x0000
  text section : offset 0x0028, size 48 bytes (12 instructions)
  data section : offset 0x0058, size 8 bytes (2 words)
  symbol table : offset 0x0060, count 6 entries
  relocation t : offset 0x00F0, count 0 entries

.text (48 bytes / 12 instructions):
  0000:  93 80 00 00   addi x1, x0, 0       ; n_value @ symbol[0]
  0004:  13 01 00 00   addi x2, x0, 0
  0008:  93 01 00 00   addi x3, x0, 0
  000c:  ...
  0020:  6f 00 00 ff   beq  x0, x0, -32     ; loop @ symbol[1]
  ...

.data (8 bytes / 2 words):
  0000:  05 00 00 00   .word 5              ; n_value @ symbol[0]
  0004:  00 00 00 00   .word 0              ; result  @ symbol[5]

Symbol table (6 entries):
  [0] n_value      section=.data offset=0x0000 binding=local
  [1] _start       section=.text offset=0x0000 binding=GLOBAL
  [2] loop         section=.text offset=0x000c binding=local
  [3] loop_body    section=.text offset=0x001c binding=local
  [4] done         section=.text offset=0x0024 binding=local
  [5] result       section=.data offset=0x0004 binding=local

Relocation table: empty
```

A student who can read this dump understands what their assembler produced. *A student who cannot has missed something in §6.5 / §6.6 / §6.8 and should return.*

ELF migration map (CSA-201 forward-pointer)

CSA-201 retires VOF in favour of ELF. The conceptual organs map cleanly:

VOF v1	ELF	Notes
`magic = "VOF1"`	`e_ident[0..3] = "\x7FELF"`	Magic location identical (file offset 0); content differs
40-byte fixed header	52-byte `Ehdr` (ELF32)	ELF carries more fields (entry point, machine, type, OS-ABI)
`text_offset` / `text_size` etc.	`Phdr` / `Shdr` (program / section headers)	ELF separates loader-view (`Phdr`) from linker-view (`Shdr`); VOF v1 collapses the two
v1.0: 24-byte symbol entry (16-byte name) / v1.1: 40-byte (32-byte name)	16-byte `Elf32_Sym` (string-table reference)	ELF stores names in a separate `.strtab`; VOF v1 inlines
`section` field (2 bytes; v1.0: 2 values / v1.1: 3 values incl. `SECTION_UNDEF`)	`st_shndx` (section header index; `SHN_UNDEF` is the ELF cousin of `SECTION_UNDEF`)	ELF has dozens of section types
`binding` field (local/global)	`st_info` (binding × type packed)	ELF adds STT_FUNC, STT_OBJECT, STT_SECTION etc.
12-byte relocation entry	8-byte `Elf32_Rel` or 12-byte `Elf32_Rela`	ELF supports addend (`Rela`); VOF v1 has implicit addend = 0
`R_VIRTUS_32`	`R_RISCV_32`	Bit-identical patch arithmetic
`R_VIRTUS_BRANCH13`	`R_RISCV_BRANCH`	Bit-identical patch arithmetic
`R_VIRTUS_LA_GP12` *(v1.1)*	*(no cousin)*	Dropped in CSA-201; replaced by standard `R_RISCV_HI20` + `R_RISCV_LO12_I` once `auipc` + `addi` land

The Ch 6a `virtus2elf.py` adapter (the chapter's "real RISC-V, not a Virtus fork" verifier) converts a VOF file to a minimal ELF file that `riscv32-unknown-elf-ld` accepts as input and `riscv32-unknown-elf-readelf -a` displays as a real ELF object. **The same relocation arithmetic, the same symbol semantics. Just a different wrapper.**

Where to read more

- **Ch 6** *Assembler*. Full VOF emission walkthrough; §6.7.1 + §6.8.1-§6.8.2 (this card distills these).

- **Ch 6a** *Static Linker*. Relocation arithmetic in detail; §6a.3 (symbol resolution), §6a.4.1 (`R_VIRTUS_32`), §6a.4.2 (`R_VIRTUS_BRANCH13`), §6a.4.3 (B-format immediate shuffle walked).
- **Ch 4** *Machine Language*, RV32I-Lite encoding; the encoded bytes that fill `.text` (see RV32I-Lite encoding card).
- **Ch 7 §7.8 / Ch 8 §8.6**. `la` pseudo-instruction and `.data`-resident-pointer trick; the dominant use of `R_VIRTUS_32` once Ch 7 + 8 introduce them.
- **Ch 12 §12.10**. Multi-VOF link in production (OS + application objects together).
- **`vof_dump.py`**, the inspection utility shipped with the Ch 6 reference repo.
- **`virtus2elf.py`**, the Ch 6a adapter that converts VOF → ELF for cross-validation against the GNU toolchain.
- **`man elf` / RISC-V ELF psABI spec** (forward-reference for CSA-201), the production format VOF v1 transitions to.
