Instruction & Data Memory Layout

1,277 words · ~6 min read

Cross-chapter quick-reference handout for CSA-101. Covers the instruction-memory and data-memory partition for the Virtus Console runtime image.

Purpose: complete reference for the instruction-memory and data-memory partitions every Virtus Console program inhabits at runtime. Print and pin during Labs 6a.3 (linker bring-up), 8.2 (function-call protocol emission), 8.4 (gdb stack-walk), 12.5 (capstone Virtus Console). The map below is what `objdump` shows, what `readelf` reports, and what gdb walks. Drift between the map and the running silicon is a curriculum bug; the audit cycle catches and reconciles such drift.

At a glance

Property	Value
`instr_mem` total size	Student-silicon BRAM-backed (Tang Primer 25K canonical Phase-1 baseline ~125 KiB BRAM; Tang Nano 20K advanced-track ~64 KiB practical); ~64 KiB practical baseline per Ch 5 §5.7 + §5.8 sized against the smaller TN20K fabric so any design fits both targets
`data_mem` total size	16 KiB BRAM-backed (4096 × 32-bit words on Ultra96 silicon post-R6B.2)
Partition phases (instr_mem)	3. Synth-time bootstrap / link-time prologue / compile-time user code
Partition regions (data_mem)	4. Segment-pointer slots / la-ptr-table reserve / user-scratch / stack
Address-space style	Flat physical; no MMU; no virtual addresses (CSA-201 adds Sv32 paging)
Endianness	Little-endian throughout
Source-of-truth (linker side)	`linker/prologue.py:40-66` (materialize_const) + `:69-149` (emit_prologue)
Source-of-truth (silicon side)	`peripheral-ip-pack/hdl/.../bootloader.mem` (synth-baked 8-instruction bootstrap)

---

§1. Instr_mem layout (the 3-phase partition)

Three regions; three different sources-of-truth; three different emission times. The student walks through the boundary at each phase in Lab 8.4's gdb session and Lab 12.5's silicon-cert harness.

PC range	Region	Content	Size	Emitted by	Emitted wh
`0x000`. `0x01F`	**Synth-time bootstrap**	8 RV32I-Lite instructions: zero `data_mem[gp+0..0x10]` segment-pointer slots; set `sp` to canonical initial; `jalr` to `0x200`.	32 bytes (0x20)	`peripheral-ip-pack/hdl/.../bootloader.mem` baked via `xpm_memory_*` `MEMORY_INIT_FILE`	At FPGA bitstream synthesis (p bitstream; constant for programs th bitstream ru
`0x020`. `0x1FF`	(synth-time padding)	Zeroed by `xpm_memory` initialisation; reserved for bootstrap growth in CSA-201 (when `.bss` zero-fill + segment-pointer init move into the bootloader).	480 bytes	(zeroed by `xpm_memory`)	At FPGA bitstream synthesis
`0x200`. `0x11FF`	**Link-time prologue**	Per-program `materialize_const` + `sw` for every la-ptr-table entry the linker resolved; final `jalr` to `0x1200`. NOP-padded to fill exactly the 4 KiB reserve.	4096 bytes (0x1000)	`linker/prologue.py:emit_prologue`	At every `python linker.py` invocation (per-program changes per re-link)
`0x1200`, ...	**Compile-time user/OS code**	`Sys.init` runs first (per linker text-section ordering); invokes `Virtus.init` and `Virtus.scheduler`; eventually calls `Main.main`. All compiled-from-Jack via Ch 9-11 compiler chain, plus stdlib service implementations from `stdlib/*.virtus`.	Program-dependent	`compiler.py` + `linker.py` (text region)	At every compile-link cycle (per source chan

Why these boundaries. The bootstrap reserve at `0x000-0x01F` is sized at 32 bytes because that fits the 8-instruction bootloader the FPGA loads via `MEMORY_INIT_FILE` on bitstream load. The bootstrap padding at `0x020-0x1FF` is for bootloader growth without bitstream regeneration, CSA-201 expands the bootloader to handle `.bss` zeroing and

segment-pointer init. The prologue reserve at `0x200–0x11FF` is sized at 4 KiB because that's the worst-case prologue size the reference toolchain has observed (~256 distinct la-pseudo-resolved symbols × ~32 instructions per `materialize_const` before dedup-and-delta; observed typical post-optimisation size ~500-1500 bytes). Padding to 4 KiB gives substantial headroom while keeping user code at the round, memorable address `0x1200` (= `0x200` + `0x1000`).

Cross-chapter: Ch 6a §6a.5.5 walks this from the linker's perspective; Ch 12 §12.10.3 walks it from the runtime image's perspective.

---

§2. Data_mem segmentation (the 4-region partition)

The `gp` (global pointer) register is set by synth-time bootstrap to `0x00010000`. Everything in this section is `gp`-relative.

`gp_offset` range	Absolute range	Region	Contents	Maintained by
`gp+0x00` (`gp+0x10`	`0x00010000`) `0x00010010`	**Segment-pointer slots**	`LCL_addr` (gp+0x00) / `ARG_addr` (gp+0x04) / `THIS_addr` (gp+0x08) / `THAT_addr` (gp+0x0C). Each slot holds a 32-bit pointer at the base of the corresponding VM segment.	Caller's `call` protocol (Ch 8 §8.6); callee's `return` (Ch 8 §8.7); `pop pointer i` from VM source. Bootstrap zeroes slots at boot.
`gp+0x11` (`gp+0x3F`	`0x00010011`) `0x0001003F`	(segment-pointer padding)	Reserved for additional segment pointers in CSA-201 (when OS introduces per-task segment regions).	(no maintainer; reserved space)
`gp+0x40` (`gp+0x3FF`	`0x00010040`) `0x000103FF`	**la-ptr-table reserve** (1 KiB; 256 slots × 4 bytes)	Each slot holds the resolved 32-bit address of one la-pseudo target. Read at runtime by `lw rd, gp_offset(gp)` instructions emitted from `R_VIRTUS_LA_GP12` relocations (Ch 6a §6a.4.6).	Linker prologue (Ch 6a §6a.5.4) populates at runtime via `materialize_const + sw` per slot.
`gp+0x400` (`gp+sp_high`	`0x00010400`) stack	**User / scratch / stack**	Test sentinels (post-R7.2-A2 sentinel-relocation discipline puts test sentinels here at `M[0x400..0x40C]`); user `.data`; the program's stack region growing up	User code (Jack-compiled or hand-written stdlib); allocator's heap (post-R6B.2 silicon expansion); call-protocol stack discipline.

`gp_offset` range	Absolute range	Region	Contents	Maintained by
			from a canonical initial sp address.	

The 12-bit signed offset bound. la-ptr-table slot offsets must fit in 12-bit signed range (`-2048 ≤ gp_offset ≤ 2047`) because the `lw rd, gp_offset(gp)` instruction's I-format immediate field is 12 bits signed. The reserve at `gp+0x40..gp+0x3FF` (= 1 KiB = 256 slots × 4 bytes) sits comfortably within positive 12-bit range; slot capacity is the design's hard ceiling at 256 distinct la-pseudo-resolved symbols. (CSA-201's `auipc + addi` la-pseudo lowering retires this ceiling.)

Cross-chapter: Ch 6a §6a.5.5 specifies the reserve; cross-chapter-vm-segment-cheat-sheet.md walks the segment-pointer slots in detail.

---

§3: Control transfer between phases (worked example)

The full sequence from FPGA reset to first `Main.main` instruction:

```
1. FPGA reset asserts; bitstream loads; `xpm_memory` initialises instr_mem
   from `bootloader.mem` (synth-time bootstrap at 0x000-0x01F).

2. CPU's PC un-gates at 0x000. First instruction fetch:

   PC=0x000:  addi t0, x0, 0
   PC=0x004:  sw   t0, 0(gp)         ← zero LCL_addr
   PC=0x008:  sw   t0, 4(gp)         ← zero ARG_addr
   PC=0x00C:  sw   t0, 8(gp)         ← zero THIS_addr
   PC=0x010:  sw   t0, 12(gp)        ← zero THAT_addr
   PC=0x014:  addi sp, x0, 0x7FC     ← set initial sp
   PC=0x018:  addi t0, x0, 0x200     ← target = prologue entry
   PC=0x01C:  jalr x0, t0, 0         ← jump to PC=0x200

   (Synth-time bootstrap done. Total: 8 instructions, ~8 cycles at 27 MHz.)

3. PC=0x200: linker prologue starts. For each la-ptr-table entry the linker
   resolved, the prologue emits ~24-32 instructions of materialize_const +
   one sw to gp_offset(x0). After all slots populated:

   PC=0x????:  (final materialize_const for 0x1200)
   PC=0x????:  jalr x0, t0, 0         ← jump to PC=0x1200

   (Link-time prologue done. Total: program-dependent, NOP-padded to
   exactly 0x1000 bytes; runs once per silicon cycle.)

4. PC=0x1200: user/OS code starts. Sys.init runs first per linker
   text-section ordering. Sys.init invokes Virtus.init (registers IRQ
   handler addresses), then Virtus.scheduler (cooperative loop:
   poll IRQ, run Main.main, halt).

5. PC=Main.main address: student's compiled application runs.

6. Eventually Main.main returns; Virtus.scheduler returns; Sys.halt
   spins on `beq x0, x0, _halt` until next FPGA reset.
```

Total elapsed time from power-on to `Main.main`'s first instruction: ~100 ms (dominated by FPGA configuration), of which ~30 µs is executable bootstrap + prologue + library initialisation.

Source-of-truth: `linker/prologue.py:40-66` (materialize_const for the `addi`/`add`-only 32-bit constant materialisation; RV32I-Lite has no `lui` per Findings §16) + `linker/prologue.py:69-149` (emit_prologue for the per-slot pattern + 4 KiB-reserve padding).

§4: Debugging tips

Tool	Command	What it shows
`objdump`	`riscv32-unknown-elf-objdump -d program.elf --start-address=0x1200`	Compiled user code starting at 0x1200; skips bootstrap + prologue.
`objdump`	`riscv32-unknown-elf-objdump -d program.elf --start-address=0x200 --stop-address=0x1200`	The linker-emitted prologue (4 KiB; ~500-1500 substantive bytes + NOP padding). Useful for reviewing materialize_const sequences.
`xxd`	`xxd -c 4 program.hex | head -8`	Raw bytes of the bootstrap (8 instructions × 4 bytes = first 8 lines of hex).
`readelf`	`riscv32-unknown-elf-readelf -s program.elf`	Symbol table; check `Sys.init` resolves to an address ≥ `0x1200`; check `Main.main` resolves to higher.
`readelf`	`riscv32-unknown-elf-readelf -r program.elf`	Relocation table; `R_VIRTUS_LA_GP12` entries map to la-ptr-table slot offsets the prologue populates.
gdb (per Lab 8.4)	`(gdb) x/8wx 0x000`	Bootstrap as raw words; should match `bootloader.mem`.
gdb	`(gdb) x/8wx 0x1200`	First 8 words of user code (Sys.init prologue).
gdb	`(gdb) x/64wx 0x10040`	la-ptr-table contents at runtime; each word is a resolved address.
gdb	`(gdb) print/x $gp`	Should be `0x10000` (segment-pointer-region base).

Common confusion: running `objdump` against a fresh ELF with no `--start-address` shows the bootstrap (which is mostly zeros after the first 8 instructions due to the padding), the linker prologue (which looks like a long sequence of `addi` + `add` + `sw` + final `jalr`), and only then user code. Most students expect "compiled code starts at 0x0". It does not on Virtus silicon. Always pass `--start-address=0x1200` for reading user code.

§5: How the 3-phase partition was discovered

The 3-phase partition (synth-time bootstrap / link-time prologue / compile-time user code) differs from what the original chapter outlines projected. The original Ch 12 outline projected a hand-written `crt0.S` source-level bootstrap at `0x000` that included library `init` calls inline. Implementation work revised the model in two structurally important ways: (1) bootstrap moved to a Verilog ROM baked at synthesis time; (2) library `init` calls moved to `Sys.init` running at `0x1200`, with a linker-emitted prologue at `0x200`-`0x11FF` populating the la-ptr-table that all user code's la-pseudos depend on.

An alignment audit found that the 3-phase partition had not been promoted to chapter prose or quick-reference handouts. Ch 6a §6a.5.5 + Ch 12 §12.10.3 and this handout are the write-up.

Pedagogically: students see the audit-then-write-down cadence operational. The chapter's prose is a living document; the handouts are quick-reference distillations of the post-discovery spec; the audit cycle is what reconciles the two.

Where to read more

- **Ch 6a §6a.4.6** *Static Linker*. `R_VIRTUS_LA_GP12` relocation type and the la-pseudo's lowering against the la-ptr-table.
- **Ch 6a §6a.5.4** *Linker Prologue*. Full narrative on the prologue's emission, materialize_const cost-model, dedup + delta-from-previous optimisation, NOP padding to fill exact reserve.
- **Ch 6a §6a.5.5** *Instr_Mem Layout*, the canonical 4-row table this handout's §1 distills.
- **Ch 8 §8.6.2** *Register convention and source-of-truth*, RV32I ABI register classes, caller-clobbered/callee-saved/argument/return; `vm/protocol.py:14-39` ground truth.
- **Ch 12 §12.10.3** *The runtime image*. Runtime-image perspective on the 3-phase partition; cross-references this handout.
- **Ch 12 §12.10.4** *The cooperative scheduler*. `Virtus.scheduler` loop running at `0x1200`; what `Sys.init` invokes after the prologue jumps here.
- **cross-chapter-rv32i-lite-encoding-card.md**, RV32I-Lite encoding card; "Register convention" section enumerates the same convention this handout's §1 references for `gp`.
- **cross-chapter-vm-segment-cheat-sheet.md**, VM segment translation; "Calling-convention diagram" section walks the saved-frame layout that the segment-pointer slots at `gp+0..0x10` mediate.

- **`linker/prologue.py:40-66`**. `materialize_const` ground-truth source.
- **`linker/prologue.py:69-149`**. `emit_prologue` ground-truth source.
- **`peripheral-ip-pack/hdl/.../bootloader.mem`**. Synth-time bootstrap ground-truth source.
