P32W Design Record

Every decision, dated, and what is still to do

P32W — Ken Rother's 32-bit TTL computer

Generated 2026-11-13 from the repository at 3cacc22 2026-11-13

Contents

P32W `README.md`

- The cards
- Layout
- Build and test

P32W design decisions `docs/decisions.md`

- D1 — 32-bit data, 32-bit address, virtual memory (2026-10-11)
- D2 — Custom minimal ISA (2026-10-11)
- D3 — Load/store, 16 registers, fixed 32-bit instructions (2026-10-11)
- D4 — r0 = 0, byte addressing, little-endian, trap on misaligned (2026-10-11)
- D5 — Compare-and-branch, no architectural flags (2026-10-11)
- D6 — Calls push the return address on the stack (2026-10-11)
- D7 — Instruction-level reference emulator first (2026-10-11)
- D8 — ISA v0 accepted (2026-10-11)
- D9 — Goal: a protected, MMU-based multitasking OS, built in stages (2026-10-11)
- D10 — Stage A (traps and privilege) accepted (2026-10-11)
- D11 — Stage B (interrupts and timer) accepted (2026-10-11)
- D12 — Stage C (virtual memory) accepted (2026-10-11)
- D13 — Split instruction/data TLB (amends D12) (2026-10-11)
- D14 — Microarchitecture accepted (2026-10-11)
- D15 — Two stage A details pinned down for the hardware (2026-10-11)
- D16 — The register-file write strobe; the r0 write sink (2026-10-11)
- D17 — The CPU card: clock, GALs, abort, BADADDR, front panel (2026-10-11)
- D18 — Decode traps by a suppressed step; 55 ns flash (2026-10-11)
- D19 — Boards: through-hole only; 4 layers; 0.13 mm clearance (2026-10-12)
- D20 — Interrupt decision timing: TRAPENTER and the interrupt shadow (2026-10-13, ACCEPTED by Ken 2026-10-13)
- D21 — The ALU is a Tang Nano 20K module; every part through-hole (2026-10-13, Ken)
- D22 — P32W Unix: a multi-user, V6-style OS in C (2026-10-13, ACCEPTED by Ken 2026-10-13)
- D23 — A real-time clock on the I/O card (2026-10-14, Ken; the part proposed)
- D24 — A native toolchain is the next milestone (2026-10-14, Ken)
- D25 — Moving files: Kermit, tar, gzip and zip (2026-10-14, Ken)
- D26 — 16-bit CompactFlash transfers (2026-10-14, Ken)
- D27 — BASIC's numbers: integers plus software floating point (2026-10-14, Ken)
- D28 — ustat, and fsck -y on the mounted root (2026-10-14)
- D29 — The C compilers emit functions in definition order; cc frees as it goes (2026-10-14)
- D30 — The C compiler upgrade: C99 pieces, bit-fields, structs by value, long long, double (2026-10-14)
- D31 — switch: a jump table when the cases are dense (2026-10-15)
- D32 — malloc: boundary tags and size bins (2026-10-15)
- D33 — The native cc's memory: freed expansions, shared hidesets, small tokens freed as parsed (2026-10-15)
- D34 — An 8 MiB address space per process (2026-10-15)
- D35 — C and libc pieces for MicroPython (2026-10-15)
- D36 — Python for P32W: MicroPython 1.29 as /usr/bin/python (2026-10-15)
- D37 — Faster program loading, and heap pages made on first use (2026-10-15)
- D38 — The maths library in 62 bits: correctly rounded 99.5% of the time (2026-10-15)

D39 — Python: deeper recursion, cmath, and musl's functions in libc (2026-10-15)

D40 — A 512 MB disk; boot checks it only after a crash (2026-10-16)

D41 — A process's stack may grow to 1 MiB (2026-10-16)

D42 — pow in 96 bits: correctly rounded (2026-10-16)

D43 — USB storage: two CH376S on the I/O card, FAT by tools (2026-10-16)

D44 — The FAT tools: one program, sectors by number (2026-10-17)

D45 — Networking: Ethernet and Wi-Fi through a Pico 2 W and a W5500, before SLIP (2026-10-17)

D46 — Remote logins, FTP and the network clock (2026-10-23)

D47 — Bypass capacitors on a 0.1" footprint, on every board (2026-10-23)

D48 — Board layout rules for every system: the front edge, readable labels, placement drawings (2026-10-24)

D49 — The frame: 9U × 340 mm cards, three connectors, ten slots 1 inch apart (2026-10-24, Ken: "go with 366.7 x 340 and 3 connectors")

D50 — The system bus, the cards, and the choices under them (2026-10-24, Ken: "agree with all your recommendations")

D51 — SWAP, the atomic exchange (2026-10-24; under D50's "ISA kept, plus an atomic swap")

D52 — The address map: every card placed by jumpers, nothing by slot (2026-10-24, Ken: "Address jumpers. Not slot specific")

D53 — The video console (2026-10-24, Ken: "1")

D54 — The bus lock in the microcode (2026-10-24, under D51; Ken: "Go ahead with the microcode")

D55 — Every card designed for the best result; the ISA may change with it (2026-10-25, Ken: "GO with all the Alternatives now open ... I am fine starting as much from scratch as need for the best results")

D56 — The three-bus datapath (2026-10-25, under D55)

D57 — The bus's spare pins: more power, vectored interrupts (2026-10-25, Ken: "Power Pins and Vectored interrupts")

D58 — The CPU card's open choices (2026-10-25, Ken: "the first three, not the trace bus")

D59 — The address adder, indexed and post-increment loads and stores, LI32 (2026-10-25, Ken: "Go with the address adder and the three cheap additions")

D60 — Memory cards may be partly populated (2026-10-26, Ken: "I would like option to not fully populate cards fully"; "for now stick to Static")

D61 — The CPU card as built (2026-10-26, under D55-D59)

D62 — The MMU / data card as built (2026-10-26, under D55; Ken: "continue building hardware")

D63 — The memory card as built (2026-10-26, under D60)

D64 — The debug card: a bus tester that listens or drives, in a slot of its own (2026-10-26, Ken)

D65 — The I/O card as built (2026-10-26, under D50, D52; Ken: "go ahead with the I/O card")

D66 — The disk card as built (2026-10-26, under D26, D50, D52; Ken: "Start the disk card"; "CF-to-IDE adapters")

D67 — The video card as built (2026-10-26, under D50, D52, D53; Ken: "go ahead with the video card"; "Discrete logic")

D68 — The backplane's netlist (2026-10-26, under D49, D50, D64; Ken: "go ahead with the backplane netlist"; ACCEPTED by Ken 2026-10-27: "accept D68 as proposed")

D69 — The power budget (2026-10-27; Ken: "go ahead with the power budget")

D70 — The Tang Nano 20K's footprint and pins from Sipeed's datasheet (2026-10-27; Ken sent "Sipeed Tang nano 20K Datasheet V1.3": "nano foot print")

D71 — The timing pass: a static timing analysis against the datasheets (2026-10-27; Ken: "go ahead with the timing pass then schematics and layout")

D72 — 3 MHz at the datasheets' worst case (2026-10-27; Ken: "3 MHz worst case", "One wait state", "Take writes at cycle end", "Swap the parts (worst case 3 MHz)")

D73 — The debug card's timing mode at 12 MHz (2026-10-27; Ken: "run the debug card at 12 MHz")

D74 — A clock jumper: 4 MHz besides 3 (2026-10-28; Ken: "go ahead with the clock jumper", the timer dividing by 4)

D75 — The disk card's DMA engine, built in now (2026-10-28; Ken: "Although we won't start with the need for DMA i do want it all designed in so new hardware does not need to be ordered"; per-sector pacing, multiword DMA wired; 32-bit buffered cycles)

D76 — Stepping the machine's clock, from the CPU card or the debug card (2026-10-28; Ken: "Design in and also since we have spare bus lines this can travel from debug card to cpu via the bus"; "keep controls on the cards")

D77 — Every board on 6 layers; the first layouts (2026-10-29; Ken: "I am ok going to six layers for all boards")

D78 — A dedicated 5 V supply on screw terminals; a supervisor makes PWROK (2026-10-29; Ken: "I am going to use a dedicated +5V 30A supply ... So I think the bus card just needs heavy duty screw terminals"; agreed to the rest; "assume supply also has +12v or there is a second supply for +12")

D79 — The backplane's board, placed and routed by its generator (2026-10-29; Ken: "go ahead with the backplane layout generator")

D80 — Placement by function; a direction for each routing layer (2026-10-29/30; Ken: "Does it make sense to rearrange chips by functional proximity", "a lot of routers use some planes for mostly horizontal runs and other planes for mostly vertical runs", "consider starting over for each board")

D81 — Schematics for every card, wired where local; the other four layouts (2026-10-30; Ken: "lets generate nice schematics for all cards in kicad and a pdf", "for schematics where appropriate show inter chip connections and not just labels", "lets also start routing of unrouted cards using all the steps and rules for the three that worked")

D82 — The last connections: what more tries, layouts and 8 layers did (2026-10-30; Ken: "keep trying routing, look at possible chip rearrange by function and consider 8 layers if required")

D83 — Freerouting with its two fixes built in; every card routed (2026-10-31; Ken: "do some web research on best layout options and tricks in using the router", "Build with fixes")

D84 — The kernel's sources in /usr/src (2026-11-05; Ken: "go ahead and add the kernel sources to /usr/src")

D85 — A clock counter: `p32uemu -G` and `cycles` (2026-11-05; Ken: "a clock counter, which may need to be 64 bits ... start the counter, do a series of steps, say make the cmd src tree, and stop the counter", "go ahead in that order")

D86 — The kernel builds on P32W itself (2026-11-05; Ken: "I will want to be able to build kernel and for that matter all software on the actual p32w machine", "go ahead with step 1")

D87 — Two speed fixes for `ls -l` (2026-11-05; Ken: "yes do both after the kernel Makefile")

D88 — A profiler for user programs: `p32uemu -q` and `tools/uprof.py` (2026-11-05; Ken: "yes add the profiler after the two fixes")

D89 — Three speed fixes in the C library and `as` (2026-11-05; Ken: "go ahead with those three, score each on the benchmark")

D90 — The boards put right before any order (2026-11-05; Ken: "go ahead and fix the P32W boards, all issues, none have been ordered and are a work in progress")

D91 — Four more speed changes, three kept (2026-11-05; Ken: "go ahead with those four")

D92 — The kernel in the profile, and what a program costs before its own code (2026-11-05; Ken: "yes add the startup profile")

D93 — What every program pays: seven fixes from the startup profile (2026-11-05; Ken: "go ahead with items 1–6 and the process-slot fix")

D94 — A third round: what the tools do for every token and every line (2026-11-06; Ken: "go ahead with the next speed round")

D95 — Separate drill files for plated and non-plated holes (2026-11-06; Ken: "my fab is jlpcb")

D96 — P32W in a GitHub codespace, from a web browser (2026-11-07; Ken: "go ahead with Codespaces, i will be using a chromebook and would like to access via webbrowser")

D97 — The debugger, in a browser tab (2026-11-07; Ken: "I want to develop a web based emulator that can also display graphics and maybe cpu internals"; then: on the host, as YACC1-D's y1dbg, with the screen, the console and debugger, and the CPU's insides)

D98 — The video card again: a 6845, as YACC1-D's (2026-11-10; Ken: "I think the p32w video card is too complicated. Lets go with a card similar to Yacc1 card so we can use a 6845"; then "VGA, 31.5 kHz", "ATmega328P, as YACC1 V2.0", "Text and a graphics layer")

D99 — The backplane: ten slots at 1.5 inches, a front and a back (2026-11-11; Ken: "Drop to 10 slots. Keep 0,1,2 and then only 3-9. Lets move all compenents to one side ... both power entries to that side as well. So the right side is going to be the 'back' ... All cards will be put in with compenents faceing the front or left side. The card numbering sould satart on the left with 0. Lets increasse the space to 1.5\" between cards. Lets also add addirtion mointing support to the top and bottom (more holes)"; "when generarting PNGs generate bottom as well as current top and side")

D100 — The debug card: a two-digit status display, the mode switch turned to the edge (2026-11-11; Ken: "Next up is the debug board, lets get it ready for ordering"; "How about adding 2 TIL-311 Hexadecimal displays near the LEDS. Those could be used for indicating status info. Also turn the listen/drive switch 90 degress so toggle is at the edge of the board (overhangs slighly)"; "the TIL-311 can go just below LEDS. Since this is the front board in a system they will always be visible"; "til-311 is a display ... for better rendering")

D101 — The debug card, ready to order: its three faults put right, the lines DRIVE did not reach, a slide switch (2026-11-11; Ken, to four questions: "slide switch, fix all three, add drive lines, drop both")

D102 — The debug card cannot drive against a card that is in: `-CPUIN`, `-MMUIN` (2026-11-11; Ken: "if debug is on drive mode is there protection thsat it wont drive bus if CPU is?"; "yes add the interlock, both cards"; "go ahead as above")

D103 — The video card on the HD63484 ACRTC, from YACC1-D's V3 design (2026-11-12; Ken: "The video card changes again: base it on YACC1-D's V3 design (the HD63484 ACRTC, VGA) in place of the 6845 revision (D98). I am not building V3 for the YACC1."; "Keep the memory side and the dot path as they are: they are simulated and timed. Design the bus side new for P32W's bus. Write the design note and the list of choices for me before any netlist")

D104 — The ACRTC video card's six choices (2026-11-12; Ken: "agree with all your recommendations")

D105 — The ACRTC video card in the emulators and the kernel (2026-11-12; step 2 of D103's plan, after Ken's "agree with all your recommendations")

D106 — The ACRTC video card's netlist; the picture stays off the bus (2026-11-12; Ken: "leave it off, go ahead with the netlist")

D107 — The ACRTC video card simulated and timed (2026-11-12; Ken: "go ahead with step 4 when the tests pass")

D108 — The ACRTC video card laid out; its timing checked with the board's tracks, and the kept part placed for it (2026-11-12; Ken: "go ahead with step 5 when the tests pass")

D109 — The PS/2 firmware's time in the emulators, at gate level and in the kernel (2026-11-13; Ken: "go ahead with the PS/2 firmware timing")

P32W backlog `docs/backlog.md`

Hardware still to design

Devices: what is left

Operating system and tools

A PDP-11/23 on P32W hardware — analysis `docs/pdp11-analysis.md`

Short answer

Three ways to do it

What already fits

What changes

Software

Speed (an estimate, to be measured)

A path, if taken up

P32W

README.md

P32W: a 32-bit homebrew computer and its operating system. Discrete 74-series logic, GALs, SRAM and flash on **9U × 340 mm cards** with three 96-pin bus connectors, on a 10-slot backplane, with the ALU in an FPGA module (a Tang Nano 20K in sockets: every part is through-hole). A custom minimal load/store ISA with traps, interrupts and virtual memory, and **P32W Unix**, a multi-user V6-style Unix in C, with a native C compiler, assembler and linker, MicroPython, Lua, SQL, BASIC and networking.

The cards: **CPU**, **MMU / data**, **memory** (1–4 × 16 MiB), **disk** (2 CompactFlash + 2 IDE), **I/O** (4 serial ports, timer, clock, 2 USB, Ethernet and Wi-Fi), **video** (VGA text and graphics, PS/2 keyboard and mouse) and **debug**. They meet on a synchronous system bus that carries physical memory cycles; the CPU and the MMU / data card have a private local link besides.

Start here: docs/architecture.md (the machine, the cards, the address map) and docs/programmers-guide.md (how to program it).

- Design decisions: docs/decisions.md
- The operating system: docs/os-design.md; using P32W Unix: docs/unix.md
- Working from a web browser, in a GitHub codespace: .devcontainer/README.md
- The debugger, in a browser tab (the screen, the console, the code, the CPU's insides): tools/p32dbg/README.md
- The backlog: docs/backlog.md
- Instruction set: docs/isa-v0.md; traps, privilege, interrupts, timer and virtual memory (stages A–C): docs/isa-system.md; every instruction in one table: docs/instruction-reference.md (generated)
- Microarchitecture: docs/microarch.md; the CPU card: docs/cpu-card.md; the MMU / data card: docs/mmu-card.md; the memory card: docs/mem-card.md; the I/O card: docs/io-card.md; the disk card: docs/disk-card.md; the video card: docs/video-card-acrtc.md (revision 2, an HD63484 ACRTC; revision 0, never built: docs/video-card.md); the debug card: docs/debug-card.md; each card's theory of operation beside it (`*-card-theory.md`); how it boots: docs/boot.md; the kernel: docs/kernel.md; the filesystem: docs/filesystem.md. `python3 tools/mkpdf.py` makes them PDF books (docs/pdf). The GALs' fuse files and how to program every chip (TL866II+ / T48): hardware/gal/README.md
- ISA additions measured for the new datapath: docs/isa-measurements.md
- The backplane bus: docs/bus.md (generated by `generators/p32bus.py`); the backplane itself: docs/backplane.md; the power budget: docs/power-budget.md; timing: docs/timing.md
- The emulators: docs/emulators.md

The cards

The card netlists come next, the CPU card first (docs/architecture.md, the order of work). Until then the cards exist as the emulators' models (`emulator/p32io.h`, `emulator/p32uemu.c`) and the architecture document.

Layout

Directory	Contents
`microcode/`	`p32isa.py` — the opcode table, single source of truth for the ISA; `p32ucode.py` — the microcode generator

Directory	Contents
`generators/`	`p32bus.py` — the bus (`pinnet()`, `busnet()`); `p32net.py` — device library, netlist helpers, electrical checks; `p32cards.py` — the card list; `p32gal.py` — GAL equations, exhaustive checks, a logic minimiser; `p32sim.py` — netlist-level simulator; `p32schem.py` — schematic PDFs; `p32kicad_sch.py` — KiCad schematic projects; `p32kicad_pcb.py` + `build_board.sh` + `kicad_tools.py` — place, autoroute (Freerouting 2.5) and finish boards; `gen_footprints.py` — the project's footprints (`hardware/lib`); `gen_p32opc.py`, `gen_isaref.py` — the emulator header and the instruction reference
`assembler/`	`p32asm.py` — two-pass assembler, linker, archiver
`cc/`	`p32cc.py` — the C compiler
`emulator/`	`p32emu.c` — instruction-level reference emulator; `p32uemu.c` — microcode-level emulator, cross-checked against it; `p32io.h` — the devices, shared by both; tests in `test/`
`kernel/`	P32W Unix
`software/`	the boot ROM monitor, libc, the commands, MicroPython, Lua, SQL, the tests (software/README.md)
`fpga/alu/`	the ALU in Verilog, its testbench and vector generator
`hardware/lib/`	the project's KiCad footprints (generated); the cards will go beside it
`docs/`	design documents

Build and test

```
cd generators && npm install      # once: elkjs, for the wired KiCad schematic sheets
cd emulator && make test          # everything: emulators, microcode, P32W Unix
```

Assemble and run a program:

```
python3 assembler/p32asm.py prog.asm -o prog.bin -l prog.lst
emulator/p32emu [-t] [-l N] [-d disk.img] prog.bin
```

Tools used: Python 3, a C compiler, KiCad 10 (schematics and boards), Freerouting 2.5 on Java 25 (`~/freerouting/`), Icarus Verilog (the ALU test).

P32W design decisions

docs/decisions.md

This is the log of P32W's design decisions, in the order they were made. Each entry gives its date and, where Ken made the decision, his words; it states the decision, what it settles and what it leaves open. New decisions are added at the end as dated entries.

D1 — 32-bit data, 32-bit address, virtual memory (2026-10-11)

Decision: 32-bit data, 32-bit (virtual) addresses, and virtual memory supported in hardware.

What it settles:

- **Address width:** full 32-bit addressing, not 24-bit.
- **Datapath width:** the data word and data bus are 32 bits. A 16-bit internal datapath with two-cycle ops is possible in principle but would have to sit behind a 32-bit bus; with the ALU in an FPGA (D21) removing the ALU's chip-count argument, the datapath is a full 32 bits.
- **The bus (D50):** 32 data and 32 address bits, not multiplexed.

What virtual memory requires (settled in stages, D9–D13):

- **An MMU** translating virtual to physical addresses. Classic homebrew/TTL options: an SRAM page map indexed by the upper virtual-address bits (plus a process/context ID), a small TLB with page-table walks done by microcode, or a software-managed TLB filled by the OS on a miss.
- **Page size** (e.g. 4 KiB, giving a 20-bit virtual page number).
- **Physical address width** — may be smaller than 32 bits; decides whether the bus address lines carry virtual or physical addresses (normally physical, after the MMU).
- **Page faults with restartable instructions:** the microcode must be able to abort an instruction mid-way, trap to the OS, and re-execute it cleanly — this shapes the microcode sequencer and which registers are updated when.
- **Privilege modes** (at least user/supervisor) and protection bits per page (read/write/execute, user/supervisor), so user programs cannot change the page tables.
- **Exceptions and interrupts** as first-class microcode paths (P8X has IRQ wired but simple).
- **ISA interplay:** RISC-V RV32 defines a virtual-memory scheme (Sv32: 4 KiB pages, two-level page tables, 34-bit physical) and supervisor mode, so choosing RV32 would also choose much of the MMU design; a custom ISA leaves it free (D2).

D2 — Custom minimal ISA (2026-10-11)

Decision: P32W gets its own minimal instruction set, not RV32I or another existing 32-bit ISA.

What it settles:

- **The ISA is custom.** It keeps the P8X lineage: the opcode table lives in the microcode generator and the assembler imports it (one source of truth), and documentation is generated from the same tables.
- **The MMU design (D1) is free:** no Sv32 constraints; page size, page-table format and TLB/page-map style are P32W's own choices.

What "minimal" still has to cover (because of D1):

- **Privileged operations:** enter/leave supervisor mode, return from exception, read/write MMU state (page map or TLB entries, context ID), and control/status registers.
- **Traps:** a system-call instruction, plus fault entry for page faults, protection faults, illegal opcodes and interrupts.
- **Restartability:** every instruction that touches memory must be safe to abort and re-execute after a page fault — simplest when only loads and stores touch memory (load/store architecture).

Costs accepted: no existing compiler or OS. The toolchain is built in-house as on P8X (assembler first; a C compiler later — P8X's p8cc is the starting point, or a retargetable compiler such as LCC or vbcc).

Left open here (settled by D3–D5): register count and width of register specifiers; load/store vs. accumulator/memory-operand style; instruction encoding (fixed 32-bit words vs. variable length); addressing modes; how much is microcoded.

D3 — Load/store, 16 registers, fixed 32-bit instructions (2026-10-11)

Decision:

- **Load/store architecture:** only load and store instructions access memory; ALU instructions operate on registers (and immediates).
- **16 general-purpose 32-bit registers** (4-bit register specifiers).
- **Fixed-length 32-bit instructions.**

Why: with virtual memory (D1) only loads, stores and instruction fetch can page-fault, so restarting a faulted instruction is simple. Fixed 32-bit, 4-byte-aligned instructions never straddle a page boundary, so an instruction fetch faults at most once, and decode is a fixed field split.

What it implies:

- **Encoding budget:** a three-register form (rd, rs1, rs2) uses 12 bits, leaving 20 for opcode and function bits; a two-register-plus-immediate form leaves room for an opcode and a 12–16-bit immediate. A full 32-bit constant takes two instructions (e.g. load-upper-immediate + OR/ADD).
- **Microcode addressing:** the microcode ROM is addressed by an opcode field (6–8 bits), not by the whole instruction register as on P8X (where ROM address = IR | step<<8 | cond<<12). Register fields go straight to the register-file address inputs.
- **Register file:** 16 × 32 bits = 512 bits. In discrete latches that is 64 octal latch chips plus read-port multiplexing — strongly favouring a fast-SRAM register file. A three-register ALU op needs two reads and one write; options are dual SRAM copies (two read ports, writes to both), or two read cycles per operation.

Left open here (settled by D4–D6): register 0 hardwired to zero or not; dedicated stack pointer / link register conventions; byte addressing with byte/halfword loads and stores vs. word-only addressing; exact instruction formats and opcode width; condition codes (a flags register, as on P8X) vs. compare-and-branch.

D4 — r0 = 0, byte addressing, little-endian, trap on misaligned (2026-10-11)

Decision:

- **Register r0 is hardwired to zero:** reads return 0, writes are discarded. r1–r15 are general purpose.

- **Byte addressing:** every byte has its own address; loads and stores come in byte, halfword and word sizes (signed and unsigned byte/halfword loads).
- **Little-endian:** the least significant byte of a word is at the lowest address.
- **Misaligned access traps:** a halfword access to an odd address or a word access to an address not divisible by 4 raises an alignment exception; the hardware never splits an access.

Why:

- r0 = 0 turns whole opcodes into ordinary instructions: move (`ADD rd,rs,r0`), load small constant (`ADDI rd,r0,imm`), clear, NOP, compare with zero, discard a result, absolute addressing (`LOAD rd,addr(r0)`). Keeps the ISA minimal (D2).
- Byte addressing is what C, byte-wide peripherals (UARTs, the CF/IDE task file, PS/2) and byte-oriented file/disk formats expect; pointers stay plain numbers.
- Trapping on misalignment means no access ever spans two words or two pages, so every memory access faults at most once — consistent with simple restart (D1, D3).

Hardware consequences:

- **r0:** register-file writes to register 0 must never land; the reset microcode writes 0 into location 0, and reads then need no extra logic (D16: such writes go to a write sink).
- **Bus:** the bus carries A2–A31 and four byte-enable lines BE0–BE3 in place of A0–A1 (D50).
- **Memory:** organised as four 8-bit byte lanes, each with its own write enable.
- **Load path:** byte/halfword loads need a lane-steering mux to bring the selected byte/halfword to bit 0, plus sign or zero extension.
- **Alignment check:** compares the low address bits with the access size before the bus cycle starts and raises the exception instead.

D5 — Compare-and-branch, no architectural flags (2026-10-11)

Decision: conditional branches compare two registers themselves; there is no programmer-visible flags register.

- Branches: `BEQ`, `BNE`, `BLT`, `BGE` (signed), `BLTU`, `BGEU` (unsigned), each of the form `Bcc rs1, rs2, offset`. Comparisons with zero use r0 (D4); `BGT`/`BLE` are assembler aliases with the operands swapped.
- Set-on-compare: `SLT rd, rs1, rs2` and `SLTU` (rd = 1 if less than, signed / unsigned, else 0) — used for boolean results and for the carry in multi-word arithmetic.
- The ALU still produces C/Z/N/V **internally**, routed to the CPU card's condition mux for the microcode to test (as P8X's FC/FZ/FN/FV bus lines do), but they are not stored as architectural state.

Why: on a microcoded, non-pipelined machine every instruction pays a full fetch + decode, and most conditional branches follow a compare. Folding the compare into the branch removes roughly 5–10% of instructions from typical compiled code. The hardware is the same either way (ALU flag outputs + the condition mux); only programmer visibility differs. No flags means nothing extra to save on traps and context switches, and keeps the ISA minimal (D2).

Costs accepted:

- Multi-word arithmetic is longer: a 64-bit add is 4 instructions (ADD, SLTU for the carry, ADD, ADD) instead of 2 with ADC.
- Signed-overflow detection takes several instructions (no V flag).

- Branch offset is smaller: opcode + two 4-bit register fields leave ~18 bits; scaled by 4 (instructions are word-aligned, D3) that is ±512 KiB. Longer jumps use an unconditional jump/jump-register.
- P8X's carry-flag lesson (active-high carry) applies only inside the ALU and microcode: internal C is active-high, and SLTU/BLTU are defined on it.

Left open here (settled by D8): exact branch encoding and offset width; unconditional jump and call forms and their reach.

D6 — Calls push the return address on the stack (2026-10-11)

Decision: the call instruction pushes the return address onto the stack and jumps; the return instruction pops it into the PC — as P8X's `JSR`/`RTS`. There is no link register.

What it implies:

- **A hardware stack pointer.** CALL/RET (and trap entry/return) use one specific register, so the stack pointer is architectural, not just a software convention. It is one of r1–r15 (r15, below), so ordinary instructions can address stack frames (`LOAD rd, off(sp)`).
- **CALL is a store and RET is a load,** so both can page-fault (D1). The microcode must not change SP or PC until the memory access has succeeded: compute the new SP in a temporary, do the access, then commit SP and PC. Same rule for any other instruction that updates a register and touches memory.
- **Alignment:** SP must stay word-aligned; a misaligned SP traps on the push/pop like any other access (D4).
- **Cost accepted vs. a link register:** every call costs a memory write and every return a memory read, including leaf functions that call nothing (with a link register those never touch memory). In exchange: simpler call sequences, return addresses always where a debugger or stack walker expects them, and the P8X programming model.

Settled with D6 (2026-10-11):

- **SP is r15.**
- **Full-descending stack:** a push decrements SP by 4, then writes; SP points at the last item pushed, so `0(sp)` is the top of stack. (P8X is empty-descending — write, then decrement.)
- **Banked supervisor SP:** in supervisor mode r15 is a separate register (supervisor SP); in user mode it is the user SP. Traps and interrupts push onto the supervisor stack, so an invalid or unmapped user SP cannot cause a second fault on trap entry. Supervisor code needs a privileged way to read and write the user SP (e.g. for context switches).

Left open here (settled by D8): call reach — direct call offset width (PC-relative), and a call-via-register form for function pointers.

D7 — Instruction-level reference emulator first (2026-10-11)

Decision: the first P32W emulator works at the instruction level: it decodes each 32-bit instruction and performs its effect directly, without a microcode model.

Why: a microcode-interpreting emulator (the P8X rule: the emulator runs the real ROM images) needs the whole microarchitecture first — microword, control lines, bus timing, register file. An instruction-level emulator can run programs and test the ISA and assembler now, while ISA mistakes are cheap to fix.

Role: it is the **golden reference model** for the ISA. The microcode-level emulator, the FPGA ALU (D21) and eventually the hardware are checked against it, as P8X's FPGA build is co-simulated against p8xemu. This is a deliberate departure from P8X's "the emulator knows no instructions" rule, which applies to the microcode-level emulator.

Single source of truth kept: the opcode table lives in one Python module; the assembler imports it and a generator emits a C header for the emulator (as P8X's gen_p8xopc.py does for its tables).

v1 scope: flat physical memory (no MMU), supervisor mode only, no interrupts; exceptions (illegal opcode, misaligned access, unmapped address) stop the emulator with a message and a register dump. Instruction formats, opcode map and memory map: see isa-v0.md.

D8 — ISA v0 accepted (2026-10-11)

Decision: isa-v0.md is accepted as the ISA for the reference emulator v1, with all four review points as proposed: 8-bit opcode (branches ±128 KiB), offsets relative to PC + 4, the emulator's v1 memory map (ROM at 0, RAM from 1 MiB, I/O page in the top 32 KiB; the machine's map is D52), and a `NOR` instruction (0x1A) so that `NOT rd,rs` = `NOR rd,rs,r0`. Multiply and divide (0x30–0x37) are included; the FPGA ALU (D21) implements them in hardware.

D9 — Goal: a protected, MMU-based multitasking OS, built in stages (2026-10-11)

Decision: P32W targets a protected multitasking operating system: separate processes, each in its own virtual address space (D1), unable to touch each other's memory, the kernel, or the hardware directly.

Staged plan — each stage is specified, added to the reference emulator and tested (D7) before the next:

Stage	Machine support	What runs
A	Trap/exception mechanism, status register, user/supervisor mode, SYSCALL, return-from-exception, privileged instructions	a small kernel running user-mode programs that request services by system call
B	Timer interrupt (and the interrupt-enable machinery)	preemptive multitasking among tasks in one address space
C	MMU: page tables/translation, per-page protection, page faults	protected processes, each in its own address space

Why staged: each stage is testable on its own, and the trap mechanism of stage A is designed from the start for the page faults and interrupts of stages B and C, so nothing has to be redesigned later.

Stage A proposal: isa-system.md.

D10 — Stage A (traps and privilege) accepted (2026-10-11)

Decision: isa-system.md stage A is accepted with all five review points as proposed:

1. Trap entry writes no memory (EPC, CAUSE, BADADDR and the mode bits are saved in control registers; the handler saves general registers), so trap entry cannot fault.
2. One handler entry point at `VBR`, dispatching on `CAUSE`.
3. `EPC` holds the address of the trapping instruction for every exception, SYSCALL and BREAK included (the handler adds 4 to continue after them).
4. User-mode access to the I/O page traps (PRIVILEGE) already in stage A.
5. In the emulator, `VBR` = 0 means "no handler": an exception stops the emulator with a register dump, as in v1.

D11 — Stage B (interrupts and timer) accepted (2026-10-11)

Decision: isa-system.md stage B is accepted with all five review points as proposed:

1. Interrupts are taken only between instructions; `EPC` = the next (unexecuted) instruction; `CAUSE` = `0x8000_0000 | line`.
2. Eight level-triggered IRQ lines, fixed priority (lowest number first), enabled per line by `IMASK` (CR 7) and globally by `STATUS.IE`; `IPEND` (CR 6) shows the asserted lines.
3. The timer is a memory-mapped device on IRQ0 (`TCOUNT`/`TRELOAD`/`TCTRL` at `0xFFFF_FF10`–`FF18`); in hardware it is on the I/O card (D50, D52).
4. The emulator's timer ticks once per instruction, so interrupt timing is exactly repeatable; hardware ticks from a divided clock.
5. A privileged `WAIT` instruction (0x75) for idle loops.

D12 — Stage C (virtual memory) accepted (2026-10-11)

Decision: isa-system.md stage C is accepted with all six review points as proposed:

1. **Software-managed TLB:** a miss raises an exception and the kernel refills the entry from its own page tables (format chosen by software); no hardware page-table walker.
2. **Translation in user mode only;** supervisor mode uses physical addresses, so the refill handler can never miss.
3. **4 KiB pages; 64-entry direct-mapped TLB** tagged with an **8-bit ASID** (context switches need no flush). `ASID`, `TLBHI`, `TLBLO` (CR 8–10), `TLBW` (0x76); exceptions 8–10 TLB miss on fetch/load/store, 11 PROTECT.
4. **User space is the lower 2 GiB;** user access at or above `0x8000_0000` traps (PRIVILEGE), which includes the stage A I/O-page rule.
5. **Permission bits V/W/X only;** no hardware accessed/dirty bits.
6. **The system bus carries physical addresses** (32-bit physical; RAM up to 64 MiB, D52). Settles the D1 question.

With stages A–C the machine has everything D9 asked for: protected, preemptively multitasked processes in separate address spaces.

D13 — Split instruction/data TLB (amends D12) (2026-10-11)

Decision: the stage C TLB is split into a **64-entry I-TLB** (instruction fetches) and a **64-entry D-TLB** (loads and stores), each direct-mapped and ASID-tagged. A miss or protection fault sets `TLBHI.I` (bit 8) = 1 for a fetch, 0 for a data access; `TLBW` writes the half that `TLBHI.I` selects.

Why: found by the emulator, running the stage A trap test. One instruction can need two translations — its own fetch and its data access. In a single direct-mapped TLB, when the two pages share a slot (VPN mod 64), each refill evicts the other entry and the instruction never completes: a livelock that any program could hit by chance. With the split, every instruction needs at most one entry per half, so it always completes.

Cost: almost none — in hardware both halves are one 128-entry SRAM whose extra address line is the fetch/data signal. A 2-way set-associative TLB would also have fixed it, at the cost of a second SRAM and comparator, replacement logic, and kernel control of the way.

D14 — Microarchitecture accepted (2026-10-11)

Decision: microarch.md is accepted with its review points as proposed:

1. **One shared 32-bit internal data bus,** one driver per microcycle via encoded `DOE`/`DLD` fields decoded where each source and destination sits (P8X style). Between the CPU card and the MMU / data card it runs on the local link (D50); the MMU / data card puts the physical address (A2–A31, BE0–BE3) on the system bus.
2. A 32-word fast-SRAM register file: r0–r15 (15 = USP), EPC, CAUSE, BADADDR, VBR, eight microcode scratch registers T0–T7, and the SSP at 31 (the r15 bank is one gate); r0 by write redirection (D16).
3. PC is a dedicated counter (+4 without the ALU; it holds bits 31:2 only).
4. Microcode in 5 V parallel flash behind a pipeline register (55 ns parts, D18); microaddress = {SEQ, opcode, step, cond}.
5. Precise exceptions: every instruction's microcode commits programmer- visible state only after its last possibly-faulting access (checked by the generator), and a fault aborts the cycle into the trap-entry sequence.
6. A 4 MHz first estimate of the microcycle (≈ 0.6 MIPS), refined on the bench: D17 sets 3 MHz, and D50 designs the bus for 5 MHz.

The cards that carry it are in D50.

Next: the microcode generator and a microcode-level emulator checked against the instruction-level reference emulator (D7).

D15 — Two stage A details pinned down for the hardware (2026-10-11)

Writing the microcode showed two places where isa-system.md left the behaviour open or tied to the reference emulator's internals:

1. `BADADDR` **is 0** for exceptions without an address (illegal and privileged instructions, SYSCALL, BREAK, reserved control register) and for interrupts.
2. `VBR` **and** `EPC` **read back exactly as written;** because the PC holds only bits 31:2 (D14), trap entry and `RTE` ignore their low 2 bits. Both emulators implement these, so their traces can be compared exactly.

D16 — The register-file write strobe; the r0 write sink (2026-10-11)

Decision:

1. `-WSTB`**,** a write strobe on the local link (D50), low late in the datapath phase and rising with `CLK`. Every asynchronous SRAM write on the CPU and MMU / data cards (the register file, the TLB) is gated by it, so no write starts before its address has settled.
2. **r0 by redirection:** a register write whose address comes from an IR field equal to 0 goes to physical register 28 (the write sink), chosen with the register address; physical 0 is written with 0 only by the reset microcode. The register file itself has no r0 logic.
3. **Activity LEDs** (register-file access, PC load) are gated with the real enables.

D17 — The CPU card: clock, GALs, abort, BADADDR, front panel (2026-10-11)

Decision: for the CPU card's control logic:

1. **Clock:** a 15 MHz master oscillator divided by 5: a **3 MHz microcycle** (333 ns), with the control-store lookup in the first 2 ticks (133 ns, `CLK` high) and the datapath in the last 3 (200 ns, `CLK` low); `-WSTB` low during the last tick. Every memory access stays one cycle; ≈ 0.5 MIPS. Why: with the datapath in half of each microcycle, a 4 MHz clock (D14) leaves 125 ns, and the memory path through the TLB needs ~200 ns. 3 MHz is the safe setting; the bus is designed for 5 MHz (D50).
2. **Technology:** 74HCT plus **ATF22V10 GALs** (5 V, DIP-24) for the glue logic. GAL equations are generated from Python and verified there (single source), like everything else.
3. **Abort gating:** while `FAULT` is asserted the CPU card forces `DLD` to NONE, `PCINC` to 0 and the MODE action to none, so no destination latches in a faulting cycle.
4. **BADADDR:** the trap-entry microcode tests a condition **MMUF** (the condition mux's 8th input): BADADDR comes from the MMU / data card for MMU faults and is 0 for traps the CPU card raises itself (D15).
5. **Front panel:** RUN/HALT, STEP (one microcycle; one instruction by running to the next fetch), RESET, activity LEDs.
6. **Decode ROMs** (opcode table, control-register table) in the same flash as the control store, generated from `p32isa.py`.

D18 — Decode traps by a suppressed step; 55 ns flash (2026-10-11)

1. **Illegal/privileged-instruction traps take one suppressed step.** The decision (IR → decode table → trap) cannot be made before the first microword of the instruction (step 2) is looked up within the 133 ns lookup phase. So step 2 runs with all its effects suppressed by the abort gating (D17: no DLD, PCINC, MEM or MODE action), then trap entry follows. Invisible to programs; those two traps take one extra cycle. The microcode-level emulator models it.
2. **55 ns flash** (SST39SF010A-55) for the control store: the condition path from the ALU's compare outputs to the flash address needs ~137 ns with 70 ns parts, over the 133 ns lookup phase.

D19 — Boards: through-hole only; 4 layers; 0.13 mm clearance (2026-10-12)

- **Every component is through-hole** (Ken: important). Footprints are DIP, through-hole sockets, axial/radial passives, THT connectors and modules. `p32kicad_pcb.py` refuses a board with any SMD footprint.
- **4 copper layers by default** (F.Cu signal, In1 GND, In2 VCC, B.Cu signal); **6 where a card needs it** (Ken: "if need be move to 6 layer"; 6 layers cost ~1.5–2× at JLCPCB). Layer count is per card (`LAYERS` in `p32kicad_pcb.py`). The 6-layer stackup: F.Cu signal, In1 GND, In2 signal, In3 signal, In4 VCC, B.Cu signal.
- **Placement:** parts in affinity order (parts sharing nets adjacent), rows snaking, then pairwise swaps of identical footprints while the half-perimeter wire length falls.
- **Freerouting 2.5** (on a private Java 25 runtime): it logs every pass, stops when passes stop improving and saves what it routed.
- **Clearance 0.13 mm** (5 mil, a standard fab tier), tracks 0.2 mm — the P8X backplane's fix. The DIN 41612 pads are 1.55 mm on a 2.54 mm pitch, a 0.99 mm gap; a card using most connector pins must escape two tracks per gap, which needs 1.0 mm at 0.2 / 0.2 and 0.79 mm at 0.2 / 0.13.
- **Tracks 0.2 mm everywhere:** Freerouting 2.5's automatic neck-down is off (it thins tracks to 0.10–0.12 mm to pass between pins).
- `build_board.sh` gates routing on a placement DRC (colliding parts short nets, and Freerouting then searches for hours) and stops Freerouting after `FRTIME` seconds (default 2400).

D20 — Interrupt decision timing: TRAPENTER and the interrupt shadow (2026-10-13, ACCEPTED by Ken 2026-10-13)

Found by simulating the sequencer cycle by cycle running `irq_test`: the sequencer decides "take an interrupt" in an instruction's last microcycle from `IE` and `IPEND & IMASK` as they are during that cycle; the cycle's own writes (STATUS, IMASK) land at its closing edge. Two rules follow:

1. **No interrupt at the end of a TRAPENTER cycle.** At the end of trap or interrupt entry (`MODE = TRAPENTER`, which clears IE at that edge) IE is still 1, so with an interrupt pending the machine would enter *again* and overwrite `EPC` with the handler's address (and an exception taken with interrupts enabled would lose its `EPC` the same way). The sequencer takes `MODE0–2` and never starts an interrupt at the end of a TRAPENTER cycle.
2. **The interrupt shadow (ISA rule, isa-system.md B2):** after `MTCR` or `RTE` the check uses `IE/IMASK` from before the instruction, so one more instruction runs before a pending interrupt is taken. Both emulators do exactly this. (The alternative — computing next-cycle IE and IMASK in the sequencer — needs more sequencer logic; the shadow is the behaviour of many CPUs, e.g. x86 STI, and guarantees progress after RTE.) Verified: `p32uemu -C` models the I/O card's timer exactly (it ticks every third microcycle and reloads one cycle after 0); all cross-checks between the two emulators pass with the rule.

D21 — The ALU is a Tang Nano 20K module; every part through-hole (2026-10-13, Ken)

Ken: "go ahead with the Nano 20K and DIP LVC parts".

1. **The ALU is an FPGA** rather than 74181/74182 logic; the rest of the machine is discrete 74-series logic. So the ALU's function set is not bound to the 74181's: barrel shifts and multiply/divide are single-cycle, as the microcode assumes. The flags are computed inside the FPGA, with the P8X lesson applied from day one: active-high carry and real signed overflow. The ALU is off the critical path; register and bus timing dominate. The emulator stays the golden model and the ALU's Verilog is checked against it.
2. **The FPGA is a Sipeed Tang Nano 20K** (GW2AR-18: 20,736 LUTs, 48 multipliers, 2 PLLs) in two 1×20 female headers on the CPU card. The module brings its own 27 MHz crystal, configuration flash and USB programmer (no JTAG header), and runs from the bus's 5 V. Its pinout is published (Sipeed's pin diagram); the pin map is set with the CPU card.
3. **A 16-bit local bus to the module.** The module has 34 I/O pins; the ALU needs 53 with a 32-bit data path. Eight 74LVC574s split it: four capture `D0–31` at every `CLK` edge and give the FPGA one half at a time; four take the result a half at a time and drive the data bus while `DOE = ALU`. A GAL decodes `-ROE` (so the FPGA cannot drive the bus at all) and registers "this edge loaded A / B" for the FPGA. 31 of the 34 pins are used.
4. **Through-hole 3.3 V logic.** TI's SN74LVC574AN / SN74LVC244AN come in PDIP-20: 3.3 V parts whose inputs (and outputs when off) take 5 V, the only boundary between the 5 V logic and the 3.3 V FPGA. They drive at 3.3 V, above every receiver's TTL threshold (VIH 2.0 V). An LD1117V33 (TO-220) powers them. **No card has a surface-mount part.**
5. **Timing, verified in Verilog with models of the latches** (D17 timing, fields from the pipeline register at `CLK` falling, fclk 108 MHz unrelated to the bus clock): every function's result is on the bus at worst 270 ns after B's edge (the budget is 300 ns; division dominates: read-in ~60 ns, 16 + 2 divider cycles, write-out ~40 ns); sampling at 250 ns fails 1,298 vectors, so the check is real.
6. **Branches take one more microcycle.** The compare flags (`EQ`, `LT`, `LTU`) are ready ~84 ns after B's edge; the sequencer's next-address lookup needs them ~55 ns after it (the pipeline register clocks at `CLK` falling, 133 ns, minus

the 55 ns flash, D18). The six conditional branches test the condition one step after loading B: 4 microcycles not taken, 6 taken.

7. **Still to confirm with the module in hand:** the header row spacing (0.7 in, from Sipeed's drawing) and position, and synthesis timing in Gowin EDA (or Yosys + nextpnr-himbaechel) at 108 MHz — the multiplier is pipelined two stages.

D22 — P32W Unix: a multi-user, V6-style OS in C (2026-10-13, ACCEPTED by Ken 2026-10-13)

Ken: "a multitasking OS where each user is in a protected area with their own memory space … a filesystem, system calls into OS and classic unix like commands." Chosen with Ken: **C, with our own compiler**; **our own kernel in the style of Unix V6**; **real multi-user** (logins, uids, permissions, one terminal per UART); **a Unix inode filesystem** on the CF card. The design and its review points are in os-design.md: the toolchain (relocatable assembler, linker, `p32cc`, a C library), the kernel (processes in private address spaces (D34) with a one-load TLB refill, ~35 V6/V7 system calls, signals, terminal and disk drivers), the filesystem (1 KiB blocks, 64-byte inodes, `p32fs` host tool), users and permissions, the commands, emulator support and tests, and the phases M1–M6. It needs no hardware beyond stages A–C (D10–D12).

D23 — A real-time clock on the I/O card (2026-10-14, Ken; the part proposed)

Ken: "We need to add a realtime clock maybe on IO card then commands and calls to read and set." Without one, P32W Unix would start every boot at 1 January 1970.

1. **The part: a DS12C887** (Maxim / Analog Devices): a 24-pin, 600-mil DIP module with the 32.768 kHz crystal and a lithium cell inside (about ten years), the MC146818 register set of the PC (seconds to century, alarms, periodic and update interrupts, binary or BCD, 12/24-hour) and 114 bytes of battery-backed RAM. Through-hole, 5 V, no extra parts. (To check before ordering: the pin numbers, taken as the MC146818 / DS12887's.)
2. **On the I/O card's byte bus**, PC-style, two byte registers: `FFFF_FF40` the register number (write), `FFFF_FF41` the register (D52). Intel bus mode (`MOT` = GND). A write to `FF40` makes a pulse on the chip's `AS`, and the chip latches the register number from the byte bus as `AS` falls, the data still valid; `FF41` selects the chip, whose `RD` / `WR` are the card's read and write strobes. `-RESET` is the bus `-RES`; the chip's open-drain `-IRQ` drives its interrupt level (**IRQ 5** by default, D52).
3. **Software:** the kernel (`kernel/rtc.c`) keeps it in binary, 24-hour mode, UTC; reads it at boot (the system time is then the clock ticks on top); `stime()` (root) writes it. `date` prints the time and sets it, `date [cc]yymmddhhmm[.ss]` as in Unix V7. A card without the chip is detected (a RAM byte does not read back) and ignored.
4. **The emulators** model the registers (`emulator/p32io.h`): by default the host's clock; `-R SECONDS` starts it at a fixed time that follows emulated time, so the session tests stay repeatable.

D24 — A native toolchain is the next milestone (2026-10-14, Ken)

Ken: "add the native toolchain as the next milestone." P32W Unix gets its own assembler, linker, library tool (`ar`) and C compiler, written in C, so programs can be written and built on the machine itself; the goal is self-hosting (`cc` compiles itself on P32W Unix, identical to the host build). The plan is M7 in os-design.md: binary objects shared with the host tools, a C version of `p32cc` with the same design and ABI (one test suite for both), and register allocation for locals first.

D25 — Moving files: Kermit, tar, gzip and zip (2026-10-14, Ken)

Ken: "We will need a kermit to get files onto the machine, and probably a zip/unzip and a tar command." Done before M7 continues (Ken: "Now, then M7"). All in C under `software/cmd`, all exchangeable with the Mac:

- `tar`: POSIX ustar, `c` / `t` / `x` with `v`, `f`, `z` (through gzip); reads macOS pax headers and GNU long names; keeps modes, owners (by name), times, hard links, devices.
- **Deflate** (RFC 1951) in a shared library, both directions — `gzip` / `gunzip` (RFC 1952; Ken: yes) and `zip` / `unzip` (PKZIP, CRC-32, stored and deflated entries, directories, DOS times), compatible with the Mac's zip / unzip and Finder.
- `kermit`: send, receive and **server mode**, with **long packets and sliding windows**; binary and text files. The Mac side is C-Kermit (Ken installs it); the automated tests use a small Python Kermit in `tools/`. Needs the terminal in raw mode and 1 KiB terminal queues.

Kernel support added for them: `utime` (V7 30) to restore modification times, `times` (43) for CPU time. Checked by a test in `software/tests`: host-made tar, gzip and zip files extracted and made again on P32W in the emulator, then compared on the host (contents, modes, times, links).

D26 — 16-bit CompactFlash transfers (2026-10-14, Ken)

Ken: "yes go to 16 bit cf". CompactFlash is driven 16 bits wide, not in 8-bit True-IDE mode: 8-bit transfers are optional in the CF standard, and many newer and larger cards and most CF-to-SD / SSD adapters lack them; 16 bits is every card's default. It also halves the copy loop per sector.

- **The disk card** (D50): a halfword access to a port's data register (at `0xFFFF_EF00 + 16 p`, D52) is a 16-bit transfer on the drive's `DD0–DD15`. Every other task-file register is a byte.
- **Software:** the monitor's `cf_init` / `cf_read` / `cf_write` and the kernel driver move halfwords (buffers halfword-aligned); SET FEATURES 81h undoes an 8-bit mode left set by another host (a refusal is ignored). The kernel reads a 1 KiB block with one two-sector command.
- **Emulators:** the CF model answers halfword data accesses; byte accesses still model a card in either mode.

D24 done (2026-10-14): M7 complete. The native `as`, `ld`, `ar` and `cc` write byte for byte what the host tools write, and `cc` compiles itself on P32W Unix into a binary identical to the host-built one (os-design.md M7).

D27 — BASIC's numbers: integers plus software floating point (2026-10-14, Ken)

Ken: "integers plus software float for BASIC" (over decimal fixed point). P32W has no FPU, so the floating point is a C library, `softfp` (`<softfp.h>`, `software/libc/softfp.c`):

- **IEEE 754 binary64** (C's `double`): 53-bit significand, about 15-16 digits, exponents to 10^±308, with subnormals, infinities and NaNs. A wider format than the classic BASICs' 32/40-bit floats, chosen so a BASIC program's arithmetic agrees with any other machine's, and so the library could be tested bit for bit against the host's hardware.
- **Correctly rounded** `+ − × ÷` and square root (nearest, ties to even): identical to the host on 2 million random cases. Decimal conversion round-trips exactly at 17 digits. `exp`, `log` and `pow` work internally to 62 bits and are within about 1 ulp; `sin` `cos` `tan` `atan` within 3.
- **The interface takes pointers** (`fp_add(&r, &a, &b)`), so it needs no structures passed or returned by value. Only integer instructions are used; 64-bit quantities are word pairs, products are built from 16-bit pieces.

- `cc/tests/softfp.c` checks that P32W's build prints exactly what the host's does.

BASIC keeps whole numbers that fit in 32 bits as integers (exact and fast) and switches to floating point when a result is not whole or overflows. The C compiler's `double` (D30) is built on this library.

D28 — ustat, and fsck -y on the mounted root (2026-10-14)

From the review of the administration commands (Ken: "lets address all in order sh, kernel, commands, ...").

- **ustat (system call 74):** returns a mounted filesystem's blocks, free blocks, inodes and free inodes from the kernel's in-memory superblock, given the block device's number (st_rdev). df uses it, so any user gets current numbers without reading the root-only devices. mkfs and fsck use it to find out whether a device is mounted. The structure is P32W's own; V7's ustat returned free counts and a name.
- **fsck -y:** repairs, but refuses a mounted filesystem other than /. The root cannot be unmounted, and the kernel's buffer cache holds some of its blocks, which a repair makes stale. So after repairing / fsck restarts at once with `reboot(RB_REBOOT|RB_NOSYNC)`. The flag RB_NOSYNC (0x200) tells the kernel not to write the cache back or mark the filesystems clean. This is what V7's fsck did. Dropping the cache instead would need a kernel call to invalidate one device's buffers, and every open inode on it, so restarting is the simpler choice.

D29 — The C compilers emit functions in definition order; cc frees as it goes (2026-10-14)

From the toolchain step (Ken: "lets address all in order sh, kernel, commands, toolchain and library, documentation").

- **Order:** both compilers (p32cc and cc in C) write functions in the order they are *defined*, not in the order of first declaration (prototypes included). The output is identical between the two (cc_test), and self-hosting reproduces cc byte for byte. Only the layout of `-S` listings and objects is affected.
- **Why:** cc in C generates each function as soon as its body is parsed (a callback, `function_done`). It then frees that function's tree (nodes come from a pool released after each function) and its body's tokens. Closed block scopes and the preprocessor's temporary arrays are freed too, and each macro's output tokens are made once rather than copied two or three times.
- **Result on P32W:** peak memory falls by about half. `parse.c` goes from 3.25 to 1.56 MB, and `sh.c` from out of memory to 1.78 MB. That is about 1 MB per 1000 lines, so files up to about 3000 lines fit in a process.
- **Next, if needed:** run the preprocessor and the parser together, a token at a time, so the whole token list never exists at once.
- **Related:** `volatile` locals are never kept in registers in either compiler, so longjmp cannot undo them.

D30 — The C compiler upgrade: C99 pieces, bit-fields, structs by value, long long, double (2026-10-14)

Ken: "ok go ahead with the compiler upgrade" (the backlog's first step towards Python). Both compilers (p32cc and cc in C) change together and keep writing the same assembly.

1. **C99 pieces:**
   - designated initialisers (`[i] =`, `.m =`, chains like `.a.b[2] =`);
   - brace elision, following C's current-object walk;
   - compound literals: at file scope a static object, in a function a local one initialised where the literal is evaluated;

    - `__func__`. Declarations anywhere, `//`, `_Bool` and `inline` were already in.
2. **Bit-fields** follow the System V layout: from a unit's lowest bit, never across a unit the size of the field's type, and `:0` ends the unit. `op=` and `++` evaluate the place once.
3. **The calling convention for memory values.** A struct, a `long long` and a `double` are *memory values*: handled by their address, never in a register.
    - A function with any memory-value parameter takes **every** argument on the stack, as a variadic one does. Each memory value is copied into as many words as it takes. `va_arg` steps by the size rounded to words.
    - A memory-value result is written where a **hidden first argument** points (the caller's temporary), and that address comes back in r1.
    - Other functions are unchanged: arguments 1–4 in r1–r4.
4. `long long` is 64 bits, aligned to 4 (unlike most ABIs' 8: P32W's alignment never exceeds 4).
    - Its operations are calls of `libc/llong.c`, written on a struct of the same layout.
    - Constants are objects in the data, and constant expressions fold at compile time.
    - 32-bit literals keep their C89 types, so existing programs compile as before.
5. `double` is softfp (D27) behind helper calls (`libc/dbl.c`), so it is correctly rounded and agrees bit for bit with any IEEE machine.
    - `float` **is the same 8-byte type** (C allows this): one implementation, and only more precise than a 4-byte float.
    - Comparisons follow C's NaN rules.
    - Constants fold, except a division by zero or a NaN result, whose bits differ between machines.
    - **printf prints doubles only in programs that pass one.** The float formatting is about 56 KB, too much for every binary. printf calls it through a pointer, `__fp_hook`, which is null in `libc/fpdefault.c`. A compiler that sees a double passed to a variadic function emits a reference to `__printf_fp`, which pulls in `libc/fphook.c`, where the pointer is set. (The linker takes library members in name order, so the default must sort first.) `true` is about 50 KB.
6. `#if` **arithmetic is C99's** (added 2026-10-15): 64-bit intmax_t, uintmax_t when an operand is unsigned, in both compilers. Without it, `-1 < 0u` is true in `#if`, and Lua's limit checks go wrong.

Cost: a `long long` or `double` operation is a library call, so it is tens of times slower than an int one. That is fine for programs that only sometimes need 64-bit or real numbers (Lua, MicroPython, awk-like tools). The 64-bit multiply could use the MULHU instruction later.

D31 — switch: a jump table when the cases are dense (2026-10-15)

Context: both compilers turned every `switch` into a chain of compares, two instructions per case (LI, BEQ). An interpreter's dispatch switch has dozens of cases, so it paid for all of them on every instruction it ran. Lua's main loop spent about 58% of its time there.

Decision: a switch with at least 4 cases, whose values span at most 3 times as many slots as there are cases, becomes a jump table. Other switches keep the compare chain.

- **The dispatch** is a fixed 8 or 9 instructions, whatever the case:

```
addi  d, v, -lo         (or LI + SUB when -lo does not fit 16 bits;
                        none when lo is 0)
li    t, hi-lo+1
bgeu  d, t, default     one unsigned compare: below lo wraps to big
slli  d, d, 2
la    t, table
add   d, d, t
lw    d, 0(d)
jr    d
table: .word case_lo, ..., case_hi   (default for each hole)
```

- **The table sits in .text** right after the JR, so it is read-only and shared, like the code. Each entry is an ordinary ABS32 relocation, so the assembler and linker need no change.
- **Both compilers apply the same rule** and write the same code (all 197 C files identical). The switch value's register is never written, since the native cc may keep a variable in it.

Effect on Lua (software/lua/test/bench.lua, P32W CPU seconds, before and after): fib(25) 240.5 to 161.9; an integer loop 37.2 to 15.9; the whole benchmark 668 to 546. The float, string and sort parts barely change: their time goes to softfp, malloc and the library, not dispatch.

Not done (yet): a binary search for large sparse switches, and splitting a switch into dense clusters. No P32W program has needed either so far.

D32 — malloc: boundary tags and size bins (2026-10-15)

Context: libc's malloc was K&R's: one free list in address order, first fit. Every `free` walked the list to find its place, so a program with thousands of free blocks paid thousands of steps per free, and `malloc` searched the same list. Lua had to bring its own allocator (`lp32_alloc`) to work round it.

Decision: a small dlmalloc-style allocator (`libc/malloc.c`).

- **Boundary tags.** Each chunk starts with two words: the size of the chunk before (valid when that one is free) and its own size with two flags, "in use" and "the one before in use". A free chunk's size is also written at its end. So `free` finds both neighbours and merges with them in O(1).
- **Size bins.** Free chunks wait in doubly linked bins: one per size up to 512 bytes (8-byte steps, so a request takes the first chunk with no search), then one per power of two. A bitmap finds the next bin with chunks. Requests no bin can meet are carved from `top`, the free end of the heap, which `sbrk` extends.
- `realloc` **grows in place** into a free neighbour or into `top`, and gives back the tail when shrinking.
- **The break moved by someone else** (a program calling `sbrk` itself) starts a separate region; the earlier one ends in an in-use "fence" header so nothing merges across the gap.
- Overhead stays 8 bytes per block, blocks stay 8-byte aligned, and the interface is unchanged.

Effect: 100,000 random mallocs and frees over 10,000 slots: 288 million P32W instructions before, 24 million after. Lua uses the standard allocator (realloc and free) and is faster with it than with its own: its benchmark 552 s to 494 s, a full GC of 10,000 tables 53.9 s to 19.3 s. `cc/tests/malloc.c` checks it under load (random sizes, fills and checks every block, realloc both ways, calloc, the break moved behind its back).

D33 — The native cc's memory: freed expansions, shared hidesets, small tokens freed as parsed (2026-10-15)

Context: the native cc ran out of memory on five of Lua's files, so Lua could not be rebuilt on P32W. D29 had assumed the cost was the token list; measuring showed most of it was waste.

- **Macro expansion threw tokens away without freeing them:** the macro's name, a call's parentheses and commas, its arguments (only copies are used), `##`'s operands, directive lines and skipped `#if 0` text. For `lvm.c` (53,000 tokens out) there were 191,000 tokens alive.
- **Every expanded token had its own copy of its hideset:** 287,000 list cells.
- **One function can be huge.** `luaV_execute` is 35,500 tokens and 16,000 tree nodes, and its tokens could not be freed while its tree pointed at them.

Decision (cc in C only; p32cc on the host is unchanged, and the two still write the same code for all 223 C files):

1. **Expansion owns its tokens:** each one goes to the output or is freed. `#define`, `#if` and computed `#include` work on copies, and `process` frees every directive line.
2. **Hidesets are shared:** a (name, rest-of-list) cell is made once and found again in a table. When preprocessing ends, the macros and the hidesets are freed.
3. **Tokens are 28 bytes** (the flags are bit-fields, and a character's value shares space with a string's bytes), from 4 KB slabs without a malloc header each. Each slab counts its live tokens, and a slab with none goes back to the heap.
4. **Tree nodes keep a position, not a token:** one shared token per source line, which is all error messages and `-S` listings use. So the parser frees each statement's tokens as soon as it is parsed (string literals stay for the data). Node blocks are 4 KB too, so they reuse the slabs that come back.
5. **The generated lines** waiting for a function's prologue are packed into 4 KB chunks rather than one allocation each.

Result on P32W: all of Lua compiles on P32W, and the output is the same as the host's. `lvm.c` takes 2.99 MB of heap, and `lstrlib.c` 1.23 MB (2.4 MB before). The limit is about 3.1 MB, so `lvm.c` is close to it. If a bigger single function appears, the next step is a smaller tree node (60 bytes, of which `struct vec list` is 12) or streaming the preprocessor into the parser (D29).

D34 — An 8 MiB address space per process (2026-10-15)

From Ken, with the per-process window sized to the RAM a big program needs.

- **8 MiB per process.** `USERTOP` is `0x0080_0000`. A page table is 2,048 entries in **two physically contiguous frames**, so the TLB-miss fast path is one indexed load. `pt_new` finds two adjacent free frames from the frames' reference counts, a scan that happens only at fork and exec. If memory is too fragmented for that, fork fails with ENOMEM. A two-level table would avoid the contiguity at the cost of one more load per refill.
- **The RAM itself is set by D52:** 16 MiB memory cards, up to four (64 MiB). One card gives 15 MiB above the boot ROM. The kernel's frame table, `sysinfo` and `/dev/mem` follow the RAM it finds at boot.
- `db`'s backtrace bound, the man pages and the docs say 8 MiB.

Why 8 MiB: big programs (the native cc on big files, Lua, MicroPython) are the ones that need room, and 8 MiB covers them. One process can take a large share of a one-card machine's RAM; swapping (backlog) is the answer to that, not a smaller window.

D35 — C and libc pieces for MicroPython (2026-10-15)

Porting MicroPython 1.29 ("Python for P32W", stage 2) found gaps in both C compilers and in libc. Each is fixed in both compilers, which still write the same code for all 228 C files, and each has a test in cc/tests.

Compilers.

- `_Bool` **is C99's.** As a plain `unsigned char`, `bool b = 0x100;` would store 0, the low byte. Any value converted to it, by assignment, initialiser, argument, return or cast, is 1 if it is not zero (or not a null pointer), else 0. *(bool.c)*
- `offsetof` **is an integer constant.** `(size_t)&((T *)0)->m` folds to a number when it is cast. So it works in array sizes, `case` labels, enum values and static initialisers, including the static assertions MicroPython builds from it. *(decl99.c)*
- `restrict` **is accepted, and ignored.** *(decl99.c)*
- **A parameter or local may be named like a typedef** (`qstr qstr`): after the specifiers, an identifier is the declarator's name. *(decl99.c)*
- **A flexible array member given elements in a static initialiser** (a GCC extension) makes the object run on past `sizeof`. Dropping its elements silently left MicroPython's qstr pools reading garbage. *(flexarray.c)*
- `volatile` **after a** `*` **(**`T *volatile p`**) makes** `p` **volatile.** So do `volatile` parameters, and each declarator in a list gets its own volatility. Volatile variables are never kept in registers, so `longjmp` cannot undo a change to them. MicroPython's VM keeps its exception-handler pointer like that; without this, `finally` blocks are skipped. *(volatile.c)*
- **Bit-fields wider than 32 bits** (a GCC extension on `uint64_t`) are a clear error, rather than laid out and given bad code.
- **Hexadecimal floating constants** (`0x1.8p3`, `0x1p-1022`, with an `f` or `l` suffix): the `p` exponent is required, so `0x1f` is still 31. p32cc reads them with `float.fromhex`; the native cc with `strtod`, and P32W libc's `strtod` already reads them exactly. *(c99lit.c)*
- **Designators reach into anonymous structs and unions**: `.lo = 1` where `lo` belongs to an unnamed member names the path to it, and initialisation carries on from there. *(c99lit.c)*
- `__builtin_nan("")` **and** `__builtin_inf()` are double constants, so `NAN` and `INFINITY` (defined with them) work in static initialisers. *(c99lit.c)*

libc.

- `<stdint.h>`, `<stdbool.h>`; `SSIZE_MAX` **in** `<limits.h>`; `nan()` (with `nanf` and `nanl`).
- **More** `errno` **names, with Linux's numbers:** `EWOULDBLOCK` (= `EAGAIN`), `EOPNOTSUPP` 95, and networking's `EADDRINUSE` 98, `ECONNABORTED` 103, `ECONNRESET` 104, `ENOBUFS` 105, `ENOTCONN` 107, `ETIMEDOUT` 110, `ECONNREFUSED` 111, `EHOSTUNREACH` 113, `EALREADY` 114, `EINPROGRESS` 115, with `strerror` messages. MicroPython's `errno` module lists them, and the network stack (D45) returns them.
- `pow` **is correctly rounded for whole exponents up to ±2047.** Its double-double path is renormalised after each product, so it does not overflow past ±64. A denormal result is rounded once, ties to even. Without this, about 30% of large powers were 1-3 ulps off, and MicroPython's float literals such as `1e301` parsed wrong. Checked against exact arithmetic: 0 wrong of 15,127 (every exponent from -1080 to 1080, seven bases, 138 denormal results).

Emulator. `-w` also knows Python's prompts `>>>` and `...`, so scripted sessions can drive the REPL.

D36 — Python for P32W: MicroPython 1.29 as /usr/bin/python (2026-10-15)

From Ken: "go ahead with the python port" (stage 2 of the backlog's "Python for P32W"; D35 was the C it needed).

What. MicroPython 1.29.0 (the release's source distribution, without its submodules: `software/micropython/README.P32W`) is `/usr/bin/python`, also named `micropython`, with python(1). It runs a file, `-c` code or the standard input, or gives a REPL at a terminal (line editing, history, Tab completion, auto-indent). `sys.path` is the current directory, then `/usr/lib/micropython`; `sys.argv` is as CPython's (`['-c', args...]` for `-c`).

Configuration (the port's `mpconfigport.h`). MicroPython's "extra features" level, as its Unix port, with arbitrary-size integers (mpz) and IEEE double floats. Left out: `select` and `asyncio` (P32W has no `poll`), `framebuf`, and `cmath` (a candidate to turn on; D39 does). Added: `re`'s `match.groups/start/end/span`, and `os.getenv`, `putenv`, `unsetenv` and `system` (the port's `modos.c`, as the Unix port has them). Python objects live in a fixed 2 MiB heap; Python frames on a 64 KiB stack of their own; C may use 448 KiB of the process's 512 KiB stack, which stops a recursion about 700 calls deep with a `RuntimeError` (D39 and D41 deepen both).

Built on the host, by `software/Makefile` running the port's `build.sh`: MicroPython's own scripts make the generated tables (the sources preprocessed by the host's clang with only P32W's headers), then p32cc compiles all 134 files (about a minute). The same sources give the same 1.7 MB binary. Built on P32W with the native cc: D39.

Changes to imported files are few and marked "P32W:" (README.P32W lists them): the double's 64-bit bit-fields taken with shifts and masks (D35), and 4-byte floats. P32W's C `float` is the 8-byte double (D30), so `struct`/`array` `'f'`, half-precision `'e'` and `uctypes.FLOAT32` would be 8 bytes, or read back as 0. They go through two libc functions in `<math.h>`, `__f32_bits(double)` (rounded to nearest, ties to even) and `__f32_value(bits)` (exact): the conversions Lua's `string.pack("f")` already uses.

The reference. `build.sh` with `HOST=1` builds the same MicroPython, same configuration, for the Mac. Its output is what P32W's must match, since MicroPython's own expected results are CPython's. It is built with `-ffp-contract=off`: clang otherwise fuses `a*b + c` into one rounding, which changes `math.gamma(5.5)` in the last digit, while P32W rounds each step as C says. Where P32W and the Mac still differ, P32W's libm is the cause: `atan2(2, 1)` is 0.58 ulp off where the Mac's is correctly rounded, so complex powers can differ in the last digit (D38 settles it).

Tested.

- MicroPython's own `tests/basics` and `tests/float`, 645 programs, run on P32W and on the reference: all 645 outputs identical (610 run to the end; 23 skip themselves for features left out; 12 stop the same way on both, needing t-strings or `unittest`). This found the 4-byte float bug, and a shell bug: a `for` list over more than 512 names was cut short without a word. `sh` says "too many names" (status 2) for that and for wildcards past its 16 KiB of names (`/usr/tests/sh/glob.sh`).
- `/usr/tests/cmd/python.sh` on P32W: the language, the library, files and `os`, and the command's options and exit statuses. Its three scripts' output is the reference's.
- `session15` in `make test`: the REPL at a terminal (blocks, an error, Tab completion, ^D).

Costs. Starting `python` took about 12 s of P32W time, nearly all of it the kernel reading the 1.7 MB program (D37 brings it to 3.1 s). A Python function call then takes about 2 ms, and a loop step 0.35 ms.

D37 — Faster program loading, and heap pages made on first use (2026-10-15)

From Ken: "go ahead with 1, 2 and 3", 1 being the backlog's "starting `python` takes 12 s". A profiler showed where the time went: `p32emu -k FILE` writes how many instructions ran at each kernel address (and in user mode), and `tools/kprof.py FILE` totals them by function. For three starts of `python -c pass`, 50 million kernel instructions: `wakeup` 23%, the CF loop 21%, `memset` 15%, `memcpy` 12%, the buffer cache's lookup 8%. Little of it was the disk itself.

What changed (kernel; 12 s → 3.1 s for `python`, about 4 → 1.9 s for `lua`; all tests unchanged):

- `exec` **reads the program straight into its pages.** `readi_pages` (fs.c) loads a file's bytes into a page table's pages. Runs of up to 64 consecutive blocks that are not in the buffer cache go through an optional block-device entry, `readv` (`cf_readv`). It reads them in multi-sector commands, past the cache. Its scatter list of pieces copes with the a.out header's 32 bytes, which put every block across a page boundary. A block that is in the cache comes from there, since it may be newer than the disk (a program just compiled). The snapshot of the cached blocks holds because the file is locked. The leaf indirect block is kept while its 256 entries are walked, not read again for each block.
- **The CF transfer loop** is a small function of its own (`cf_in`, `cf_out`), so its two variables live in registers: 19 instructions per 16 bytes, against 22 per 8 with them in memory. Every disk read and write gains.
- **Pages the program fills whole are not zeroed first** (`map_page_raw`). A short file fails the `exec` before it can run, so no stale data shows.
- `brelse` **wakes waiters for a free buffer only when there are any** (rather than scanning all 64 process slots for every block).
- **Heap pages are made on first use.** `sbrk` promises pages without mapping them; a fault, or `copyin`/`copyout`, maps a zeroed frame then. Each promise holds a frame in reserve (`nreserved`), so `sbrk` still fails when memory is short, and a page it granted is always there later: no overcommit. `fork` reserves the child's untouched pages too; exit, exec and shrinking give back what was never touched. `ptrace` reads an untouched page as zeros. MicroPython's 2 MiB heap costs only the pages it uses.

What is left is mostly the CF loop itself: 1.7 MB through a 16-bit data register, about 1.2 instructions a byte. Only DMA would remove it (backlog, "Faster disk transfers").

D38 — The maths library in 62 bits: correctly rounded 99.5% of the time (2026-10-15)

From Ken's "go ahead with 1, 2 and 3", 3 including "make `atan2` correctly rounded". Measured first: against the Mac's libm, P32W's `sin` and `cos` differed in about 22% of results, `atan2` 37%, `asin` 63% (16% by more than an ulp). `exp` and `log` agreed 99%, because they were already worked out in softfp's unpacked form, which carries 62 bits, and rounded once. The others were plain double arithmetic, rounding at every step.

What changed (`libc/softfp.c`, `libc/fpmath.c`). The rest of the library works as `exp` and `log` do: in 62 bits, rounded once.

- `sin`, `cos`, `tan`: the argument is reduced by k·π/2 with π/2 in four parts (three of 33 bits, so k times each is exact), then a series to 13 terms. Arguments past 2^19 are reduced exactly first (Payne-Hanek).
- `atan`: past 1, use 1/t; then atan t = atan(j/8) + atan((t − j/8)/(1

    - t·j/8)), the nine atan(j/8) held to 106 bits as pairs of doubles; the series on the rest (< 1/16). `atan2`, `asin`, `acos` build on it, with sqrt((1 − a)(1 + a)) from exact factors.
- `expm1`, `log1p`: their own series for small arguments, so there is no cancellation. `sinh`, `cosh`, `tanh`, `log2`, `log10`, `cbrt` (one Newton step) follow, and `asinh`, `acosh`, `atanh` are added.
- `exp`**'s reduction** uses ln 2 in two parts (a single 62-bit ln 2 loses bits for large k).

Checked against exact values (60-digit decimal arithmetic), 1,000 to 2,000 random arguments per function, on P32W and on the Mac. P32W's result is the correctly rounded one for 99.5–100% of arguments in every function above, and never more than 0.51 ulp off. The Mac's libm, for comparison: `tan` 58%, `acos` 84–87%, `atan` 92%, `cbrt` 91–93%, `asinh`/`acosh`/`atanh` 66–88%, with errors up to 2 ulp. So P32W and the Mac still differ in the last digit now and then, and more often it is the Mac that is off.

Not changed here: `pow` (correctly rounded for whole powers up to 2047, D35; otherwise about 93%, within an ulp: y·log x needs more than 62 bits; D42 settles it). `erf`, `erfc`, `tgamma` and `lgamma` are musl's (D39), within a few ulp.

Speed (instructions per call on P32W, before → after): `sin` 29,500 → 29,300, `cos` 31,100 → 28,400, `atan` 35,300 → 17,700, `asin` 76,800 → 59,000, `sinh` 64,900 → 47,700, `expm1` 60,800 → 45,900, `exp` 37,700 → 38,700. No function got slower by more than 3%.

D39 — Python: deeper recursion, cmath, and musl's functions in libc (2026-10-15)

From Ken's "go ahead with 1, 2 and 3" (2: deeper recursion; 3: the rest of the Python follow-ons).

- **MicroPython is built stackless** (`MICROPY_STACKLESS`): a Python function calling another does not recurse in C, so recursion depth is set by the Python stack, 512 KiB from `malloc`. With D37 that costs only the pages used. Plain recursion goes from about 700 calls deep to about 10,000. A recursion that passes through C (a class's `__init__` making another instance) still uses C's stack, about 450 deep (D41 raises the stack limit).
- `cmath` **is on.**
- `erf`, `erfc`, `tgamma`, `lgamma` **are in libc** (`libc/erf.c`, `libc/gamma.c`, musl 1.1.16's code under its MIT licence; sinpi uses libm's `sin` and `cos`). `asinh`, `acosh` and `atanh` are libc's own (D38). MicroPython's copy, `lib/libm_dbl`, is not used, and C programs get the functions too.
- **Built on P32W itself.** `/usr/src/micropython` holds the sources, the tables `build.sh` made in `genhdr/` (MicroPython makes them with Python scripts, which P32W cannot run), and the host's objects (`tools/mkmpsrc.py`); its Makefile is the port's `Makefile`. With no objects, `make` there compiles all 126 files with the native cc in about 20 hours of P32W time (67 minutes in the emulator), and links a `python` byte for byte the host's. Before that run, the two compilers were compared on all 126 files: the same assembly. Three things had to change on the way:
    - `ld` **ran out of memory** linking 1.7 MB from 126 objects, keeping every input file whole until the end and building the output in a second buffer. An object file's text and data are read again when they are copied into the output, and the output is written straight to the file. The peak on the host falls from 10.4 to 6.5 MB, and the output is the same, byte for byte.
    - `-DNDEBUG`, as MicroPython's own release builds: no assertions. `python` is 13% smaller (1.79 → 1.55 MB).
    - **Relative** `-I` **paths in** `build.sh`, so `__FILE__` puts none of the host's paths into the binary.

    The disk then held 26,262 of its 32,768 blocks (80%; D40).

D40 — A 512 MB disk; boot checks it only after a crash (2026-10-16)

From Ken: "Raise to 512MB" (MicroPython's sources had filled a 32 MB disk image to 80%).

- **The disk image is 512 MB** (`software/Makefile`'s `DISK_MB`, and the test sessions). The design allows far more: 32-bit block numbers, and LBA28 addressing in the CF driver (128 GiB). Any real CF card holds it.
- **16,384 inodes** (`DISK_INODES`, `p32fs build --inodes`): one per 32 KB, more than ten times today's 1,400 files. The default, one per 8 blocks, would give 65,536, a 4 MB inode table.
- **Boot checks the filesystem only after a crash.** A full `fsck` every boot at 512 MB takes boot plus login from 17 to 68 million instructions. So `/etc/rc` runs `fsck -q -c`: `-c` checks only a filesystem that was not shut down cleanly. The kernel writes the superblock's clean flag down at mount, so `ustat` (D28) has `f_clean`: the flag as it was when mounted. A clean boot takes 6.4 million instructions on 512 MB; after a crash, with the full check, 44.5 million (68 with 65,536 inodes).

D41 — A process's stack may grow to 1 MiB (2026-10-16)

From Ken: "make it 1 MiB", after weighing the trade-offs (512 KiB before).

- **What it costs:** nothing for a program that does not go deep, since stack pages are made only as the stack grows, and the limit reserves no address space (the heap can still grow to within a page of the stack's actual bottom). A runaway recursion holds up to 1 MiB of RAM before it is stopped: one fifteenth of a one-card machine's RAM (D52). Stack growth is not reserved as heap pages are (D37), and the stack does not shrink once grown.
- **What it gains:** Python recursion through C (a class's `__init__` making another instance) about 950 deep against 450; awk's recursive functions about 2,500 deep against 1,000; and deeper recursion in cc and Lua's C code.
- **Changed:** `MAXSTACK` (kernel/param.h); MicroPython's `CSTACK` (960 KiB) and awk's own limit (1,008 KiB), each kept just short of the kernel's; sbrk(2), malloc(3), a.out(5), awk(1), python(1) and the kernel README.

D42 — pow in 96 bits: correctly rounded (2026-10-16)

From Ken: "ok improve pow". `pow(x, y)` is e^(y log x). In 62 bits, a relative error ε in log x becomes an absolute error |y log x|·ε in the exponent, up to 745ε. Measured against exact values over 3,000 arguments weighted toward large exponents, only 75% of results were correctly rounded, and the worst was 3.4 ulp off. The "93%" of D38 came from milder arguments.

What changed (`libc/softfp.c`):

- A three-word unpacked number, `w3`: 96 bits, with add, multiply and divide.
- **log x in 96 bits:** x = 2^e·f; with c = 1 + j/16 the nearest sixteenth, log x = e·ln 2 + log c + 2·atanh((f − c)/(f + c)). The remaining argument is below 1/64, so eight series terms are enough. The 17 values log(1 + j/16) and ln 2 are held to about 159 bits, as three doubles each.
- **y·log x and the reduction by k·ln 2** are also done in 96 bits. Only e^r, |r| ≤ 0.35, is left to the 62-bit `exp`.
- Whole powers up to 2047 still go by squaring (D35).

Result: correctly rounded on all 3,000 arguments, never more than 0.50 ulp off. The Mac's libm got 99.9% of the same. `pow` costs 77,000 instructions against 62,000 (24% more); the table of sixteenths keeps that down from the 106,000 of a plain series.

D43 — USB storage: two CH376S on the I/O card, FAT by tools (2026-10-16)

From Ken, after the design study (`docs/usb-design.md`): "CH376S plan A, FAT tools only".

- **Two CH376S on the I/O card** (D50), each a USB port: the first at `FF50` (data) and `FF51` (command and status), the second at `FF54` / `FF55`, both on interrupt level 6 (D52).
- **The CH376S** (WCH; datasheet CH376DS1) in parallel mode, as a module with 2.54 mm headers and a USB-A socket, used in its sector mode. It connects directly to the 5 V bus, needs little glue, and has working drivers to learn from (Fuzix). **Plan B:** a Raspberry Pi Pico running TinyUSB's mass storage host behind a mailbox interface, if too many drives fail with the CH376S. The driver keeps the controller behind one small layer, so a change touches only that layer.
- **FAT through user-space tools only:** `dosdir`, `doscp`, `dosrm`, `dosformat` on the raw device. A stick can also carry a P32W filesystem (`mkfs /dev/usb`, `mount`). No FAT in the kernel.
- **Order of work:** an emulator model of the CH376S (both emulators), the block driver and its tests, the FAT tools, then the hardware.
- **The module** (schematic from Ken) has every parallel signal on a 16-pin header, plus +5 V and GND; RSTI is not there (it has its own power-on reset), so the driver always starts with `RESET_ALL`. Its CH376 runs at 3.3 V from an LM1117; **Ken: run it at 5 V** (the regulator bridged, or a 5 V jumper), as in the datasheet's circuit: no level shifting.

The driver (2026-10-16): `kernel/usb.c`.

- Block device 2 (`/dev/usb`), for `mkfs`, `mount` and `fsck`; character device 5 (`/dev/rusb`, mode 666 as a floppy's would be), raw sectors for the FAT tools. `ioctl USBIOSIZE` gives the size (`<sys/usb.h>`). usb(4) starts manual section 4 (devices); `man` and the index know it.
- **Polled,** with a time limit on every wait (2 s, or 5 s for `DISK_INIT`), counted in clock ticks. A drive is found at first open. Pulling it out drops its blocks from the cache, and the access fails with EIO.
- **Read errors reach the filesystem,** for every disk. A failed read filled with zeros and cached as valid would make a pulled-out drive's free-block bitmap look empty, and `cp` would carry on "writing". So `bread` leaves a failed block uncached with `B_ERROR` set, and `balloc`, `readi` and `writei` return EIO. The "disk error" message is printed once per device until it works again.
- **Tested:** session 17 (a blank 4 MB stick: `mkfs`, `mount`, copy, `fsck`, read back, `/dev/rusb`); session 18 (pulled out mid-copy: `cp` gets a write error, the system carries on, the root stays clean).

D44 — The FAT tools: one program, sectors by number (2026-10-17)

From Ken: "go ahead with the FAT tools" (D43's third step).

- **One program,** `software/cmd/dos.c`**, under five names:** `dosdir`, `doscp`, `dosrm`, `dosmkdir`, `dosformat` (links to `/bin/dos`, as `gunzip` is to `gzip`). `dosmkdir` is beyond D43's four: making a directory is half of `doscp -r` already. mtools-style: drive paths start with `::` for `doscp`; names match without regard to case. dos(1).
- **FAT12, FAT16 and FAT32, with long names,** read and written; the volume in an MBR partition (the usual stick) or from sector 0 ("superfloppy"). Long names are UTF-8 on P32W, UCS-2 on the drive. A name that fits 8.3 in one case gets only a short entry (with the lower-case flags that Windows and the Mac honour); others get long-name entries and a `BASIS~N` short name. Times are local time, by TZ. The FAT is read through an 8-sector cache written back to both copies at the end; the FAT32 FSInfo free count is kept right.
- `doscp` **is careful with space:** a file's clusters are allocated and written first, and its directory entry made last, so a drive that fills up mid-copy loses nothing (the clusters go back). A long name over P32W's 28 characters comes off

the drive under its short name, with a message.

- `dosformat` writes an MBR with one partition from 1 MiB, then FAT32 for 512 MB and more (Windows' cluster sizes) or FAT16 below (the smallest cluster that fits; 3 MB at least). `-F` overrides, `-v` labels. No questions asked: P32W's `mkfs` asks none either.
- **Two kernel ioctls on** `/dev/rusb:` `USBIOREAD` and `USBIOWRITE` move 1 to 128 sectors by number (`struct usbio` in `<sys/usb.h>`), because a file offset stops at 2 GB and sticks are bigger. Four sectors go per CH376 command, staged in the driver's 2 KiB buffer, so a bad user address is found before the drive is touched. `USBIOWRITE` needs the descriptor open for writing. `lseek` works on `/dev/rusb`, as on the CF card's raw device.
- `-i image` runs the tools on an image file, so they also build and run on the Mac (`cc dos.c`), which is how they were first tested.
- **Tested against the Mac's own FAT code** (`software/tests/fat_test.sh`, in `make test`): a stick image the Mac formats and fills (`hdiutil`) is read and written by P32W in the emulator, then checked by `fsck_msdos` and a read-only mount; likewise sticks `dosformat` made (600 MB FAT32, 40 MB FAT16) and one that fills up. Session 19 (a blank 40 MB stick: both formats, copies both ways, the error messages) is recorded, and is the same on p32uemu.

D45 — Networking: Ethernet and Wi-Fi through a Pico 2 W and a W5500, before SLIP (2026-10-17)

From Ken, after the design study (`docs/net-design.md`): "B: Pico coprocessor", and "Ethernet first".

- **A Raspberry Pi Pico 2 W with a W5500** on the I/O card (D50), beside the CH376S ports, behind a **mailbox**: latches, with flags, on interrupt level 7 (D52). The Pico's 3.3 V side is buffered to the 5 V card. Its registers follow the first CH376S's (`FF52` on).
- **The mailbox speaks network operations,** not W5500 registers: socket, connect, listen, accept, send, recv, close, DNS, status. The coprocessor does DHCP, DNS, ARP timing and keepalives, and **TLS** (mbedTLS), so P32W can reach `https://` itself. The deciding point: a TLS handshake on P32W (about 0.5 MIPS) would take minutes.
- **P32W keeps everything above the socket:** sockets as file descriptors (`FD_SOCKET`), BSD-style system calls, and the programs. A network-interface layer underneath leaves room for SLIP.
- **Ethernet before SLIP.** SLIP, with P32W's own TCP/IP stack, stays in the backlog as a later project, for learning; the W5500's MAC-raw mode could carry the same stack.
- **Order of work:** the mailbox model in both emulators, mapped onto the Mac's sockets (and the Mac's TLS); sockets in the kernel and the driver; the programs; the firmware; then the hardware.
- **Single latches, not FIFOs,** in the mailbox: the Pico answers within a microsecond, under one P32W instruction, so a FIFO would buy nothing the driver could use.

The protocol and the emulator model (2026-10-17; Ken: "go ahead with the emulator model"):

- `docs/net-mailbox.md` is the reference for the model, the driver and the firmware.
  - Registers: `NDATA` at `FF52`, `NSTAT/NCTRL` at `FF53`, with the bits `RXF`, `TXE`, `ATTN`, `IEN` and `READY`, and `RESET`.
  - Messages: requests and replies with the same 4-byte header and up to 1024 bytes of payload; 17 commands; 8 sockets of 4 types (TCP, UDP, TLS, DNS); socket states; 15 status codes.
  - `CONNECT` only starts a connection; `ATTN` and `EVENTS` say what changed, so one slow handshake never holds the mailbox.

- **The model** (`emulator/p32io.h`, `-N`, both emulators) carries the commands out with the Mac's sockets:
  - 10.0.2.2 is the Mac (QEMU's numbers);
  - listening ports below 1024 go up by 8000;
  - DNS is the Mac's resolver;
  - TLS is `tools/p32tls.py`, Python's `ssl` with the host-name check. LibreSSL's `s_client` was tried first; it cannot check the host name.
  - An idle P32W waits for the sockets in real time, so emulated time keeps pace with the network.
- **Tested** (`emulator/test/net_model.c` with `net_peer.py`'s servers, in `make test`, the same on both emulators): TCP there and back with shutdown and the end of the stream; a refused connection; DNS, and a name that does not exist; UDP; listen and accept, with `ATTN` and `EVENTS`; TLS with a test certificate, and its refusal for the wrong name; all 8 sockets in use; bad requests; `RESET`; no coprocessor without `-N`.

Sockets in the kernel (2026-10-22; Ken: "go ahead with the kernel sockets and driver"): `kernel/net.c`.

- **A socket is a file** (`FD_SOCKET`): `read`, `write`, `close`, `dup`, `fork` and `select` work on it, and `fstat` says `S_IFSOCK`. Each is one of the coprocessor's 8 sockets; `socks[]` is indexed by its number.
- **BSD's calls at 4.2BSD's numbers:**
  - `socket`, `connect`, `bind`, `listen`, `accept`;
  - `sendto`/`recvfrom` (`send`/`recv` in libc), `shutdown`;
  - `setsockopt`, `getsockopt`, `getsockname`, `getpeername`;
  - `select`, and `fcntl` (`O_NONBLOCK`). A system call carries four words, so `sendto` and `recvfrom` pass an argument block, the options pass level and name in one word, and `select` drops `nfds` (an `fd_set` is one word: `NOFILE` is 20). `netctl` (160) is P32W's own: interface information and settings, and host names.
- **TLS** is `socket(AF_INET, SOCK_STREAM, IPPROTO_TLS)` (253, RFC 3692's experimental number), with the server's name given by `setsockopt` (`TLS_HOSTNAME`) before `connect`.
- **The mailbox** is used only in process context, an exchange at a time without sleeping (the kernel is not pre-empted), so processes never interleave on it. **Waiting:** a process checks its socket. If it cannot go on, it sleeps until `ATTN`, unless `ATTN` is already up. The interrupt handler masks `IEN` (the line stays up until `EVENTS`) and wakes every sleeper; each sends `EVENTS` and looks again. A wake every 200 ms covers a missed event; signals end a wait with `EINTR`.
- **select** covers terminals, pipes and sockets. Its sleepers are woken by terminal input, by the network interrupt, and by every clock tick (timeouts, and pipes).
- **The protocol gained one detail:** `STATE`'s "bytes ready" counts the connections waiting on a listening socket.
- **libc:** the wrappers; `htons` and the rest; `inet_aton`, `inet_addr` and `inet_ntoa`; `gethostbyname` through `netctl`; `strerror` for the network errors. **Manual:** socket(2), send(2), getsockopt(2), select(2), fcntl(2), netctl(2), inet(3).
- **Tested:** `software/tests/nettest.c`, as ken under the kernel, with `net_test.py`'s servers on the Mac (`make test`; the same on both emulators). It covers TCP there and back, 4000 bytes, shutdown and the end; a refusal; a non-blocking connect with `select` and `SO_ERROR`; a blocking read ended by `SIGALRM`; DNS; UDP; `select` timing out; a server accepting the Mac's client; TLS, and the wrong name refused; all 8 sockets in use; and the errors.

The first programs (2026-10-23; Ken: "go ahead with ifconfig, ping, telnet and fetch"):

- `ifconfig` shows the interface, and for root sets a fixed address or DHCP (`netctl`). Manual page in section 8, as BSD's is.
- `ping` builds ICMP echo requests itself (checksum, id, sequence) and sends them through a socket kind of its own, `socket(AF_INET, SOCK_DGRAM, IPPROTO_ICMP)`. That is socket type 5 in the protocol, the W5500's IP-raw mode. The model uses the Mac's unprivileged ICMP socket and strips the IP header the Mac returns. Times are in 20 ms clock ticks.
- `telnet` is RFC 854: line mode until the server offers to echo, then raw. It answers TERMINAL-TYPE (VT100), NAWS (80 x 24) and SGA; `^]` closes.
- `fetch` does HTTP/1.1 with `Connection: close`, redirections (5), chunked replies, `-o`, `-O` and `-v`. Its HTTPS is the coprocessor's TLS socket: fetch speaks only HTTP.
- **Tested** (`software/tests/netcmd_test.py`, in `make test`, the same on both emulators): every case against servers on the Mac (HTTP, HTTPS with the test certificate, a telnet server that negotiates). By hand, P32W in the emulator fetched `https://example.com/` (the certificate checked against the Mac's store) and pinged it.

D46 — Remote logins, FTP and the network clock (2026-10-23)

From Ken: "go ahead with telnetd, ftp and NTP".

- **Pseudo-terminals in the kernel** (`kernel/tty.c`), four pairs. The slaves, `/dev/ttyp0-3`, are tty minors 5-8 (D53): the same line discipline, signals and modes as tty0, so `getty`, `login`, `sh` and `vi` run on them unchanged. The masters, `/dev/ptyp0-3`, are character device 6: writing one feeds the slave's input path (as a UART's interrupt does), reading it takes the slave's output queue (which no UART empties).
  - Closing a master hangs the slave up: SIGHUP to its processes, and their reads end.
  - A master's read ends once the slave has been opened and then closed by everyone.
  - `select` covers both ends.
  - The four pairs cost about 48 KiB of kernel memory (three 4 KiB queues each).
- **utmp:** `ttyp0-3` are records 5-8 (`utmp_slot()` in `<utmp.h>`), so `who` lists remote users.
- **telnetd** (`/etc/telnetd`, started by `/etc/rc`):
  - It detaches itself, as `cron` does, and with no network it exits without a word.
  - Each connection takes the first free master, runs `getty` on the slave, and carries bytes with `select`: telnet commands out, CR LF and CR NUL as one Return, 255 doubled.
  - It offers WILL ECHO and WILL SGA (character mode, P32W echoes).
  - The session belongs to init (a double fork). Logout or the client leaving closes the master and clears the utmp record.
  - It holds one of the coprocessor's 8 sockets while it listens.
- **ftp** (`/bin/ftp`): an interactive client, passive mode only. The data connection goes to the control connection's address, not the one in the PASV reply (a server behind NAT names a private one), as curl does. Binary by default; ascii converts CR LF and LF. `-n` skips the automatic login.
- **ntpdate** (`/etc/ntpdate`): one SNTP request (three tries), the transmit time plus half the round trip. Root sets the clock, which `stime` also writes to the battery-backed clock; `-q` only asks. `/etc/rc` runs it if `/etc/ntp.conf` exists. None is shipped, so no test depends on the internet.
- **Tested** (`software/tests/netsvc_test.py`, in `make test`, the same on p32uemu), against the Mac:
  - two telnet logins in a row on `ttyp0`, `who` during and after;

    - an FTP server: a wrong password; `ls`, `nlist`, `cd`; `get` binary (5000 bytes checked on the Mac) and ascii (CR LF to LF); `put`, `delete`, `mkdir`; a missing file;
    - an SNTP server answering 2030: `-q` as ken, refused as ken, set as root, `date`.
- **The emulator flushes its output when P32W goes idle,** so a session that hangs shows how far it got.

D47 — Bypass capacitors on a 0.1" footprint, on every board (2026-10-23)

Ken's rule for every board on every system: each 100 nF bypass capacitor uses a 0.1" (2.54 mm) lead spacing, never a 0.2" (5 mm) disc.

- **The footprint:** `Capacitor_THT:C_Disc_D3.0mm_W2.0mm_P2.50mm` (KiCad calls 2.54 mm "2.50"), for `C_DISC` in `generators/p32net.py` and the backplane's bypass capacitors.
- **The small part's label** goes beside it (2.3 mm), not below: between its 0.1" pads there is no room, and below it the label meets the next part's silk.
- **Routing is re-tried:** small footprints change the DSN's order, and Freerouting is sensitive to it (the P8X lesson: re-roll the placement, not the route). `build_board.sh` places and routes again, up to `ROUTE_TRIES` (4) times, until DRC finds no unconnected pads.

D48 — Board layout rules for every system: the front edge, readable labels, placement drawings (2026-10-24)

From Ken, for all systems (P32W, P8X, YACC1, ...): "switches and LEDS should be near edge farthest from bus connector. Connectors such as video and DB9 should also be on edge farthest from bus connector"; part numbers "should be visible once chips are installed"; switches and LEDs labelled with their function, "if say a toggle switch has two positions label each option"; "for cards with an addon like a pico or nano ... near any edge (other then bus connector) with USB port facing edge"; and the placement drawing must show the board and its parts in one orientation. (Also in Ken's global instructions, ~/.claude/CLAUDE.md, so they carry to the other projects.)

- **The front edge** is the edge opposite the backplane connectors. Switches, buttons and LEDs go there, and so do the connectors for outside cables (video, D-subs, USB, network), pointing outwards so a cable plugs straight in. A plug-in module (Pico, Tang Nano, the CH376S) goes at an edge other than the bus edge, its USB port facing out. Jumper blocks and cable headers go near a board edge, never in the middle.
- **Labels stay readable with the parts in.** A chip's reference and type sit on one line above the top pin row, the reference over pin 1's end and the type over the other, never under the chip. A socketed PLCC chip or a module gets its label above its socket, and an axial diode above its body.
- **Function labels.** An LED's function sits right under it. A button carries its function above it (RESET, STEP). A switch's positions are labelled at both ends with what each selects, ON at the end the part marks "on" (for example RUN / HALT, or INSTR / CYCLE for what STEP steps: an instruction or a microcycle).
- **Placement drawings** take the board's orientation (portrait for a tall board), with the board centred above the title block (A4, or A3 if it does not fit).
- Internal cables are not covered: a drive port whose cable stays inside the case sits beside its logic.

D49 — The frame: 9U × 340 mm cards, three connectors, ten slots 1 inch apart (2026-10-24, Ken: "go with 366.7 x 340 and 3 connectors")

Decided:

- **Card size 366.7 × 340 mm:** 9U tall (the rack's height, the connector edge) and 340 mm deep, about 1,250 cm² per card.
- **Three 96-pin DIN 41612 connectors per card,** J1–J3 (288 pins), down the 366.7 mm edge.
- **A 10-slot backplane, slots 1 inch (25.4 mm, 5 HP) apart** (Ken: "lets set card spacing at 1\" (25.4mm)"). 5 HP is a whole Eurocard unit, so standard card guides and front panels fit (one 5 HP panel per card). 25.4 mm less the 1.6 mm board leaves about 23.8 mm to the next card's solder side; allowing about 2.5 mm for that card's pins and joints and 1 mm of air, **parts up to about 20 mm tall** fit on the component side. That covers DIP chips in sockets, standing 10 mm electrolytics and crystals; anything taller lies down.

Why:

- **Room for the CPU's datapath on one card.** A 340 mm-deep card holds the sequencer and control store, the instruction register and decode, the register file, the PC and the ALU module together, with the MMU and the data path on a second card beside it. The CPU's signals then stay on its own boards rather than crossing the backplane.
- **A system bus that DMA masters and several memory cards can share.** The bus (D50) carries only physical memory cycles, interrupts, arbitration, clock and reset, with room for page swapping, several memory cards, DMA, and a wait line that stretches a read or a write. J1 carries the cycle, J2 control.
- **A private local link between the CPU card and the MMU / data card.** J3 in slots 1–2 only joins those two cards, so the microcode's memory controls never become bus signals other cards must respect.
- **Cost:** JLCPCB, 5 boards at 366.7 × 340: $171.70 at 6 layers, $254.00 at 8 (2026-10-24).

D50 — The system bus, the cards, and the choices under them (2026-10-24, Ken: "agree with all your recommendations")

Ken accepted every recommendation put to him with D49. In full in architecture.md; the bus in bus.md, generated by `generators/p32bus.py`.

- **A system bus:** the backplane carries only physical memory cycles, interrupts, arbitration, clock and reset (J1 the cycle: D0–31, A2–31, BE0–3, `-MRD`, `-MWR`, `-ACK`, `-WAIT`; J2 control: CLK, `-CLK`, `-RES`, PWROK, `-LOCK`, `-IRQ0–7`, `-BREQ` / `-BGNT`). The CPU's microcode fields stay on the CPU card; J3 in slots 1–2 is the local link between the CPU card and the MMU / data card (LD0–31, DOE, DLD, MEM, SIZE, SIGNED, SUPER, TLBW, CRSEL, FAULT, FC, `-WSTB`, STALL, LOCKREQ).
- **The CPU card** in slot 1: the sequencer and control store, the instruction register and decode, the register file, the PC, the ALU module (D21), the clock and the front panel.
- **MMU / data on its own card** in slot 2: MAR, the split I/D TLB (D13), the fault checks, MDR and byte steering, BADADDR, the bus master interface, the arbiter and `-LOCK`. **GALs and TTL**, no CPLDs.
- **Clock:** designed for **5 MHz**, 3 MHz the safe setting.
- **Bus timing:** synchronous, one clock a cycle, `-WAIT` adds clocks; no `-ACK` within the first clock is a BUSERROR.
- **Arbitration: radial** (`-BREQ` / `-BGNT` per slot to the arbiter in slot 2), the slot setting only the priority. **Interrupts:** 8 shared, open-drain levels. `-LOCK` for atomic read-modify-write (D51, D54).

- **Memory cards:** 1–4, 16 MiB of SRAM each, the boot ROM on one; placed by **jumpers** (16 MiB windows, ROM enable).
- **Disk:** 2 CF + 2 IDE; PIO first, DMA-capable.
- **I/O:** four UARTs, the timer, the real-time clock (D23), ports, two USB ports (CH376S, D43), and Ethernet and Wi-Fi through **one Pico 2 W + W5500** (D45).
- **Video:** VGA 640 × 480, 80 × 25 text and a 640 × 480 one-bit overlay; **PS/2** keyboard and mouse. Ken confirmed **25 rows** over 30 the same day (asked: "25"): the classic PC console size. 25 rows of 8 × 16 cells fill 400 of the 480 lines; the text is centred with a 40-line band above and below (the overlay still covers all 480).
- **Debug:** a host link (a Pico), a logic analyzer on probe inputs, and a fast clock; and a **bus master**, so it loads and reads memory by DMA with the CPU halted.
- **10 slots**, 1 inch apart (D49). I/O cards, like memory cards, are placed by jumpers, not by slot (D52).
- **ISA kept,** with an atomic swap (D51) and any new MMU fault causes added.

D51 — SWAP, the atomic exchange (2026-10-24; under D50's "ISA kept, plus an atomic swap")

- `SWAP rd, imm(rs1)`, opcode **0x4C**, format LD: `t = mem32; mem32 = rd; rd = t` at an aligned word. The lock primitive: `SWAP` a 1 into the lock word; a 0 back means the lock was free and is now held.
- **Faults** as a load and then a store of the same word: MISALIGNED, TLBMISS_R, then PROTECT if the page is read-only. Commit-after-access (D14) holds: the old word goes to scratch T0 and reaches `rd` only after the store succeeded, so a faulting `SWAP` changes nothing and is restarted.
- **Microcode** (10 cycles with the fetch): address to MAR, `LOADW` into the MDR, MDR to T0, `rd` to the MDR, `STOREW`, T0 to `rd`.
- **Atomicity:** interrupts are taken only between instructions, but DMA masters share the bus, so the read and the write must be one bus tenure. The microcode marks the read as locked (`LOCKREQ` on the local link, held through the store), and the MMU / data card holds `-LOCK` over both (D54).
- Rejected for now: compare-and-swap (needs a third operand and a compare in the middle of a locked tenure) and load-linked / store-conditional (a reservation register on the MMU / data card). `SWAP` is enough for spin locks and semaphores on one processor with DMA.

D52 — The address map: every card placed by jumpers, nothing by slot (2026-10-24, Ken: "Address jumpers. Not slot specific")

Device cards are placed by **address jumpers**, as memory cards are, and nothing in the system depends on which slot a card is in (apart from slots 1–2, the CPU and MMU / data cards and their local link). Following from that:

- **The bus carries no slot-number pins and no per-slot interrupt lines:** a slot number is no use to software that does not know which card is in which slot. 162 signal pins, 48 spare. The radial `-BREQ` / `-BGNT` pairs are there because they are how a card asks for the bus; the slot sets only its priority.
- **Interrupts:** each device's level is chosen by a **jumper** on its card; devices may share a level (`-IRQn` is open-drain). Each card has an **interrupt status register** so the kernel can tell which device on a shared level is asking.
- **Physical map:**

Range	What
`0x0000_0000`–`0x000F_FFFF`	boot ROM (1 MiB), on the memory card with its ROM jumper fitted; hides the RAM under it; a write is a BUSERROR
`0x0000_0000`–`0x03FF_FFFF`	RAM: up to 4 memory cards of 16 MiB, window n = `n × 16 MiB` (2 jumpers)
`0x0400_0000`–`0xFDFF_FFFF`	nothing (BUSERROR)
`0xFE00_0000`–`0xFE07_FFFF`	**video memory**: the video card's text, font and overlay RAM, 64 KiB at `0xFE00_0000 + n × 64 KiB` (n = its register window's jumpers)
`0xFFFF_8000`–`0xFFFF_FFFF`	**device registers**: 8 windows of 4 KiB, window n at `0xFFFF_8000 + n × 4 KiB` (3 jumpers per card)

Device windows are 4 KiB and sit in the top 32 KiB so that every device register is one instruction away, as an offset from `r0` (`lbu r1, 0xFFFFFF05(r0)`: the offset is encoded in 16 bits, `0xFF05`, and sign-extends). A 1 MiB window per card would need a base register for every access.

- **A card's registers are the top 256 bytes of its window** (offset `0xF00`–`0xFFF`), so the I/O card, at its default window 7, has `UART0` at `0xFFFF_FF00`, the timer at `FF10`, the USB ports at `FF50` and `FF54`, the network mailbox at `FF52`. Every card has, at offsets `0xFF0` / `0xFF1`, a read-only **card ID** (1 = I/O, 2 = disk, 3 = video, 4 = debug) and **revision**, and at `0xFF4` its interrupt status (a word, a bit per device); the kernel finds the cards by reading each window's ID (an empty window is a BUSERROR).
- **Default jumpers** (the settings the documents and the kernel assume): window 7 (`0xFFFF_F000`) the I/O card, 6 (`0xFFFF_E000`) the disk card, 5 (`0xFFFF_D000`) the video card, 4 (`0xFFFF_C000`) the debug card; 0–3 free. Interrupt levels: 0 timer, 1 disk (all four ports), 2 UART 0 and 2, 3 UART 1 and 3, 4 PS/2 keyboard and mouse, 5 RTC, 6 USB (both), 7 network.
- **The disk card** at window 6: port p at `0xFFFF_EF00 + 16 p` (0 = CF 0, 1 = CF 1, 2 = IDE channel A, 3 = IDE channel B).
- **RAM sizing:** a missing memory card does not answer, so the kernel probes each 16 MiB window at boot with a load that may take a BUSERROR, and marks the frames of a window that does not answer as in use for good. The boot line says "memory: N MiB of RAM, …", and `sysinfo` (so `top`) reports the RAM found. No jumper tells the kernel the size.

D53 — The video console (2026-10-24, Ken: "1")

Asked whether the video card's screen and PS/2 keyboard should be the console when the card is fitted, or one more terminal beside UART 0's, Ken chose the console.

- **The screen and keyboard are a terminal,** `/dev/tty4` (kernel `video.c`). When the video card answers at boot (its ID read with a probe), the kernel's messages go to the screen as well as to UART 0, and `init` keeps a getty on tty4 as on tty0 and tty1. Without the card, tty4 does not open and nothing changes: UART 0 is the console.
- **Terminal numbering** (minors and `/etc/utmp` slots): tty0–tty3 the I/O card's four UARTs, tty4 the screen, ttyp0–ttyp3 (the pseudo-terminals, D46) 5–8. `ps` shows them as `0`–`4` and `p0`–`p3`.
- **The screen** understands what P32W Unix's programs send (vi, top, BASIC): CR, LF with scrolling, BS, TAB, and `ESC [ row ; col H`, `ESC [ J` / `2J`, `ESC [ K`, `ESC [ n A`–`D`, `ESC [ … m` (normal, bright, underline, inverse). The last column wraps late, as a VT100's.
- **The keyboard:** set-2 scan codes decoded in the kernel, with shift, ctrl and caps lock; the arrows type `ESC [ A`–`D`, Delete DEL.

- **In the emulators** the card is fitted only with `-V FILE` (the screen written at exit) or `-K FILE` (the keyboard types the file), so every other run is a machine without one.
- Found on the way, and fixed: a mode change (`TIOCSETP`) left typed-ahead input counted in the old mode's units, so the shell's line editor (CBREAK) read three characters of a typed-ahead line and waited. The driver recounts it.

D54 — The bus lock in the microcode (2026-10-24, under D51; Ken: "Go ahead with the microcode")

- **A one-bit microword field,** `LOCK` (bit 42: the microword is 43 bits, in six flash chips). It drives `LOCKREQ` on the local link. A `MEM` code would also work, but the field's one spare value (7) is kept free.
- **The rule, checked by** `p32ucode.py`**:** `LOCK` only on a `LOADW`, and the next bus cycle on every path after it is its `STOREW`. The MMU / data card holds `-LOCK` from the locked read to the end of that store, or until either faults; while it is held the arbiter grants nothing.
- `SWAP`**'s read is the one locked cycle.** Nothing else in the ISA needs the lock.
- `STALL` needs no microcode: the MMU / data card holds the sequencer while a cycle waits (`-WAIT`) or the bus is lent to a DMA master, and the microword's meaning does not change.
- **Tested in** `p32uemu` with a DMA master (`-D ADDR:PERIOD`) that keeps adding 1 to the word that a program empties with `SWAP`: with the lock no addition is lost; with the arbiter told to ignore it (`-Dnolock`), up to 1,800 of 19,800 are.

D55 — Every card designed for the best result; the ISA may change with it (2026-10-25, Ken: "GO with all the Alternatives now open ... I am fine starting as much from scratch as need for the best results")

- **Every card is designed from first principles** for the best result: the CPU's datapath, the clock, the register file, the ALU, the microword, the control logic's parts, and the MMU / data card and the rest with them. No circuit is kept for having been drawn before.
- **The ISA may change** where the new hardware runs better with it (Ken: "adding to or changing ISA to better operate in the new hardware is fine"). The software follows: the assembler and the opcode header are generated from microcode/p32isa.py, the compiler and P32W Unix are rebuilt, and the instruction-level emulator (p32emu) defines each instruction's meaning.
- **The three bus connectors are used to the full** (Ken: "Take full advantage of the 3 bus connectors"): the bus is redesigned with the datapath, not kept for its pin list.
- **The layout rules apply to every new card** (Ken): D19, D47, D48 and the global rules -- switches, LEDs and cable connectors at the front edge, pointing outwards; jumpers and headers at an edge; every reference readable with the parts in, ICs with their type; every control and each switch position labelled; placement drawings on a sheet that matches the board.
- **The order:** the datapath and microword first, as a microcode-level emulator model running P32W Unix and measured against today's (6.28 cycles an instruction at 3 MHz, 0.48 MIPS); then the bus; then the cards, the CPU card first.

D56 — The three-bus datapath (2026-10-25, under D55)

D55 asked for the best result from first principles. The datapath is the lever on every other choice, so it came first, built and measured in the microcode-level emulator before any card (docs/microarch.md).

- **Three buses:** A and B carry the operands (the register file's two read ports, the PC, the immediate unit), R the result (the ALU, memory data, the immediate unit, MAR, the store data, CAUSE, BADADDR, a hardware register) to one destination a cycle. **The register file reads two registers and writes one every cycle** (two SRAM copies written together).
- **A pipelined ALU:** a word latches A, B and the function; the result is on R from the next cycle, and the comparison of the latched operands is a condition from that edge.
- **Fetch overlaps the instruction before:** every instruction starts with MAR = PC and its last step sets MAR to the next PC; the generator checks it on every path.
- **MDR (the store data) on the CPU card, loaded from B**; a load's data comes back on R already steered. J3 carries R (`LR0`–`LR31`) and its source and destination codes.
- **The microword: 53 bits, seven flash chips.**
- **Measured** (P32W Unix booted, root logged in, `halt`; 7,490,575 instructions on both): **3.57 cycles an instruction** against 6.28 for one internal bus — 1.76× at the same clock: 0.84 MIPS at 3 MHz, 1.00 MIPS at 5 MHz with a wait state on every bus cycle, 1.40 MIPS without. Bus cycles are now 40% of all cycles, so a memory path fast enough for 5 MHz without a wait state is the next lever.
- Every test program runs on it instruction for instruction as on the reference emulator (`make test`); the ISA did not change.

D57 — The bus's spare pins: more power, vectored interrupts (2026-10-25, Ken: "Power Pins and Vectored interrupts")

Of the options for the bus's spare pins (D55; a CPU trace bus to every slot, more power pins, vectored interrupts, bus parity, a 64-bit memory path), Ken chose two:

- **More power pins.** J2's row B even pins 4–30 are VCC, on every card and every slot: with row B's odd pins (GND) they alternate VCC and GND down the middle of J2. **VCC 18 → 32 pins a card**, GND 60. (DIN 41612 contacts carry about 1.5 A each when warm: 32 × 1.5 A = 48 A of 5 V a card, far above any card's need; the gain is the margin and a lower-inductance supply into the card.)
- **Vectored interrupts.** One new bused line, `-IACK` (J2 A18). When the CPU takes level *n*, its interrupt entry runs an **acknowledge cycle**: the MMU / data card puts `A4:A2` = *n* and pulls `-IACK` instead of `-MRD`; the card driving level *n* answers `-ACK` and puts its **vector** on `D7:D0` — its window number (D52) × 16 + the asking device's bit in its interrupt status register (the lowest asking one on that level). Slot-independent, like everything else on the bus: the vector comes from the window jumpers the card already has.
  - **One card a level**: two cards' jumpers must not put devices on the same level (they would both answer). One card may use several levels.
  - **No answer is not a bus error**: the vector is `0xFF`, and the kernel falls back to reading the cards' interrupt status registers.
  - `CAUSE` = `0x8000_0000` | **vector** << **8** | ***n*** for an interrupt (isa-system.md B2). The kernel dispatches on the vector.
  - **The microcode:** a new memory cycle type `IACK` (the eighth of the 3-bit `MEM` field): interrupt entry puts `CAUSE` (with *n*) into MAR, and the acknowledge cycle's data — MAR with bits 15:8 replaced by the vector — goes into `CAUSE`. One cycle more per interrupt (six).
- Not chosen: the trace bus, bus parity, a 64-bit memory path. 33 pins a card stay spare (J2 rows A and C, J3).

D58 — The CPU card's open choices (2026-10-25, Ken: "the first three, not the trace bus")

docs/cpu-card.md §3 is decided:

1. **Register-file addresses in GALs** (three ATF22V10s for the A, B and write ports, with the r15 bank rule and the r0 write sink), not 74HCT151 muxes.
2. **Two copies of the register file in CY7C199 SRAM** (one per read port, written together at `-WSTB`), not dual-port SRAM.
3. **A dedicated comparator** (74HCT688s and a 74HCT85 tree, latched at the ALUGO edge), so a branch's condition is ready the cycle after its operands; not a two-cycle branch.
4. **No trace bus** (D57): the debug card watches the CPU through J2 and its own probe header.

D59 — The address adder, indexed and post-increment loads and stores, LI32 (2026-10-25, Ken: "Go with the address adder and the three cheap additions")

From the measurements in isa-measurements.md:

- **An address adder (AGU) on the CPU card** puts A + an offset on R in the cycle the operands are read: the IR's sign-extended 16-bit offset, the post-increment format's 8-bit offset, or B shifted left by 0–3 (an index). A load or store's address reaches MAR without the ALU's cycle: **loads and stores take 3 cycles instead of 4**, `SWAP` 5 instead of 6. The microword gains `AOFF` (2 bits), `MARINC` (1) and the source code AGU in `RSRC`: **56 bits, still seven flash chips**.
- **Indexed loads and stores** (format X, 0x80–0x8A): address = rs1 + rs2 × 1, 2, 4 or 8. Loads 3 cycles; stores 4 (three registers to read, two ports).
- **Post-increment loads and stores** (format P, 0x90–0x9A): the access at rs1 + off8, then rs1 += inc8. 4 cycles. rs1 is written after the access has succeeded, so a fault leaves the instruction restartable (swap, D51).
- `LI32 rd, value` (0x2E, two words): 3 cycles instead of `LUI` + `ORI`'s 5, the same 8 bytes. MAR becomes a counter on the MMU / data card (`MARINC`, a new J3 line). The assemblers' `LA` and 32-bit `LI` are `LI32`. rd is in field B so the decode table keeps eight must-be-zero classes (the X format took the last one).
- **The compilers** (`cc/cgen.py` and P32W's own `software/cc/gen.c`, which must produce the same code) use indexed accesses for `p[i]` and post-increment for `*p++`; the native assembler and the debugger know the new formats.
- **Measured** (`tools/isamix.sh`, each workload the same before and after): **13–18% fewer cycles** — the shell session 91.6 M → 75.9 M, the compile 2,147 M → 1,799 M, Python 62.2 M → 54.1 M, SQL 157.4 M → 129.6 M; the boot benchmark 26.74 M → 22.43 M (7.49 M → 7.05 M instructions): **1.19× at the same clock**, 2.10× the single-bus datapath.
- **The adder's timing** is the new critical path at 5 MHz (≈ 195 ns with carry lookahead, microarch.md §8), to be confirmed in the CPU card's gate-level simulation; at 3 MHz it has room.
- Not taken (measured, Ken's choice): link-register calls, multiple load / store, compare with a constant and branch.

D60 — Memory cards may be partly populated (2026-10-26, Ken: "I would like option to not fully populate cards fully"; "for now stick to Static")

- **A memory card is eight banks of 2 MiB**, each bank four AS6C4008 (512K × 8, one per byte lane): the smallest step for a 32-bit memory. The banks are fitted in order from bank 0: a card holds 2, 4, … 16 MiB.
- **A 3-bit jumper on the card says how many banks are fitted.** The card answers (`-ACK`) only inside its fitted banks: an address above them is a bus error, as a missing card is.
- **The kernel probes in 2 MiB steps across all four windows** (D52) and uses every bank it finds; the memory may have holes (a half-fitted card in window 0 and a full one in window 1). The emulators' `-M` takes a size per card (`-M 16,6`).
- Static RAM for now; a DRAM card for density is in the backlog, for later (Ken).
- Done (2026-10-26): the emulators' `-M S0,S1,…` (MiB per window), the kernel's probe in 2 MiB steps (saving and restoring the word it tests) and its frame pool with holes, `/dev/mem` by bank; the boot monitor's stack moved to the top of the first 2 MiB bank (`0x001F_FC00`), which every machine has; the memory card's netlist (D63).

D61 — The CPU card as built (2026-10-26, under D55-D59)

docs/cpu-card.md; `generators/card_cpu.py`, verified in the gate-level simulator (`generators/sim_cpu.py`).

- **One commit edge, eight phases.** A 24 MHz oscillator, eight phases a 3 MHz microcycle from one GAL; every destination and the pipeline register take their values at one edge (`CCLK`). The next microaddress is computed during the cycle from the cycle's own signals (the live comparison of A and B, the opcode on R during the fetch, faults), which keeps the microcode's meaning with one edge.
- **The register file is written late**, in phases 0–2 of the next cycle, before its reads (phases 3–7): no bypass.
- **196 ICs** (23 GALs, each checked exhaustively against its spec), through-hole; the ALU on the Tang Nano 20K behind its LVC latches.
- **Verified:** the card as wired runs the 17 cross-check programs with the state at every instruction equal to `p32emu`'s and the cycle count equal to `p32uemu`'s; `make test` runs five.
- **3 MHz** is this card's clock; 5 MHz needs a redesign of the read and condition paths and earlier translation (backlog).
- **The microcode changed with it:** the register-select codes are ordered for the card's field mux (A, B, C, SP, CONST, CRMAP), and the generator forbids MARPC in a word that loads the PC (the MMU / data card's shadow PC would give MAR the old PC).

D62 — The MMU / data card as built (2026-10-26, under D55; Ken: "continue building hardware")

docs/mmu-card.md; `generators/card_mmu.py`.

- **130 ICs, 8 GALs**, through-hole; no clock of its own (CCLK, CLK and `-WSTB` from the CPU card).
- **MAR and the shadow PC are counters** (74HCT161): MARINC and PCINC count, RDST / MARPC load.
- **The TLB in seven CY7C199**, both sides (64 + 64 entries) in one array, compared by four 74HCT688s; **TLBW is written late** at the next cycle's `-WSTB`, as the register file is.
- **Faults in the order of** `translate()`, split over two GALs and checked against it for every input; a bus error is no `-ACK`, except in the acknowledge cycle (vector FF: pull-ups on D0–7).

- **Verified with the CPU card at gate level** (`generators/sim_sys.py`): the two netlists joined at J1–J3, memory and devices as bus slaves; the cross-check programs run with every instruction's state as `p32emu` and every cycle count as `p32uemu`; `make test` runs three.
- To do: a DMA master and a slow slave in the simulation (arbiter, lock, STALL), timing against the datasheets, the layouts.

D63 — The memory card as built (2026-10-26, under D60)

docs/mem-card.md; `generators/card_mem.py`.

- **48 ICs, 3 GALs**: eight banks of four AS6C4008 (D60), the boot ROM in four SST39SF020A, address buffers and data transceivers, the select, bank / `-ACK` and write-strobe GALs.
- **Jumpers:** the window (2), the banks fitted (3), the ROM (1). The card answers only in fitted banks and, for reads, the ROM; each lane's `-WE` in the bus clock's second half.
- **Verified with the CPU and MMU / data cards at gate level** (`generators/sim_sys.py --mem`), every instruction and cycle as the emulators.

D64 — The debug card: a bus tester that listens or drives, in a slot of its own (2026-10-26, Ken)

Ken: "The debug card should be listen and drive but not at the same time"; "I want to be sure the bus tester can drive all the signals to test all cards other then CPU and can listen to all signals including the CPU"; asked how it reaches J3 and the arbitration lines: "Dedicated debug slot".

- **Two modes, never both:** *listen* (every driver off; it captures signals) and *drive* (it is the master: it runs cycles and sets lines). A mode switch selects; its drivers cannot be enabled in listen mode.
- **Listen: every signal, the CPU's included** — J1 and J2 (the bus, the interrupts, the clock, reset, the lock) and **J3, the CPU ↔ MMU / data card's local link**, so the microcycle-level traffic of a running CPU can be watched.
- **Drive: every signal a card other than the CPU card sees** — J1's and J2's (it replaces the CPU and MMU / data cards as the bus master, the clock and reset, the interrupt and arbitration lines, to test memory, I/O, disk and video cards alone), and **the CPU card's side of J3**, to test the MMU / data card with the CPU card pulled.
- **A dedicated debug slot, slot 0,** on the backplane beside slots 1–2: wired to J1, J2 (with the eight radial `-BREQ` / `-BGNT` pairs as slot 2 sees them, so it can watch or play every requester) and J3. The ten general slots are unchanged. The backplane has 11 positions.
- Its other duties (D50) stay: the host link (a Pico), loading and reading memory by DMA with the CPU halted, a fast clock.
- **Decided the same day** (docs/debug-card.md): capture in **21 × CY7C199, 32 K samples** of all 162 lines, sampled at `CLK` (one a bus cycle) or at the card's own **24 MHz** (a timing mode); a trigger RAM for any pattern; a Pico 2 host link over USB; drive registers on every line it may drive, enabled only in DRIVE. **The backplane gets slot 0** (J1, J2 with the request / grant fan-in and `-BREQ0` / `-BGNT0`, J3), and **the MMU / data card's arbiter a ninth requester**, slot 0, first (Ken: "Yes, both").
- **As built the same day:** `generators/card_debug.py`, 138 ICs, 6 GALs, verified at gate level (`generators/sim_debug.py`): a running program's bus cycles captured and read back exactly; a DMA write in slot 0 while the CPU runs; alone, its own clock, a RAM word written and read back. The backplane definition (`p32bus.py`, docs/bus.md) has slot 0, and the MMU / data card's GAL ARB its ninth requester.

D65 — The I/O card as built (2026-10-26, under D50, D52; Ken: "go ahead with the I/O card")

docs/io-card.md; `generators/card_io.py`.

- **86 ICs, 16 GALs**, plus the Pico 2 W and the WIZ850io: four TL16C550 UARTs, each through a MAX232A to a DB9 plug at the front edge (D50), the 32-bit timer (P32's design: the complement counting up, the reload register on a load bus, inverting buffers), the status word, the ports on IDC headers, the DS12C887, two CH376S module sockets, the network mailbox (P32's design: two latches, GALs for the strobes and the flags).
- **Every device's interrupt level is three jumpers** (D52): a 74LS145 per device (open-collector outputs) drives the chosen `-IRQ` line; in the acknowledge cycle a 74HCT138 decodes the level, a GAL per two devices matches it against their jumpers, GAL IOPRI takes the lowest asking, GAL IOVEC drives the vector (D57). 30 jumpers in all, the window's included.
- **Every strobe in the bus cycle's second half** (CLK low), the address settled. The simulation found the clock's AS, first made in the first half, falling on a decode glitch and latching a wrong register number.
- **Verified with the other three cards at gate level** (`generators/sim_sys.py --io`): only the UARTs', the clock's, the USB modules' insides and the coprocessor's firmware are `p32io.h`'s models, behind the card's pins. `hello`, `card_test`, `io_test` and a new `iocard_test` (the timer at hardware timing, its vectored interrupt from WAIT, the status word, the clock, both USB controllers, five bus errors) run identical to `p32uemu -C`, instruction for instruction and cycle for cycle; `mbox_test` sends HELLO through the mailbox's latches. `make test` runs `iocard_test` and `mbox_test`.

D66 — The disk card as built (2026-10-26, under D26, D50, D52; Ken: "Start the disk card"; "CF-to-IDE adapters")

docs/disk-card.md; `generators/card_disk.py`.

- **Four identical ATA ports**, 16-bit PIO. **The two CompactFlash ports are right-angle 40-pin headers at the front edge, each taking an off-the-shelf CF-to-IDE adapter** with its slot facing out (Ken: every CF socket is surface-mount, so the card stays through-hole); the IDE channels are boxed headers. An activity LED per port at the front edge.
- **Two wait states on every port access** (the card's `-WAIT`): the address and chip select a clock ahead, `-DIOR` / `-DIOW` a whole clock (333 ns: ATA PIO mode 0 wants 165), a recovery clock with the write data held and the read data from a latch taken as the strobe ended. IORDY stretches the strobe. `p32uemu -C` adds the same wait states, so it stays the hardware's exact model.
- **One interrupt level for the card** (three jumpers, default 1, D52); in the acknowledge cycle the vector is window × 16 + the lowest asking port.
- **An empty port reads 00** (pull-downs on its data lines); the emulators now do the same (they read FF in the registers other than status).
- **39 ICs, 9 GALs.** Verified at gate level with the CPU, MMU / data, memory and I/O cards: `card_test`, `io_test` and a new `disk_test` (four ports, master and slave, write and read back, the vector) identical to `p32uemu -C`. The first exercise of `STALL` in the simulation.

D67 — The video card as built (2026-10-26, under D50, D52, D53; Ken: "go ahead with the video card"; "Discrete logic")

docs/video-card.md; `generators/card_video.py`, `generators/gen_font.py`.

- **The VGA timing from counters and GALs** (the backlog's open question): a 25.175 MHz oscillator, 74F163 counters for the dot, the character and the line, fast GALs (ATF22V10C-7) on the dot clock. VC 0 is the screen's line 40, so the text row and scan line are VC's own bits.
- **The RAM is 16 bits wide** (two CY7C199), shared in four 2-dot slots of each 318 ns character time: the text cell, the bus's low and high halfwords, the overlay byte. A character's font and overlay bytes are ready together as its dots begin.
- **The bus waits by handshake, not a fixed count:** `-WAIT` until the dot clock's side has done the access (two flip-flops in, one back on `-CLK`). One or two wait states at 3 MHz, right at any bus clock. The registers answer at once.
- **The overlay moves to** `0x1000` (`0x1000`–`0xA5FF`). At `0x8000`, as first planned, 38,400 bytes run past the card's 64 KiB (the emulator's screen dump read past its array). Nothing else used the offset.
- **PS/2 in discrete logic** (Ken): per port a GAL state machine, a 74HCT595 to receive, a 74HCT597 to send (parity from a 74HC280), the clock held low while a byte waits; sending holds the clock five line ticks (≥ 100 µs). An empty data register reads 0, as in `p32io.h`.
- **The font** is drawn in `gen_font.py` (8 × 8 glyphs, each row on two scan lines), so nothing is copied from another font; the ROM image is the whole SST39SF010A.
- **88 ICs, 18 GALs** (the architecture's estimate was ~50).
- `p32uemu -C` **models the card's timing exactly**: the two oscillators' schedule as the gate-level simulation runs them, the handshake's flip-flops and the raster (`p32io.h` `vid_hw_*`), so the wait states and the status register agree cycle for cycle. The registers answer bytes only, in the emulators too.
- **Verified at gate level** with the CPU, MMU / data, memory and I/O cards: `vidcard_test` identical to `p32uemu -C` (every instruction, every cycle, all 614 RAM accesses' wait states); the frame the card then draws, dot for dot the picture its RAM holds, the syncs to VGA's timing; `ps2_test` (keyboard bytes, the vectored interrupt, the mouse's reset, a command).
- **Every programmable part programs on the T48** (Ken: "be sure all devices can program on t48"): the ATF22V10C, the SST39SF010A and the SST39SF020A are in minipro's T48 list (DIP). **The GALs' JEDEC files** were missing for every card: `generators/p32jed.py` now writes all 83 (hardware/gal/<card>/), reads each back through a model of the 22V10's hardware and compares it, exhaustively, with the GAL's spec; and, Ken approving the download, GALasm 2.1 (github.com/daveho/GALasm, kept outside the project) assembled the same equations to the identical fuses, all 83. `make test` keeps the files current. Found on the way: `Gal.emit_pld()` marked an inverted output's polarity twice, which in GALasm cancels; it now hands over to the checked writer. The flash images: the font file is the whole chip; the microcode's and the boot ROM's are shorter, so `minipro -s` (hardware/gal/README.md).

D68 — The backplane's netlist (2026-10-26, under D49, D50, D64; Ken: "go ahead with the backplane netlist"; ACCEPTED by Ken 2026-10-27: "accept D68 as proposed")

docs/backplane.md; `generators/card_backplane.py`. The choices below were proposed for Ken to confirm or change before layout; he accepted them all.

- **Eleven positions**, slot 0 (debug) then 1–10, each three female DIN 41612 C 3 × 32 sockets; every pin on `p32bus.busnet()`.
- **J1 / J2 spare pins bused to every slot** (22, on J2), so a later signal needs no new backplane (P8X rev D, P32 did the same); J3's spares not connected. `busnet()` now names them (`SPARE_J2A19`, ...).
- **ATX power entry**: the 24-pin header (5 × +5 V, PWR_OK to PWROK), PS_ON# to a **POWER** jumper or switch, a **+5V** LED; 3.3 V, standby and ±12 V not used. Two 1000 µF at the entry; per slot 3 × 100 nF and a 10 µF.
- **Pull-ups by a rule**: every J1 / J2 line that no card drives totem-pole gets one, and only those; the generator reads every card's drivers and checks it. `-WAIT` 680 Ω and `-ACK`, `-LOCK` 1k (rise within half a 5 MHz clock through ~200 pF), `-IRQ0–7` 2.2k, PWROK 10k; **10k on D0–31, A2–31, BE0–3,** `-MRD`, `-MWR`, `-IACK`, three-state lines that would otherwise float between masters, or with the MMU / data card out, or (D) on every cycle that is not a read or a write.
- **AC termination of CLK and** `-CLK` **at slot 10** (100 Ω + 150 pF), the far end from the CPU card; values to settle on the bench.
- **Checked against every card in every slot it may use** (CPU 1, MMU / data 2, debug 0, memory / I/O / disk / video 3–10): each connector pin a card uses is wired there. 33 sockets, 101 parts.
- **Left for the layout:** the board (9U × about 320 mm), its layers (6 or 8), mounting holes, the silkscreen (slot names, POWER, +5V, cottageworker.com), and the power budget that picks the supply.

D69 — The power budget (2026-10-27; Ken: "go ahead with the power budget")

docs/power-budget.md, generated by `generators/p32power.py` from every card's netlist, so it follows the cards; `make test` fails if a part has no supply current or the document is stale.

- **Per part, typical and maximum** +5 V current: the datasheets' figures where found (the ATF22V10C 85 / 130 mA, the CY7C199 155 mA at its fastest, the AS6C4008 30 / 60 mA, the SST39SF0x0A 10 mA read, the TL16C550, MAX232, 74F counters, the WIZ850io), marked *estimate* otherwise (CMOS logic by its switching, the oscillators, the Tang Nano 20K and the Picos, which are to measure). Only one memory bank in the machine is active at a time; the boot ROM only while booting.
- **The machine:** about **13–14 A typical, 24–26 A at most** on +5 V (one to four memory cards), and up to 2 A more for the drives on the supply's own cables. The largest card is the CPU card, 5.9 A at most.
- **The supply: an ATX unit judged by its +5 V rating, at least 20 A**; 30 A covers the worst case. The ATX header's five +5 V pins (about 30 A) and each card's 32 VCC pins carry it.
- **The GALs are half the current** (83 × 85 mA = 7.1 A typical). The quarter- and zero-power 22V10s are slower; whether some GALs can use them is left to the timing pass.

D70 — The Tang Nano 20K's footprint and pins from Sipeed's datasheet (2026-10-27; Ken sent "Sipeed Tang nano 20K Datasheet V1.3": "nano foot print")

The CPU card's ALU module (D21) had been drawn before its datasheet was in hand, its pin map marked "to verify". Checked against Sipeed's datasheet v1.3 ("Pinout", "Dimension Information"):

- **The rows are 20.32 mm (0.8") apart, not 17.78 mm:** the footprint (`gen_footprints.py`) would not have taken the module. The first pins sit 2.64 mm from the USB-C end, the last 3.14 mm from the other; 54.04 × 22.55 mm.

- **The power pins were wrong:** the placeholder map had 5 V and GND on the left row's first two pins. They are the right row's first two (5 V, GND); the left row ends 3.3 V, GND; the right row has GND, 3.3 V as its 15th and 16th pins. The 3.3 V pins are the module's own output, unused.
- **The 29 signals on 29 of its 34 free FPGA pins** (`p32net._NANO_PADS`, with each GW2AR-18 pin and what else it reaches on the module): the FD bus where only the LCD connector, the microSD slot and the HDMI EDID lines also go; the six latch selects and result clocks on the LED pins; the function on the audio amplifier's and the WS2812's inputs; GO and the ALU's reset on two BL616 high-speed SPI lines, the only pins left, which the debugger firmware does not drive (to confirm on the bench). Five pins spare.
- `fpga/alu/p32alu_nano.cst`, the Gowin constraints, is generated from the same map (`generators/gen_nano_cst.py`), and `make test` keeps it current.
- **To check on Sipeed's schematic:** whether the module's 5 V pin is isolated from its USB-C's VBUS (if not, the USB cable must not be plugged in while the card is powered).

D71 — The timing pass: a static timing analysis against the datasheets (2026-10-27; Ken: "go ahead with the timing pass then schematics and layout")

docs/timing.md, generated by `generators/p32timing.py` (the analysis), `p32delays.py` (every part's switching times from its datasheet, read with Ken's approval to fetch them) and `p32timing_config.py` (the clocks, the phase rules and every exception, each with its reason). `make test` keeps the report current.

- **How:** the cards' netlists joined over the backplane; every clock pin traced to its oscillator, CLKGEN's eight phases (and the video card's eight dots) known; the machine timed as a read, a write and an internal cycle (case analysis); worst case (datasheet maximum, -40..+85 °C) and typical. 2,411 endpoints on the machine.
- **Fixed on the way (identical logic):** the PC's, MAR's and the shadow PC's 74HCT161 chains had the first counter's ENT on the increment (PCINC, MARINC), so the slow ENT → RCO ripple (7 × 40 ns) started only when the late increment arrived. Now the datasheet's synchronous cascade: the first ENT high, the increment on every ENP; the ripple settles from the count, the increment needs only its setup.
- **The video card's dot clock passes** at worst case (tightest: 37.5 of 39.7 ns). **The machine at 3 MHz does not**, even at typical delays; what fails is in docs/timing.md §2, chiefly:
  1. the I/O card's byte-device reads (UARTs, RTC, mailbox): the strobe starts mid-cycle and the card has no `-WAIT`, so the device's access, the transceiver, the steering and the destination's setup must fit in half a cycle (a 2.2 MHz bus at typical delays);
  2. device-card registers written at mid-cycle (the I/O card's latches, the video card's registers): the address arrives late (the MMU / data card's address mux selecting the IACK level, through two AND gates), then the card's compare, GALs and the 245's enable (58 ns);
  3. at worst case also the abort loop (the TLB, the strobe, the slave's decode, `-ACK`, FAULT, ABORT, every destination's write enable), the interrupt request into the microaddress, the comparator tree into COND.
- **The debug card's 24 MHz timing mode does not meet its clock** (about 20 MHz at typical delays): its sample clock reaches the capture counters through four gate levels. 12 MHz would be safe.
- **For Ken to decide** before the schematics and layouts are final: how to fix 1–3 (see the conversation of 2026-10-27). Decided: D72.

D72 — 3 MHz at the datasheets' worst case (2026-10-27; Ken: "3 MHz worst case", "One wait state", "Take writes at cycle end", "Swap the parts (worst case 3 MHz)")

The fixes for D71's failures. docs/timing.md now: **the machine meets 3 MHz at worst case** (every path; the tightest needs a 40.4 ns oscillator phase of the 41.7) and about 4.9 MHz at typical delays (25.4 ns).

- **The I/O card's byte devices take two clocks** (UARTs, clock, USB, mailbox, ports, ID): GAL IOSTB's W1 register and `-WAIT` (a spare 74HC05 gate) hold the first; `-RD` runs to the cycle's end, `-WR` a whole clock with half a clock of hold; GAL NSTB's latch clocks, AS and mailbox flags act in the second. `p32uemu -C` adds the wait state, so the gate-level `iocard_test` (1,178 cycles) and `mbox_test` agree with it cycle for cycle.
- **Device registers take writes at the cycle's end:** the I/O timer's reload latch (its clock now rises with CLK; `p32uemu -C` takes a TRELOAD write after that edge), the video card's cursor and PS/2 send registers (GAL VWR's new KCMDK / MCMDK clocks; the PS/2 GALs keep the level).
- **The I/O timer's counters are 74F163s:** the eight-stage carry to ALLONES must settle within a clock of a load.
- **Faster parts on the critical paths** (same pinouts; all in production in DIP, TI's datasheets in `p32delays.py`):
  - CPU card: pipeline register, register-file write latches and the IRQ synchroniser 74ACT574; register-file read buffers and the address adder onto R 74ACT541; register-file address muxes 74ACT157; IR 74ABT377; opcode mux 74F153; R destination decode 74F138; the local link's gates 74ACT08; the pending ORs 74ACT32.
  - MMU / data card: MAR 74F161; TLB index muxes 74ACT157; TLB write data and the address onto the bus 74ACT541; TLB tag compare 74F521 (the '688's pinout); TLBHI and TLBLO 74ABT377; the IACK select's gates 74ACT08.
  - I/O card: the acknowledged level's decoder 74F138.
  - **74ABT377, not 74ACT377:** no 74ACT, 74AC or 74F 377 is made in DIP any more. The ABT part's high output is TTL-level (about 3 V), which every input it drives (HCT, ACT, F, GALs, the flash) reads as high.
- **The comparator in GALs:** no faster '85 is made in DIP, and the 74HCT85 tree took 150 ns. GALs CMP0–CMP7 give each nibble's GT and LT (CMP7 the signed top nibble too), CMPH the high half, CMPF the low half and EQ, LTU, LT: about 40 ns. 14 × 74HCT85 out, 10 GALs in; GAL ABDEC loses its sign inversion.
- **"Any interrupt pending" in GAL ANYP** (10 ns, against the 74HCT148's GS, 49): it is on the path into the next microaddress. The 148 still gives the level's number.
- **The cost:** the CPU card 193 ICs (34 GALs); 94 GALs in all, every fuse file checked through the fuse map and identical to GALasm's. The power budget rises to about 16 A typical, 28.5 A at most on +5 V: a supply rated **at least 25 A** at 5 V; the backplane's worst case is near the ATX header's 30 A. The ACT parts' fast edges need care in the layout (series resistors on long lines, as the backplane's clock termination).
- **Not fixed:** the debug card's 24 MHz timing mode still does not meet its clock (about 20 MHz at typical delays): its sample clock passes four gate levels. For Ken: run it at 12 MHz, or rework its clock path.

D73 — The debug card's timing mode at 12 MHz (2026-10-27; Ken: "run the debug card at 12 MHz")

At 24 MHz its capture path did not meet its clock (D71, D72). Now:

- **The sample clock is the oscillator halved:** GAL DBCLK's new register T12 toggles on every 24 MHz edge; the mux takes it in place of the raw oscillator. The card-alone bus clock (3 MHz) is unchanged.

- **The clock path is faster:** the sample-clock mux 74ACT157, its gates 74ACT00, the counter clock's inverter 74ACT04, the STEP flip-flop 74ACT74 (it cleared itself within a 24 MHz edge).
- **The timing analysis** treats the sample latches as a logic analyser's: they take the lines asynchronously, so nothing into them is timed; from them on, everything is. The card now passes at worst case (41.0 ns of the oscillator's 41.7).
- Four samples per microcycle instead of eight; 32 K samples cover about 2.7 ms of machine time.

D74 — A clock jumper: 4 MHz besides 3 (2026-10-28; Ken: "go ahead with the clock jumper", the timer dividing by 4)

The cheapest speed-up in speed.md: at the parts' typical delays every path allows 4.9 MHz (D72). Now:

- **The CPU card** gets a second oscillator, **32 MHz**, in a socket, and a 74ACT157 that puts it or the 24 MHz one on OSC. The **CLOCK** jumper (JP1, at a board edge, labelled `3 MHz` open / `4 MHz` fitted) selects it. Everything on the card still comes from OSC through GAL CLKGEN, so the microcycle is OSC / 8 either way. Move it only with the power off.
- **A new bus line,** `-FAST` (J2 A19, a spare until now): the jumper grounds it; the backplane's 10k pull-up (R6) holds it high when open or when no CPU card is fitted.
- **The I/O card's timer** (GAL TIMCTL) divides by 4 instead of 3 when `-FAST` is low, so it ticks at **1 MHz at both clocks** and the kernel is unchanged. 4.5 MHz (36 MHz) was ruled out: no whole divider gives 1 MHz.
- **What is promised:** 3 MHz at the datasheets' worst case, as before; 4 MHz at typical delays, with every path meeting its edge and 19% in hand (25.4 ns of the 32 MHz oscillator's 31.25; timing.md times it as its own run). Most boards will run at 4 MHz at room temperature; a bench run of the test programs with the debug card listening is the way to know a given machine does.
- **The debug card is unchanged:** alone it makes its own 3 MHz clock, and its oscillator also gives the 12 MHz sample clock (D73).
- **Checked:** `p32uemu -C -F` models the ÷4 timer; the gate-level CPU, MMU / data, memory and I/O cards with `-FAST` low run `iocard_test` to the same state at every instruction and the same 1238 cycles.
- **The cost:** 2 ICs on the CPU card (195), a jumper, a resistor on the backplane; about 15 mA.

D75 — The disk card's DMA engine, built in now (2026-10-28; Ken: "Although we won't start with the need for DMA i do want it all designed in so new hardware does not need to be ordered"; per-sector pacing, multiword DMA wired; 32-bit buffered cycles)

The second step in speed.md: the CPU's disk copy loop is 13–21% of the measured sessions' cycles. D50 planned the disk card "PIO first, DMA-capable", but as built (D66) it had no master logic: its DMARQ lines were pulled down and `-DMACK` held off. Now the engine is on the card, and P32W Unix keeps its PIO driver until it is wanted:

- **Registers** in the card's window: `EF40` DMAADR (the physical address), `EF44` DMACNT (words, 16 bits), `EF48` DMACTL (RUN, DIR, the port, IE, MODE; read back BUSY, DONE, ERR). The card's revision becomes 1; the engine is its fifth interrupt source (status bit 4, vector window × 16 + 4).
- **Per sector** (Ken's choice): the driver starts the engine when the drive has a sector (DRQ); the engine does the drive's PIO itself, so every drive and CF card works. Starting it is a few register writes, against about 600 instructions of copy loop a sector.

- **32-bit words, buffered** (Ken's choice): two halfwords into a word buffer, then one bus clock as master; about 11 clocks a word, the bus the engine's for one of them.
- **Sharing:** the engine's port accesses use the CPU's sequencer and port circuits; a CPU access that needs the card's byte bus meanwhile waits on `-WAIT`. The DMA registers sit on D directly.
- **Multiword DMA wired, reserved** (Ken's choice): DMARQ and `-DMACK` of every port reach a GAL; MODE and the selected DMARQ reach the sequencer. The drives' own DMA is a reprogramming of socketed GALs, not a board.
- **The cost:** 41 ICs (6 GALs, 12 × 74F163, 8 × 74HCT574, 15 buffers): the card goes from 39 ICs to 80, on its 9U × 340 mm board; its supply 1.8 A typical, 2.9 A at most. The machine's worst case with four memory cards is now 30.1 A, at the ATX 24-pin header's 30 A (typical 16.8 A).
- **Timing:** the engine's own bus cycles are a run of their own (the disk card master, the memory, I/O and video cards answering): 261.5 ns of the 333 ns clock at worst case, 144 ns typical (so the 4 MHz jumper holds for them too). In the CPU's cycles its drivers are off (case analysis); the machine stays at 40.4 ns worst case.
- **Checked:** `diskdma_test` at gate level (the engine a master through the MMU / data card's arbiter into the gate-level memory card): a sector in and out, the CPU using the card meanwhile, the interrupt, a bus error; the same program passes on `p32emu` and `p32uemu -C`, which now model the engine, and the sector it writes is the same on all three. `disk_test` stays cycle-exact against `p32uemu -C`. The simulation now models the backplane's holds on `-MRD`, `-MWR` and `-IACK` (a device master drives no `-IACK`).
- **Correction:** the speed analysis said the disk card was already able to be a master; it was not, until now.

D76 — Stepping the machine's clock, from the CPU card or the debug card (2026-10-28; Ken: "Design in and also since we have spare bus lines this can travel from debug card to cpu via the bus"; "keep controls on the cards")

STEP (D50) steps a microcycle or an instruction with the oscillator running. Now the clock itself can be stopped and advanced **one oscillator edge -- one of the eight phases -- at a time**, so every clock and bus cycle can be watched phase by phase. Nothing in the design prevents it: every part is static (no DRAM), and every device with its own clock is reached through `-WAIT` or polling.

- **The CPU card's oscillator is chosen by GAL CKSEL** (ATF22V10C-7, in place of D74's 74ACT157): the debug card's `XCLK` while `-XCLKEN` is low; else the **CLOCK STEP** button when the **CLOCK** switch (front edge, RUN / STEP) is at STEP; else the 32 or 24 MHz oscillator by the speed jumper. The button is debounced as RESET and STEP are (RC, 10 ms, a 74HCT14 gate that was spare). The step edge bypasses GAL PANEL, which is itself clocked by OSC.
- **Two bus lines, J2 A20 and A21** (spares until now): `XCLK` and `-XCLKEN`, driven by the debug card's GAL DBCLK. With XCLKEN (CTRL1 bit 7) the card supplies its oscillator halved (12 MHz: a 1.5 MHz machine) while CLKRUN, or one pulse per host STEP command. They are the debug card's own lines, driven in LISTEN too, so the host can capture a program while stepping it. With no debug card the CPU card's resistors hold `XCLK` low and `-XCLKEN` high.
- **A plain multiplexer:** the source is changed with RESET held (a change mid-phase can cut a phase short). A glitch-free handover would need edges on both clocks, which a stopped one does not give.
- **What runs on regardless:** the video card, the UARTs' baud rate, the clock chip, the USB modules, the Pico and the drives keep real time; their interfaces wait or are polled, but a UART receiving at full speed overruns. The I/O card's timer counts bus clocks, so time slows with the machine.

- **Checked at gate level** (`sim_debug.py xclk`, in `make test`): with XCLKEN off the CPU card keeps its own oscillator; on, ten host STEPs advance CLKGEN's phase by exactly ten, nothing moving between; with CLKRUN, a phase every two of the debug card's periods; the CLOCK switch at STEP, one phase per CLOCK STEP press.
- **The cost:** a GAL for a 74ACT157 on the CPU card (still 195 ICs, 35 GALs), a slide switch, a button, three resistors and a capacitor; on the debug card nothing but DBCLK's new equations.
- **Also (Ken, 2026-10-28): a POWER LED on every card**, green, from +5 V through 1k (about 3 mA), in the LED bank at the front edge, labelled POWER; added by `Card.finish()` so no card can lack it (two parts a card; the backplane keeps its own +5V LED). The machine's worst case with four memory cards becomes 30.3 A.

D77 — Every board on 6 layers; the first layouts (2026-10-29; Ken: "I am ok going to six layers for all boards")

- **6 copper layers on every board**, replacing D19's 4 by default: F.Cu signal, In1 GND plane, In2 signal, In3 signal, In4 VCC plane, B.Cu signal. `LAYERS` in `p32kicad_pcb.py` now defaults to 6, and a card may still be listed with 4 or 8. The backplane, when its generator is written, follows this too.
- **Why:** the first routing of the memory, I/O and MMU / data cards on 4 layers (2026-10-28/29, two tries each, up to 2 h a try) left 25, 4 and 5 pads unconnected, and the MMU card one clearance fault. Two more signal layers give the router room, so no board waits on hand routing; at JLCPCB, 5 boards at 366.7 × 340 cost $171.70 at 6 layers (D49).
- **Finishing a routed board in Freerouting does not work:** a routed board exported from KiCad 10 and read back by Freerouting 2.5 counted 915 of the I/O card's 1,137 connections unrouted, with the tracks locked or not and with or without our DSN edits, so it routed the board again from scratch. A board that a fresh route leaves short gets another try or hand work.
- **The memory card's own grid** (`place_rows` in `card_mem.py`): the memory chips four across and nine down, a bank to a row (the boot ROM the last) and a byte lane to a column, so each address line runs straight down a column and a lane's data lines down its own; the logic in two columns beside the bus connector, the three address buffers in rows 2, 4 and 6 with empty cells around them for their fan-out. The general packer had staggered the rows and scattered the buffers; MA8 alone left 19 pads open. The estimated wire length falls from 50.5 m to 42.1 m.
- **The layout rules so far:** the I/O card's Pico with its USB port at the front edge; the revision printed as REV 1 (not REV A) beside the board's name and cottageworker.com; jumpers in a row along the bottom edge; the cable connectors (DB9s, VGA) at the front edge facing out.

D78 — A dedicated 5 V supply on screw terminals; a supervisor makes PWROK (2026-10-29; Ken: "I am going to use a dedicated +5V 30A supply ... So I think the bus card just needs heavy duty screw terminals"; agreed to the rest; "assume supply also has +12v or there is a second supply for +12")

Replaces D68's ATX power entry, and the backlog's second ATX entry (the worst case, 30.3 A, was at one 24-pin header's limit).

- **The supply:** a dedicated +5 V switching supply, **30 A at least, 40 A recommended** (a known brand, e.g. Mean Well LRS-200-5). The machine with four memory cards is 16.9 A typical and 30.3 A at its datasheet worst case (D69), and a cheap supply sags near its rating. It is switched on the mains side (a switch or a switched, fused inlet on the case).
- **The entry: two pairs of M4 bolts** (P1 / P2 by slot 0, P3 / P4 by slot 10; +5V and GND), each a plated 4.3 mm hole stitched with vias to its plane, a ring lug on 10 AWG wire under the bolt's head. Both pairs from the same supply, so

the current splits between the ends and along the planes. (Two separate supplies would need a split +5 V plane and their PS_ON# / PWR_OK combined; one supply avoids all that.) A 1000 µF bulk capacitor at each pair.

- **PWROK from a TL7705A supply supervisor** on the backplane (U1, DIP-8; TI SLVS028K): SENSE on +5 V (threshold 4.55 V), its open-collector `RESET` (pin 5) on PWROK with the existing 10k pull-up; low until +5 V has been good for 1.3 × 10⁴ × CT (10 µF: about 130 ms), low again on any dip. `RESIN` tied high, REF with the datasheet's 100 nF, `RESET` (pin 6) unused. The CPU card's reset (GAL PANEL: `RESET = RESBTN or not PWROK`) is unchanged; PWROK is now the backplane's line, not the supply's.
- **Gone:** the ATX header (P1) and the POWER jumper (JP1, the supply's PS_ON#); the ATX24 device stays in the library unused.
- **+12 V is not on the backplane.** The disk card's IDE hard drives take +12 V and their +5 V from the supply directly, from its own 12 V rail or a second, 12 V supply; CompactFlash needs 5 V only.
- **Updated:** `card_backplane.py` (its netlist still passes every check: 108 parts, every card fits its slots), docs/backplane.md §2, docs/boot.md, the bus's PWROK row (`p32bus.py`, docs/bus.md), docs/power-budget.md (`p32power.py`: the supervisor's 3 mA, the supply paragraph), the backlog.

D79 — The backplane's board, placed and routed by its generator (2026-10-29; Ken: "go ahead with the backplane layout generator")

- `generators/p32backplane.py` builds `hardware/backplane/kicad/ backplane.kicad_pcb` from `card_backplane.py`'s netlist, placed **and routed**, as P32's backplane was: no autorouter. 376 × 366.7 mm, 6 layers (D77); 0 DRC violations, 0 unconnected.
- **The sockets** are turned 180° so that pin 1 is at the bottom, where the cards have it; their heights come from the card generator's own placement of J1–J3. To verify against a real card and socket before ordering.
- **Straight channels:** each bused net is one track between two pin rows on its column's layer (a on F.Cu, b on In2, c on In3), with a 1.27 mm stub from each pin. The request lines run their slot 0 / slot 2 C pins as one channel, carried on to the requesting slot, where a via and a B.Cu riser between two pin columns reach its A16 or A17.
- **The netlist changes this needed:** the 10k bussed networks are grouped by position, eight consecutive pins of one connector column each (`pull_networks()`), so a network stands beside the socket rows with every pin on its own net's row and the nets' channels simply run on to it: 12 networks of 10k instead of 9, some resistors unused. Eight M3 mounting holes on GND for the subrack (backplane.md §4 asked for them). 119 parts.
- **Placement:** the networks, pull-ups and termination in a field past slot 10, each packed as far left as its courtyard allows; the TL7705A at the left end on PWROK's row; the bolts, bypass and bulk capacitors in the gaps between the J1, J2 and J3 sockets.

D80 — Placement by function; a direction for each routing layer (2026-10-29/30; Ken: "Does it make sense to rearrange chips by functional proximity", "a lot of routers use some planes for mostly horizontal runs and other planes for mostly vertical runs", "consider starting over for each board")

- **Why:** the general packer lays the parts out as one chain, ordered by shared nets, in rows that snake left and right. The buses that touch nearly every chip (D, the byte bus, MAR, R) swamp that ordering, so a function ends up across the whole card: on the I/O card UART 1, its MAX232 and its DB9 spanned 299 × 165 mm, the bus decode 208 × 258 mm. And every row stopped short of the whole front strip by the widest front part (the Pico), so the right quarter of the card stayed empty.

- **A row per function**, the card's own (`place_rows`), placed afresh:
  - **I/O card:** the address decode by J1's address pins; a row per byte lane (its 245, status 541, the timer's 540s, 374 and 163s share the lane's D and load-bus lines), lane 3 to 0 so the counters' carry runs down; each UART's 16550, MAX232 and pump capacitors at the end of the row beside its DB9; the acknowledge logic by J2; the mailbox latches, the 3.3 V regulator and the Pico's pull-ups beside the Pico; the nine 74LS145s in the bottom row, each above its three level jumpers. The port headers J8 / J9 now stand above the DB9s (front order J10, J11, J8, J9, J4-J7, A2, A1), which brings the DB9s level with the UARTs.
  - **Memory card:** its grid as before, the columns spread across the card to 300 mm (channels between the lanes).
  - **MMU / data card:** a row per register, each with its multiplexer and its drivers onto R (the R bus reaches every row, a register's own lines do not), every slice low bits to high from left to right: the A / D drivers and the load steering by J1, the physical address and BADADDR, the TLB, TLBHI / TLBLO / ASID, MAR and the shadow PC by J3. A first try with all the drivers onto R in rows by J3 was 118 m.
- **Rows spread across the card** (`place_spread`): a row's gaps widen until it ends at `place_right` or short of the front-edge part beside it (the Pico, a DB9), worked out at the row's final height. Rows packed at 2.5 mm left the vertical layers nothing but the single gaps between DIP pins to cross them; spread, the I/O card's lane rows have 10 mm channels. The wire-length estimate rises a few per cent; the router gains the room.
- **Swaps only within a row** (`place_swaps = "row"`, the I/O and MMU cards). The swap pass (same-footprint pairs, whenever the estimate gets shorter) had carried lane 3's 374 into the decode row on the I/O card, and the lane's load bus stayed unrouted; free swaps also scattered the MMU's registers. The estimate prefers free swaps (I/O 64.5 against 69.8 m, MMU 102 against 105 m); the router did not.
- **The placer's options** (`p32kicad_pcb.py`): `place_free` (each row packed at its own parts' widths, no columns), `">"` in a row (the rest against `place_right`; `">276"` against x = 276 mm, a row beside the Pico), `place_spread`, `place_swaps`, and `place_right` on a grid with columns (the columns spread to it).
- **A preferred direction for each signal layer**, written into the DSN by `kicad_tools.py export_dsn`: F.Cu vertical, In2 horizontal, In3 vertical, B.Cu horizontal, at the same cost on every layer, a track against its layer's direction at `KT_AGAINST` times the cost (default 1.5). Freerouting 2.5 by itself alternated only weakly (a third of the copper at 45°) and kept off the outer layers (3 % of the copper on F.Cu and B.Cu); with the rules one pass on the I/O card left 250 open instead of 392. The cost was measured, one DSN routed several ways: memory card none 14 left open, 3.0 5, **1.5 none**; I/O card none 19, **1.0 none**, 1.25 21, 1.5 24 to 38 (three tries), 2.0 19. One sample each, and tries vary widely (the memory card at 3.0: 30 and 5), so the queue alternates the cost between tries. At 1.0 a direction costs nothing extra: what the I/O card gained is the equal cost on all four layers, i.e. the outer two in use.
- **Freerouting stops by itself** when ten passes bring no improvement; after the first few passes it falls into a cycle (the memory card: 30, 39, 137, 74, 30, ...) and the result is fixed. A board that falls short is helped by another try (a new DSN order) or another cost, not by more time.
- **The results:** the memory card 0 unconnected, 0 DRC violations; the I/O card 0 unconnected, its 14 violations the cosmetic ones (silk at the board edge and over copper, the P32W library not registered); the MMU card 0 unconnected, 0 violations (its fourth try at 1.0; the first left one connection open, -ACK, at 1.5 13 and 4).
- **Fixed on the way:** `build_board.sh` falls back to the system's Java (25 or later) when the private JRE is missing (it was: Temurin 27 is installed); the heal step (`heal_unconnected`) no longer drops a via in or beside a keepout (it had put one by the Pico's antenna).

D81 — Schematics for every card, wired where local; the other four layouts (2026-10-30; Ken: "lets generate nice schematics for all cards in kicad and a

pdf", "for schematics where appropriate show inter chip connections and not just labels", "lets also start routing of unrouted cards using all the steps and rules for the three that worked")

- **Every card has a KiCad schematic**, in its board's project (`hardware/<card>/kicad/<card>.kicad_sch`), written by `p32kicad_sch.py` and built with `sh generators/build_schematics.sh`, which also plots `<card>-schematic.pdf` beside each board and joins them into `docs/pdf/p32w-schematics.pdf` (97 pages). All eight pass KiCad's ERC with 0 violations, and KiCad's own netlist of each equals the canonical one (the CPU card: 765 nets, 4,313 pins).
- **A sheet a function** (`generators/p32sheets.py`): each card lists its functional groups (`c.sheets`); the backplane connectors and the power (decoupling, bulk, the POWER LED) get sheets of their own, and a part on no sheet, or on two, is an error. The CPU card has 17 sheets, the MMU card 13, the I/O card 12, the debug card 12, the video card 10, the disk card 9, the backplane 9, the memory card 7.
- **Wires where a schematic would draw them:** a net is a wire when all its pins are on the sheet and it is not power, not a backplane signal, not a carry chain between identical chips, and not part of a bus (five or more nets joining the very same parts: a byte between a RAM, its latch and its driver). Everything else keeps its net-name label, as on a hand-drawn schematic of a bus machine. ELK (`elk_layout.js`, elkjs) routes the wires; the parts stay in the sheet's rows, each row cut into columns no taller than an A3 page, left to right (ELK's partitions, with the connected components kept together). A sheet whose wires cannot be put on the 2.54 mm grid without two nets touching, after six spacings, falls back to labels (none does now). A free ELK layout had strung the parts into one tall column with wires around the page.
- **Symbols:** a part of another logic family (74ACT574, 74F161, ...) takes its 74HCT twin's layout when its pins are named alike; any other part gets its inputs left and its outputs right, in natural order, a connector's pins half on each side.
- **Fixed on the way:** the DE-15's shell pads are `SH` in KiCad's footprint (the netlist had `0`), and its pin 1 is 6.83 mm from the edge, not 3.03 (KiCad's "EdgePinOffset" is the row nearest the edge, pins 11-15; with pin 1 at 3.03 that row stood off the board); the mini-DIN-6 footprint (Same Sky MD-60SM, from YACC1's video card) is generated into `hardware/lib/P32W.pretty`, so the video card's board builds.
- **The other four layouts**, placed as D80 did: the disk card (each port's transceivers beside its header, the DMA engine's rows between the ports, the activity LEDs labelled DISK 0-3, the jumpers along the bottom edge), the video card (rows in the picture's order, the VGA drivers beside the VGA socket, each PS/2 port beside its jack; the sockets start lower down the front edge, `front_top`), the debug card (two capture slices a row, by the connector their signals come from; the Pico and the mode switch at the front edge). The CPU card's front panel is now listed and labelled (D48, D55, D76): RESET, STEP, the RUN / STEP and INSTR / MICRO switch with its levers across the edge, the CLOCK switch, CLOCK STEP, and the ALU module's USB-C at the edge; the CLOCK jumper at the bottom edge. Its 195 chips fill the card at the packer's tightest spacing; it is routed as the baseline, and its own rows are to come. The debug card's mode switch is labelled MODE, LISTEN / DRIVE: which end is which is to check on the part.
- **Tools:** Node and elkjs (`cd generators && npm ci`) and poppler (`pdfunite`), from Homebrew.
- **Routing**, every card on 6 layers, 0 DRC violations on each but the I/O and video cards' cosmetic ones, each matching its schematic (KiCad's parity check): the disk card 2 unconnected (the IDE ports' activity lines along the front edge; a 12 mm channel there routed worse), the video card 1 (its DE-15 fixed), the debug card 15 (CA2, a capture address bit to all 24 SRAMs, the net every try left; its rows put the bus side left and the capture side right, after two whole slices a row stalled at 94), the CPU card 29 on three nets (IR15, NSTEP2, ABORT_N; the general

packer, its 195 chips at the tightest spacing). What is left is a handful of tracks by hand, or more tries. The debug and CPU cards' passes take 15-40 minutes, so their tries ran without the time limit (a router stopped by it saves nothing).

D82 — The last connections: what more tries, layouts and 8 layers did (2026-10-30; Ken: "keep trying routing, look at possible chip rearrange by function and consider 8 layers if required")

Every card is routed on 6 layers (D81); four leave a few connections. This records the attempts to close them, so that the next one starts from here.

- **More tries** (each kept only if it beats the board in place): the disk card 2, 25, 25 and then **0** against its 2 (done); the video card 1, 5 and 16 against its 1; the debug card 28, 65, 35, **1** and then **0** (done); the CPU card **18** against its 29 (kept: LR21, NSTEP0, NSTEP1, ABORT_N left). The results swing widely from try to try, so more tries do pay.
- **8 layers** (`P32_LAYERS=8`: six signal layers, GND on In1, VCC on In6, the direction rules alternating over all six): the debug card stalled at 167 (whole local-bus and capture-address lines left), the CPU card at 60 (its 6-layer best is 29). Both fell into Freerouting's cycle after a fast start (the CPU card 145 after two passes, against 255 on 6). Not adopted: the limit is the router on these dense buses, not the layers.
- **The CPU card by function** (D80's method: the control section on top, the datapath in byte lanes, two functions a row, the R logic by J3): 147 m of wire against the general packer's 128 m, and behind it in routing from the first pass (257 open after four passes against 85). Reverted: its three-bus datapath, every bus reaching most rows, routes better packed by connectivity.
- **The debug card's SRAMs at the right end of each row** (the capture address's fifteen 24-pin lines in the clear strip by the front edge): 31 (the local bus left instead), against 15. Reverted.
- **The disk card's 12 mm channel along the front edge:** 20 and 18, against 2. Reverted (D81).
- **What the remaining connections are:** a few high-fanout nets each (the debug card's CA2, the CPU card's IR15, NSTEP2, ABORT_N); the disk and video cards' three are short. Hand-routing in KiCad is the likely end; more tries cost about an hour (debug) to three (CPU) each.
- **Why the router stalls** (a search, 2026-10-31): two open Freerouting bugs, both in 2.5.0. Issue #1017: it trades the last incomplete connections back and forth between neighbouring nets although a legal route exists (the cycles in every log here; the fix, PR #1021, is aimed at 2.6). Issue #1054: from pass 3 on, a route rejected for a clearance fault leaves the connections it ripped up unrouted, unless `router.strict_drc` is on (its fix, PR #1055, is unmerged). So more layers or more placement changes cannot cure it; more tries can, and:
- `router.strict_drc=true` **is now on for every card** (build_board.sh). Its first try closed the video card completely (0 unconnected, after ten passes at 1 where every plain try had stopped); 21 plain tries since its DE-15 fix had not. The costs are reachable too, as `router.scoring.via_costs` (50) and `router.scoring.start_ripup_costs` (100) in 2.5.0 (`router.via_costs` is ignored with a warning).
- **Kept from the attempts:** `P32_LAYERS` and `P32_PCB_OUT` (a trial on another layer count, into another folder, without touching the card's board), and `root_sheet()` in kicad_tools.py: KiCad empties the project's list of top-level sheets whenever a board is saved without its schematic open, and the finish step now names the root sheet again.

D83 — Freerouting with its two fixes built in; every card routed (2026-10-31; Ken: "do some web research on best layout options and tricks in using the router", "Build with fixes")

- **The router, not the boards, was the limit** (D82): Freerouting's open issues #1017 (the last incomplete connections traded between neighbouring nets, pass after pass, until the stall rule stops it) and #1054 (from pass 3 on, a rejected route leaves what it ripped up unrouted). Both have fixes in open pull requests, #1021 and #1055.
- **P32W routes with Freerouting built from its master with both merged** (`generators/build_freerouting.sh` builds it into `~/freerouting/p32w-fixes/freerouting.jar`; `build_board.sh` takes it when present, else the 2.5.0 release). It reports itself as v2.5.1-SNAPSHOT.
- **What it did:** the CPU card, which no 2.5.0 try had closed (its best 18), routed completely on its first try at cost 1.5: 467, 157, 49, 18, 7, 2, 1, 0 open after its passes, no cycle, 69 minutes; 0 DRC violations, matching its schematic. The memory card routed in 94 seconds (16, 3, 1, 0) against about four minutes with 2.5.0.
- **Every card is now routed:** the backplane (D79), the memory, MMU / data and I/O cards (D80), the disk, debug and video cards (D82) and the CPU card, all on 6 layers, 0 unconnected, and each matches its schematic (KiCad's parity check: 0 footprint errors). The backplane's generator now gives unused pins the nets KiCad's schematic gives them, and keeps the bolts in the BOM (the mounting-hole footprint left itself out): its 199 parity warnings were those. The only DRC violations left are silk on the I/O and video cards.
- **To do:** go back to a Freerouting release once one has both fixes (#1021 is aimed at 2.6).

D84 — The kernel's sources in /usr/src (2026-11-05; Ken: "go ahead and add the kernel sources to /usr/src")

- `/usr/src/kernel` on the disk: `kernel/*.c`, `*.h`, `*.s` and its README, copied by `software/Makefile`'s `src.stamp` rule like the other trees.
- **To read, not to build:** the kernel's Makefile runs the host's Python compiler, assembler and linker, so it is not copied. Building the kernel on P32W itself would need its own makefile, and a check that the native `cc`, `as` and `ld` can take `start.s`, the shared `mem.s` and the flat image at 0x00100000; not attempted.

D85 — A clock counter: `p32uemu -G` and `cycles` (2026-11-05; Ken: "a clock counter, which may need to be 64 bits ... start the counter, do a series of steps, say make the cmd src tree, and stop the counter", "go ahead in that order")

- **What it is for:** the real machine's time for a job, worked out in the emulator. It is **for the emulator only** (Ken: "really only for the emulator not the real hardware"): nothing is added to the hardware, and on the real machine `time` is the tool.
- **In `p32uemu`, not the gate-level simulator:** the gate level could never get through a `make`, and its cycle counts already match `p32uemu`'s in the test suite. `-G` implies `-C`, so wait states and the 1 MHz timer are exact. 64 bits: `ucycles` already was; 32 would wrap every 23.9 minutes at 3 MHz.
- **The gate is how P32W tells the emulator when to count:** bit 7 of the output port, `/dev/port`, so no kernel change, and with no `-G` nothing changes. `cycles command`, or `cycles start` and `cycles stop`, moves it. The port is the only cost: run on the real machine, `cycles` would set it and clear it.

- **The report:** cycles and their time at the clock (3 MHz, or 4 with `-F`), instructions, MIPS leaving out idle cycles, idle cycles, disk sectors and commands, console bytes.
- **Its limit:** devices take no time in the emulators, so the count is the processor's work. A drive's access time and the console's baud rate are added by hand from the disk commands and console bytes.
- **Checked:** `gate_test` (exact instruction counts, an idle wait, the other port bits ignored, the run's total unchanged by measuring) and `software/tests/cycles_test.sh`, where two independent clocks must agree on a running system: `ls -l /bin` counts 10.702 s and the kernel's 50 Hz ticks say 10.70 s.
- **First measurements (2026-11-05, 3 MHz, 16 MiB of RAM, processor time only):**

Job	Cycles	Time	Disk commands
`ls -l /bin`	32,107,476	10.7 s	18
compile and run `hello.c`	39,312,480	13.1 s	662
the kernel, `make`	3,695,936,116	20 min 32 s	7,592
every command, `make clean; make`	26,214,809,902	2 h 25 min 38 s	76,518

  The command tree built on P32W: all 101 programs byte for byte the host's.
- **Open question:** over long runs the kernel's clock falls behind the count: 0.42 s in 1,232 s, 4.13 s in 8,738 s (about 0.05%). A tick lost when interrupts stay off for more than 20 ms would do it, but that is a guess: not looked into. Trust the count.

D86 — The kernel builds on P32W itself (2026-11-05; Ken: "I will want to be able to build kernel and for that matter all software on the actual p32w machine", "go ahead with step 1")

- `kernel/Makefile.p32w`**, on the disk as** `/usr/src/kernel/Makefile`**:** the same objects in the same order as the host's `kernel/Makefile`, with the native `cc`, `as` and `ld`. `mem.s` comes from `/usr/src/libc`.
- **The proof:** `unix.bin` is byte for byte the host's, and `unix.map` too but for the objects' paths (`software/tests/kernel_test.sh`, which also checks that a touched source rebuilds that file and the link, and nothing else).
- **It takes 20 min 32 s at 3 MHz**, 3.7 billion cycles (D85).
- **Still to come for "all software":** putting a built kernel on the boot area from P32W (no `dd` on the system; only `tools/p32fs.py --kernel` does it), Lua's sources and Makefile, the boot monitor (`as` makes no flat ROM images), and one `make` over all of `/usr/src`.

D87 — Two speed fixes for `ls -l` (2026-11-05; Ken: "yes do both after the kernel Makefile")

`ls -l /bin` took 10.7 s at 3 MHz, 79 ms a file, and the count (D85) and the kernel profile showed why.

- `ls` **remembers the last owner and group.** It looked each up for every line, and each lookup reads `/etc/passwd` or `/etc/group` through.
- `dirlookup` **reads a directory a block at a time** and scans it in the buffer. It read one 32-byte entry at a time, through the buffer cache, so a name near the end of `/bin` cost about 100 trips through the cache, on every `open` and `exec`. No change to the disk.

`ls -l /bin` at 3 MHz	Time	To `/dev/null`
before	10.68 s	9.17 s
with `ls`'s change	7.36 s	5.82 s
with `dirlookup`'s too	5.04 s	3.51 s

The benchmark (speed.md §4) moves far less, 2:25:38 to 2:23:22: the kernel falls by 16%, but it was 10% of a build, and `ls` does not run in one.

They showed up a bug in the shell. `sh` waited for a pipeline's commands in order, so if the last ended first, its status was thrown away while waiting for the others, and the status read was the wrong process's. `dirlookup`'s new timing made `cmd/mail.sh` hit it (`sent: status 141`: an earlier command's SIGPIPE, not `mail`'s 0). It now takes them in whatever order they end, and `systest/sh/control.sh` forces both orders with `sleep`, independent of timing.

D88 — A profiler for user programs: `p32uemu -q` and `tools/uprof.py` (2026-11-05; Ken: "yes add the profiler after the two fixes")

- `p32uemu -G -q FILE` keeps the cycles at each user address while the measuring gate is up: cycles rather than instructions, so a load weighs more than an add, in the same units as the benchmark. The kernel's cycles are one total. Addresses are a process's, so one program at a time.
- `tools/uprof.py PROFILE MAP` totals them by function. A link map leaves out static functions, and a compiler is mostly those, so it reads each object the map names, whose symbols keep them. `kprof.py` now does the same, which unlumps the kernel's profile (`bget` had been counted as `binit`).
- **Its first find was a warning:** `cc -S` is not `cc -c`. Profiling `-S` blamed `note_line` for 55%, but only `-S` runs that. Profile what the job really runs.

D89 — Three speed fixes in the C library and `as` (2026-11-05; Ken: "go ahead with those three, score each on the benchmark")

Each scored on the benchmark (speed.md §4), which falls from 2:23:22 to 2:00:49: with D87, 17.0% below the baseline.

- `ctype` **macros** (2:10:27): `isdigit`, `isupper`, `islower` and `isalpha` are arithmetic, equal to the functions for any `int`; `isalnum` and `isspace` read a 256-byte table, so they take the argument as an `unsigned char`. Each uses its argument once, and the functions are still there for a pointer to one.
- `putc` **straight into the buffer** (2:06:02): `FILE` gains `room`, the bytes `putc` may store without a call, above 0 only on a fully buffered stream being written; every change of state goes through `fflush`, which zeroes it, so terminals and unbuffered streams keep the slow path. `fputs` and `printf` use it. A macro alone would have done nothing for `cc`: it writes with those.
- `as`**: the first character before** `strcmp` (2:00:49): not the hash table first proposed. Symbols were already hashed, and the calls were spread over 53 sites, chains a line walks through, so a small `eq` macro answers most of them in one compare; `is_reg_name` and `opcode` likewise. `as` on `sh.s` falls from 142 s to 98 s, its output unchanged.
- **Tests:** `systest/lib/fastio.c`, on P32W: the macros against the functions and the rules for -1 to 255, and `putc` through full buffers, `fwrite`, `fseek` and a read-write stream.

D90 — The boards put right before any order (2026-11-05; Ken: "go ahead and fix the P32W boards, all issues, none have been ordered and are a work

in progress")

The first run of `pcb-check`, the new pre-order check, found these on the eight boards. None had been ordered. The routing stays: `p32kicad_pcb.py --rework CARD` edits a routed card in place, and the backplane is generated, so it just builds again.

- **The LEDs were backwards on the silk.** In KiCad's LED footprints pad 1, by the flat on the outline, is the cathode, and `p32net.py` had the anode on pin 1, so a builder going by the flat would fit all 16 the wrong way round. Pin 1 is now K. On a routed card each LED turns 180° about its centre, so its two pads stay where they were, on the same nets.
- **So pad 1 is round on the LEDs, and a `K` marks it.** The first rework left the square pad where a round one had been, and on the memory card its corner clipped a track that had hugged the round one. Comparing pad positions and nets missed that; the copper proof and the DRC caught it.
- **Thermal reliefs on the planes** (0.25 mm gap, 0.5 mm spokes: `thermals`). Every plane pin was joined solid, and a 6-layer board's planes draw the heat out of the iron, so these joints would be hard to solder and harder to undo. The backplane's bolts and mounting holes stay solid.
- **Every reference and value on the silk, clear of every part** (Ken's rules of 2026-10-31 and 2026-11-02): `short_value` and `label`. An LED's reference is above it and its function below. The backplane's networks stand 5 mm apart, not 4, to make room, so it is 381 mm wide, not 376.
- **The gerber zip holds what the fab needs:** copper, masks, silkscreens and the outline, no paste or other layers (`kicad_tools.py finish`).
- `pcb-check.ini` **beside each board:** its name and revision, and on the CPU, debug and I/O cards why their thin +3V3 tracks are fine (`exempt_power`).
- **Checked:** every board has 0 failures and 0 warnings in `pcb-check`, including the copper proof, gerbers against schematic, for 3,549 nets. Before and after the rework every track and via on the seven routed cards is identical, and every pad has the same net at the same place.

D91 — Four more speed changes, three kept (2026-11-05; Ken: "go ahead with those four")

Each scored on the benchmark (speed.md §4), which falls from 2:00:49 to 1:39:18: 31.8% below the baseline. `cc -c sh.c` takes 233 s, from 277 s with D89.

- `cc`**'s lexer** (1:47:24): a punctuator is found by trying each in the table in turn with `strncmp`; now the first character is compared first.
- `cc`**'s** `is` (1:42:51): the same, for the parser's tests of a token's text.
- `strlen`**,** `strcmp`**,** `strncmp` **and** `strchr` **in assembly** (1:39:18): `libc/str.s`, as `memcpy` and `memset` are in `mem.s`. The compiler's C took about three times the instructions.
- **Dropped: one division a digit in** `printf`**.** 1:42:54 against 1:42:51: no gain, so it is not in.
- **Tests:** `systest/lib/strasm.c`, on P32W: the assembly against plain C versions, over empty strings, bytes above 127, n of 0, a difference past n, and a search for NUL.

D92 — The kernel in the profile, and what a program costs before its own code (2026-11-05; Ken: "yes add the startup profile")

- `p32uemu -q` **keeps the kernel's addresses too**, and `tools/uprof.py PROFILE unix.map -k` totals them by function, in cycles, for a whole job: the kernel has one set of addresses, so every process in the job adds up. (`p32emu -k` and `kprof.py` count instructions, from boot.) `cycles_test.sh` checks that the functions add up to the kernel's total.
- **What it found** is in speed.md §4: with seven common headers a program that does nothing takes 14.8 s to build, which over 101 programs is a quarter of the benchmark, and for the smallest command the kernel is 72% of the time.
- `time` **understates the kernel.** It counts 50 Hz ticks, and a tick cannot land in a short trip with interrupts off, such as a TLB refill. It says 12.7% of the benchmark; the cycles say 21.4%.

D93 — What every program pays: seven fixes from the startup profile (2026-11-05; Ken: "go ahead with items 1–6 and the process-slot fix")

Each scored on the benchmark (speed.md §4), which falls from 1:39:18 to 1:18:39: 46% below the baseline, with all 101 programs still byte for byte the host's. `cc -c sh.c` takes 183 s, and disk commands fall by 39%.

- `cc`**'s integer constants** (1:34:15): every digit of every constant ran a 64-bit division in software to check for overflow, which was 11% of `cc`. The digits that cannot overflow now go in 32 bits.
- **The process table** (1:31:06): the clock tick and the scheduler walked all 64 slots, nearly all empty, 50 times a second and at every switch: 4.3% of the benchmark. `proctop` marks the end of the slots ever used, and `wakeup`, `exit`, `wait` and the load average stop there too. `allocproc` still takes the lowest free slot, so the order of running does not change.
- `ialloc` (1:29:56): every new file scanned the inode table from the start, with a `bread` for each inode. Each mount now keeps `inext`, below which every inode is in use, and the search reads a block at a time. It still gives the lowest free inode. 12,100 fewer disk commands, against estimates of 34,000 and then 20,000.
- `ld` (1:28:02): it read all 282 KB of `libc.a` and decoded every member, when a small program takes 66 KB. The archive's index already says what each member defines, so it reads that, and then only the members it takes, in the same order. 17,000 fewer disk commands. `qsort` now swaps a word at a time.
- `as` (1:23:24): pass 2 allocated a 256-byte buffer for each line's 4 bytes. One buffer now serves every line, and a line's operands come from blocks. `malloc` and `free` had been 15% of `as`.
- **Runs of text** (1:19:57): `printf` sent its format through `put` a character at a time and set eight locals for each. It now finds the next `%` with `strchr` and copies the run (`putrun`), and `%s` and padding likewise. `fputs` goes straight into the buffer when it fits, and `gen.c`'s `emit` uses the length `vsprintf` returns. The C library grows by 632 bytes.
- `make` **and page tables** (1:18:39): a command with nothing in it for the shell runs by itself, one process fewer. Anything else, or a command exec cannot find such as `cd`, goes through `sh` as before, so failures read the same. The last four page tables freed are kept, zeroed, because `pt_new` walked the whole free list. They count as free memory, and a shortage takes them back.
- **A bug this uncovered:** a buffer that had failed a transfer kept `B_ERROR`, and `bnoread`, with no transfer to clear it, handed it out for a new block, so the next write failed with EIO and leaked the block. `ialloc`'s scan had churned the cache enough to hide it. The inode numbers were the same with and without the new search: what changed was which buffers stayed in the cache. `session14` now tests it directly, failing on the old kernel with `cp: write error`.

- **Tests:** `systest/cmd/make.sh` (plain commands, the shell's, builtins, failures), `systest/lib/fastio.c` (runs of text at the end of the room, and into a buffer and past it), `systest/kern/kernel.c` (removed inodes used again, lowest first; free memory unchanged by a run of forks). `toolchain_test.sh` proves `as`, `ld` and `ar` byte for byte the host's.

D94 — A third round: what the tools do for every token and every line (2026-11-06; Ken: "go ahead with the next speed round")

Each scored on the benchmark (speed.md §4), which falls from 1:18:39 to 0:57:46: 60% below the baseline, with all 101 programs still byte for byte the host's. `cc -c sh.c` takes 129 s, from 346 s at the baseline. The round was expected to save about 10 minutes and saved 21. Most of what went was a function called many times for each thing, where one test would do.

- **One call for a binary expression, and token tests inline** (1:12:13): `binary` went down ten levels of calls for every operand, one a precedence level, and each level tested the next token against its operators. It now takes an operand and loops while the next operator binds tightly enough (`bin_level`, a `switch` on the first character), giving the same trees in the same order. `is` and `at` are macros that compare the first character inline, since nearly every test fails there. `cc` grows by 10 KB.
- **Punctuators** (1:08:38): the lexer looks only among those with the token's first character (`punct_first`, `punct_next`, longest first), and they are interned at the start, so a punctuator is neither copied nor hashed. `VPUSH` is `vpush` inline on the hot paths.
- `printf` (1:06:11): plain `%s` and `%d` are tested first, before the walk through flags, width, precision and length.
- `as` (1:00:42): `opcode` searched the table three times a line; now once, and the line keeps the answer. `reg` takes `r0-r15` and `sp` as the compiler writes them, with no copying. A line with no `;` skips `strip_comment`'s walk, and an operation's case changes in line. `toupper` and `tolower` no longer call another function.
- **Plain numbers** (0:57:46): `eval` takes a decimal of up to 9 digits without the expression evaluator, which was six levels of calls here too.
- **Tests:** `cc/tests/ops.c` (every operator against its neighbours and every punctuator, with and without spaces, against the host compiler), `toolchain_test.sh`'s `forms.s` (spellings `as` now reads quickly and those it still reads the long way, byte for byte against the host's assembler), and `%d` in `systest/lib/fastio.c`.

D95 — Separate drill files for plated and non-plated holes (2026-11-06; Ken: "my fab is jlpcb")

The order zips held one drill file, with the non-plated tools marked only in comments, so a fab that does not read those could plate them. JLCPCB's KiCad guide says to keep the two apart, and YACC1-D's and elf's orders already do. `kicad_tools.py` now exports `NAME-PTH.drl` and `NAME-NPTH.drl`, and `orderfiles` remakes a zip alone. On a card the non-plated holes are the six mounting holes of the three DIN connectors. All eight boards pass `pcb-check` with the new zips, the copper proof included. By JLCPCB's published limits the boards fit (the backplane, 381 x 366.7 mm, is under the general 576 x 469 mm), but the 0.3 / 0.6 mm vias' 0.15 mm ring is their stated minimum for multilayer boards, under the 0.20 mm they recommend.

D96 — P32W in a GitHub codespace, from a web browser (2026-11-07; Ken: "go ahead with Codespaces, i will be using a chromebook and would like to access via webbrowser")

- **What:** `.devcontainer/` makes a codespace of the repository: VS Code in a browser tab on a Linux machine of GitHub's, so P32W's software can be worked on from a Chromebook. Made once at creation (`setup.sh`, about ten minutes): the host's tools, both emulators, P32W Unix's disk, and the test suite's own copy (`test.img`). `.devcontainer/README.md`, opened at the start, says how to use it.
- `tools/p32w` boots P32W from `software/build/disk.img`, on any machine: `-u` for the microcode-level emulator, `P32W_DISK` for another image. In a codespace that disk is the codespace's own; on a Mac it is the one P32W is used from, so try things on a copy.
- **Proved on Linux** by `.github/workflows/codespace.yml`: it builds the same container and boots P32W in it, and with `full` runs the whole test suite. It runs only when started (the Actions tab, or `gh workflow run codespace.yml`); `run` runs a command of one's own in the container instead, to look into a failure there.
- **Portability:** the emulators are plain C99 and POSIX; they define `_DEFAULT_SOURCE`, as glibc shows the POSIX calls under `-std=c99` only when asked (a Mac shows them anyway). The host's builds of the test programs link with `-lm`: Linux keeps the maths library apart, a Mac has it in libSystem. `archive_test.sh` read times and inodes with the Mac's `stat -f`; it now asks Python, and asks GNU tar for pax where it means pax. Tests that need a Mac's own tools print SKIP lines elsewhere, counted at the end: the two `ditto` checks of `archive_test.sh`, and `fat_test.sh` (`hdiutil`, `fsck_msdos`).
- **What Linux found besides** (all fixed, 2026-11-07): the TLS helper waited forever under TLS 1.3, which a Mac's Python (LibreSSL, TLS 1.2) never speaks; ping took no reply as its own, as Linux's echo socket puts its own identifier in, and has no FIONREAD; the network services test typed ahead of a prompt it had already seen. Where C leaves a choice to the C library, P32W's makes the Mac's: on another host the p32cc tests and `/usr/tests` compare `floatlib` and `lib` with the Mac's output (`cc/tests/NAME.host`, which a Mac checks is still its own), and the host's compiler is clang, as a Mac's `cc` (gcc evaluates arguments right to left). The whole suite passes in the codespace's container: 151 tests, and the 3 SKIP lines.
- **Not here:** KiCad. The boards stay on the Macs.

D97 — The debugger, in a browser tab (2026-11-07; Ken: "I want to develop a web based emulator that can also display graphics and maybe cpu internals"; then: on the host, as YACC1-D's y1dbg, with the screen, the console and debugger, and the CPU's insides)

- **What:** `tools/p32dbg` runs P32W in `p32emu`, or with `-u` in `p32uemu`, and shows it on a page in the browser: the video card's screen as the card draws it, with the PS/2 keyboard (the screen is `/dev/tty4`, a login of its own: D53); UART 0's console; the registers, the code with the monitor's and the kernel's symbols, memory and breakpoints; and from `p32uemu` the last microcycle: the microaddress, every field of the microword by name, the A, B and R buses, the ALU, MAR, MDR, IR and the 32 physical registers. The same command language runs from `-c` and from scripts.
- **How:** the machine stays in C. `-dbg PORT` starts a debug server in either emulator (`emulator/p32dbg.h`): one connection on 127.0.0.1, a line a command, a line of JSON each. Without `-dbg` its hooks are an untaken branch an instruction (a microcycle in `p32uemu`). While P32W idles in WAIT the connection wakes it, so an idle debugger costs nothing. The front end is Python, the standard library only; its disassembler is made from

`microcode/p32isa.py` and the microword's names from `microcode/p32ucode.py`, so neither can drift from the machine.

- **Where:** on a Mac, http://127.0.0.1:8832/; in a codespace GitHub forwards the port and opens the page, at an address only its owner reaches. The disk is `software/build/disk.img` unless `-d` names another, as for `tools/p32w`.
- **Choices on the way:** a reset under the debugger is the hardware's (`-RES`): the register file keeps its contents, so both emulators agree. A step counts instructions begun (the RESET sequence after a reset is none). The server never reads a device register, as a read can take its data: memory there shows `??`. The screen's keys go as set-2 scan codes, make and break, so shift and ctrl are the keyboard's own.
- **Tests:** `make test` section 22: a session of commands against the monitor in both emulators, recorded (the console's input queued before the run, so every stop is at the same instruction), and the browser side (`tools/p32dbg/tests/web_test.py`).

D98 — The video card again: a 6845, as YACC1-D's (2026-11-10; Ken: "I think the p32w video card is too complicated. Lets go with a card similar to Yacc1 card so we can use a 6845"; then "VGA, 31.5 kHz", "ATmega328P, as YACC1 V2.0", "Text and a graphics layer")

- **Why:** revision 0 (D67) came to 88 ICs, 18 of them GALs: one RAM shared four ways in each character time, a handshake across two clocks, the raster counted out in 74F163s and GALs, PS/2 in logic. It was laid out (D81, D82) but never ordered or built.
- **What:** revision 1, `docs/video-card-6845.md`, a description for Ken to check:
  - **a 6845** makes the raster, the cursor and the RAM's address. VGA's 3.147 MHz character clock is above Motorola's MC6845 rating (3.0 MHz): Hitachi's S type, rated 3.7 MHz by secondary sources (to confirm in Hitachi's datasheet), the HD68B45SP of YACC1-D's V2.0 card;
  - **the text in dual-port RAM**, as YACC1-D's card: the CPU and the 6845 each have a port, so text never waits and nothing schedules slots. Four IDT7134s, so words work (two would do for halfwords: open with Ken);
  - **a graphics layer**, as YACC1-D V2.0's: one bit a dot, 640 × 400 over the text, a plane per scan line as the 6845 addresses it, two pages, one fast RAM shared with the display in the two halves of each character time. The bus waits a fixed two clocks, so no handshake. The card can be built text-only;
  - **PS/2 on an ATmega328P** with YACC1-D's firmware (`embedded/video-ps2`): the keyboard as ASCII and VT100 sequences, the mouse's packets, one queue, one vector;
  - **kept:** VGA 640 × 480 at 60 Hz on the DE-15 and its levels, the font ROM and font, 80 × 25 cells (the character, then the attribute: inverse, underline, bright, blink), the RAM window and register page (D52), the interrupt level and vector (D57), D72's register writes at the cycle's end.
- **About 53 ICs, 6 GALs; about 42 built text-only** (an estimate until the netlist).
- **Next**, each for Ken in turn: the emulators, the debugger's screen and the kernel's driver on the new card, with tests; the netlist and its gate-level check; the layout, pcb-check and a design review; the order. Until then the emulators and the kernel model revision 0, and its files (`docs/video-card.md`, `generators/card_video.py`, `hardware/video-card/`) stay as they are.
- Supersedes D67, its PS/2 in logic included.

D99 — The backplane: ten slots at 1.5 inches, a front and a back (2026-11-11; Ken: "Drop to 10 slots. Keep 0,1,2 and then only 3-9. Lets move all

compenents to one side ... both power entries to that side as well. So the right side is going to be the 'back' ... All cards will be put in with compenents faceing the front or left side. The card numbering sould satart on the left with 0. Lets increasse the space to 1.5\" between cards. Lets also add addirtion mointing support to the top and bottom (more holes)"; "when generarting PNGs generate bottom as well as current top and side")

The backplane was laid out (D79) and had passed pcb-check, but was not ordered.

- **Ten positions, slots 0–9**, slot 0 at the left: 0 the debug card, 1 the CPU, 2 the MMU / data card, 3–9 any card. The seven general slots are what a full machine fills: four memory cards, I/O, disk and video.
- **1.5 inches (38.1 mm) between slots**, 1 inch before (D49, D64). By pcb-check's figures a card now has 34.0 mm to its neighbour (21.3 before), so the I/O card's WIZ850io, about 24 mm on its socket, fits; the cards' `pcb-check.ini` say `slot_pitch = 38.1`.
- **A front and a back.** The left end, slot 0's, is the front; the right end, past slot 9, is the back. Everything that is not a socket or its bypass is at the back: the pull-up networks and resistors, the clock's termination, the TL7705A (it was at the left end), and **both pairs of power bolts** with their 1000 µF, in the gaps between the socket rows where no channel runs. **The +5V LED and its resistor are the exception, at the front edge** (Ken, the same day: "Move the power LED and resistor to front edge where it can be seen"). 34 mm of board stay in front of slot 0, the room its card's components have.
- **Cards go in with their components facing the front.** The sockets did not have to turn: column c was already on slot 0's side of column a, and on a card's right-angle connector row a is the row nearest the board and row c the farthest above it. That is read from KiCad's footprints of the two connectors, so it is still to verify with a real connector and socket before ordering. The silkscreen says FRONT, BACK and which way the components face.
- **18 mounting holes** (8 before), M3, plated, on GND: the four corners; along the top edge and along the bottom edge one in every second gap between slots (between 1–2, 3–4, 5–6 and 7–8) and one over the parts at the back; four mid-board between the socket rows, in the same four gaps. First drawn with one in every gap (28); Ken thinned them the same day: "I think you can remove mounting slots beween 1-2, 3-4,5-6,7-8,9-10", read as counting the ten sockets from the left, so the gaps between slots 0–1, 2–3, 4–5, 6–7 and 8–9 lost theirs.
- **The board's name, revision and cottageworker.com run along the front edge, 7 mm tall** (1.5 mm in the middle of the board before), with FRONT and the line about the cards' components, all reading from the front (Ken: "font for cottageworker.com can be increased quite a bit as well as moved to the front of the board (parallel) to front edge along with board name and Rev").
- **"a1" is printed beside pin a1 of every socket** (Ken: "yes print a1 beside each slot"). Ken asked whether a socket can simply be turned round to match the card's connector, and is right that it can: a socket is straight-through contacts, the same electrically either way, and "the actual bus connector labels are just labels". What the wiring fixes is how a card must stand (components to the front, pin 1 down), and the socket's keyed shell must let it in that way; a socket the other way round would only take a card turned about, with the supply reversed. In the design the a1s line up (the memory card's J1, J2, J3 a1 at 89.37, 222.72, 356.07 mm from its top edge, the board's a1 holes the same), and by the parts' markings the socket's a1 goes at the printed a1. To try with real parts: Ken will. (A turned socket's fixing holes would be 0.6 mm off the board's.)
- **The slots are labelled, large** (Ken: "Can you label the slots? Debug, CPU, MMU/DATAS, general purpose?"): SLOT 0 DEBUG, SLOT 1 CPU, SLOT 2 MMU/DATA, SLOT 3–9 GENERAL PURPOSE, 3 mm and 2.5 mm tall, between each slot's J1 and J2 sockets (they were 1.2 mm, and slots 3–9 had only their number).

- **14 mm taller than the cards at the top and at the bottom** (Ken, asking whether the cards overhang the bus board: they were the same, 366.7 mm; then "add 14mm at top and bottom and move the edge holes"): 394.7 mm, as a standard 9U backplane is about 28 mm taller than its cards, so the frame's rails bolt to the board outside the cards' ends. The corner and edge holes are in the middle of those two strips.
- **The board is 477 × 394.7 mm** (381 × 366.7 before), 6 layers, routed by its generator as before: 0 DRC violations, 0 unconnected, the schematic's netlist equal to the canonical one.
- **J3's eight spare pins are wired between slots 0, 1 and 2** (B28, B30, C25–C30; they were not connected), as the local link is: Ken asked whether more dedicated lines between the debug, CPU and MMU / data cards might be designed in after the backplane is ordered, and then: "yes wire the 8 spare pins". J2's 19 spare pins were bused to every slot already (D68). No card changes: the cards leave these pins unused.
- **The bus definition is otherwise unchanged**: `p32bus.py` still has a slot 10 and the arbiter a request pair for it, which on this backplane joins only slots 0 and 2. The MMU / data card pulls an empty slot's `-BREQ` up, so it is never granted. Taking slot 10 out of the bus would move pins on the MMU / data and debug cards and mean routing both again: Ken's to ask for.
- **Unchanged and still for the bench:** the pull-ups' values (a line is one socket shorter and about 15 cm of track longer, much the same load) and the clock's termination.
- **Renders:** `kicad_tools.py finish` now makes the bottom view too, for every board: `<card>-render-top.png`, `-render-bottom.png`, `-render-3d.png`. A board too large for an A2 placement sheet takes A1 or A0 (the backplane: A1).
- `build_schematics.sh` joins the PDFs with Python's pypdf where poppler's pdfunite is not installed.
- **Open with Ken, from D49's reasons for 1 inch:** 38.1 mm is 7.5 HP, not a whole number of Eurocard units (5.08 mm), so in a standard subrack, whose card guides and front panels sit on that grid, every second slot is half a unit off; 1.6 inches (8 HP, 40.64 mm) is the nearest pitch on the grid. And at 477 mm the board is wider than a 19-inch subrack's 84 HP (426.7 mm), as ten slots at this pitch with a field beside them must be. Neither matters in a frame built for it. **Settled the same day** (Ken: "ok we can ignore frame pitch"): the pitch stays 1.5 inches.

D100 — The debug card: a two-digit status display, the mode switch turned to the edge (2026-11-11; Ken: "Next up is the debug board, lets get it ready for ordering"; "How about adding 2 TIL-311 Hexadecimal displays near the LEDS. Those could be used for indicating status info. Also turn the listen/drive switch 90 degress so toggle is at the edge of the board (overhangs slighly)"; "the TIL-311 can go just below LEDS. Since this is the front board in a system they will always be visible"; "til-311 is a display ... for better rendering")

- **Two TIL311 hexadecimal displays** (U139 the high digit, U140 the low one) at the front edge, just below the LED bank, on 0.45 in centres as TI's datasheet mounts them. The Pico's firmware writes a byte to local-bus address `4D` and they show it: each TIL311 has its own latch, so its strobe is the decoder's `4D` output and no register is added. Blanking tied low, the decimal points unused; they show their latches' power-on contents until first written. What the byte means is the firmware's to define.
- **Upright with the card's front edge up.** The backplane lies with its front edge toward the user (D99), so the debug card, in slot 0 with its components facing the front, is seen with its connector edge down and its front edge up. The digits are turned for that, the high digit on the left. To check with Ken.

- **The part:** TI SODS001D, read from a copy of the datasheet: pins 1 LED supply, 2 B, 3 A, 4 left decimal point, 5 latch strobe (low: follow; high: hold), 7 ground, 8 blanking (high: blank), 10 right decimal point, 12 D, 13 C, 14 logic supply; pins 6, 9, 11 omitted. The same as on Ken's YACC1 IO card and System-5 IO panel. Logic 60 / 90 mA and LEDs 45 / 90 mA each: the debug card goes from 2.95 to 3.16 A typical. Setup 50 ns, hold 40 ns, strobe 40 ns.
- **Its footprint and picture** (`generators/gen_footprints.py`): a DIP-14 pin field with all 14 pads, so a 14-pin socket fits (pads 6, 9, 11 unconnected), the body 18.8 × 10.16 mm; and 3D models that show the 4 × 7 dot matrix with a digit lit (`hardware/lib/P32W.3dshapes/TIL311_0.wrl` … `_F.wrl`), so a render shows a display, which way up it is, and which digit is which: the two on the card show 32.
- **The mode switch S1 is turned a quarter turn**, its lever pointing out past the front edge: pin 1 9.5 mm in, so the last 4 mm of the lever stand beyond the board. LISTEN and DRIVE are printed at the two ends of its travel, S1 and MODE beside it. **Open with Ken:** the footprint (here and for the CPU card's S4) is E-Switch's EG1224, a right-angle slide switch, while the parts list says "SPDT toggle, right angle": which part is meant; and which end is LISTEN is still to check on the part.
- **The layout engine** (`p32kicad_pcb.py`) learned: a chip at the front edge (not in the rows, its capacitor at its inner end); two front parts closer than the usual 4 mm (`front_gap`); a part's own 3D model (`model3d`); a switch's positions at the ends of its travel (`panel_ends`); and, for a card that asks (`place_clear_front`), rows that would run into the front edge's parts start below them. The front edge is now taller (the displays, the Pico, the switch), so the debug card's full rows start below the switch. No other card's placement changes.
- **Verified:** the netlist's checks (140 ICs, 603 nets, 296 parts); at gate level (`sim_debug.py alone`) a write to `4D` shows on both digits and stays through other writes; the power budget and the timing pass regenerated; the new placement has no colliding parts. **Not done yet: the board.** It is to be placed and routed again (it was routed, D83), with its schematic, gerbers and renders, once the card's open points (debug-card-theory §15, the backlog's "to settle before fabrication") are decided, so that it is routed once.

D101 — The debug card, ready to order: its three faults put right, the lines DRIVE did not reach, a slide switch (2026-11-11; Ken, to four questions: "slide switch, fix all three, add drive lines, drop both")

The card's open points (debug-card-theory §15; the backlog's "to settle before fabrication", items 3 and 9), decided and done before it is routed again.

- **The capture SRAMs' write race** (fault 1). A sample was written in the sample clock's low half and the write ended two gate delays after the next rising edge, the edge that changes the latches: the SRAMs want their data 9 ns before. In the simulation every sample landed one address early, and a DMA read's snapshot held the next clock's idle bus. **Now a sample is written in the high half after its own edge**: `-WE` = NAND(`WEX`, `SCLK`), ending at the clock's fall, with the data steady half a period more. The write enable passes through a 74ACT74 at the clock's fall (WQ: the spare half of STEP's), so it changes only while the clock is low and `-WE` has no short pulses. In DRIVE, `WEX` is SNAP and not WQ: the one high half after a DMA cycle's last edge. The address counter is clocked by `-WE`'s own rise, two gates later: it moves on after every sample written, and never before the write has ended. So a run of DMA cycles leaves its snapshots at consecutive addresses (CA did not move in DRIVE before). GAL DBCAP: `WEX` in place of `PCNT` (the post-trigger counters count on POST's own pin).
- **Power-on** (fault 2). `CRESN`, low until the Pico's firmware runs, now also clears GO, ARM and STEP (each GAL's clear line takes it), GAL DBSEQ's and DBCAP's registers (the 22V10's reset), and the new interrupt register. Nothing can start before the firmware does, whichever way the switch stands.

- `-BREQ0` **in LISTEN** (fault 3): GAL DBSEQ's request output is gated by DRIVE, so it goes the moment the switch leaves DRIVE, not at the next `CLK` (which may never come with `XCLK` stopped).
- **The lines DRIVE did not reach** (Ken: "add drive lines"):
  - `-BGNT3–10` **and** `-FAST` join the SYS group (alone): a third SYS slice, one more 74HCT574. The card alone can grant a device's DMA engine the bus, and set the I/O card's timer for the 4 MHz clock. `-FAST` is sampled now too: 165 lines. The slices from 11 on are one higher than before (the local bus map's addresses with them).
  - `-IRQ0–7`: a seventh 74HCT273, written at local address `4E`, a bit a line, pulling low through two 74HC05s, in DRIVE, beside a running CPU or alone. Its clear is `CRESN` AND `DRIVE` (the 74HCT08's spare gate): empty after a reset and held empty in LISTEN, by wiring.
- **The buffer's wrap is readable**: CA15, the address counter's sixteenth bit, is bit 7 of `42`.
- **The mode switch is a slide switch** (Ken): E-Switch's EG1224, as the footprint always was; the parts list said "SPDT toggle, right angle" (here and for the CPU card's S4; that card's files take the name at its next build). Which end is LISTEN is still to check on the part before ordering (the drawing could not be fetched): the silkscreen takes the usual slide switch.
- **Dropped** (Ken: "drop both"): the card's own register window on the bus (an ID register and a mailbox; window 4 is free), and writing the boot ROM in place with a PROG jumper (the memory card ties the flash chips' write line high). In no netlist; now out of debug-card.md, architecture.md and the programmer's guide.
- **144 ICs** (140 after D100): the SYS slice's drive register, the interrupt register, two 74HC05s. 304 parts, 617 nets. The layout's tables name each chip by what it is, not by its number, so a slice more no longer breaks them.
- **Verified** (`sim_debug.py`, in `make test`): listen as before (768 samples, 355 cycles); dma, and then **a DMA read whose snapshot holds its address,** `-MRD` **and** `-ACK` **low and the data, CA one further**; alone, and with it `CRESN` clearing GO, ARM and STEP, `4E` pulling interrupt lines low in DRIVE and none in LISTEN, and the SYS group driving a grant and `-FAST`; xclk. The timing pass: 0 failing (the card's 12 MHz mode at 41.0 ns of its 41.7); the write's edges argued by hand in debug-card-theory §13. The GALs' fuse files regenerated (DBCAP, DBSEQ, DBCLK).
- **Still open** (debug-card-theory §15): RESET for XCLKEN with the CPU card in; which end of S1 is LISTEN; the Pico's firmware and the host tool, not written.
- **The board, routed (2026-11-11):** placed and routed once, as D100 asked, with the fixed Freerouting (D83), built on this Mac for it (Ken: "yes build the fixed router if needed"; `generators/build_freerouting.sh`, about 4 minutes). The release, 2.5.0, had taken 27 minutes for its first pass and left 333 of 1,952 connections when the 40-minute limit stopped it; the fixed one finished in 47 minutes with `FRTIME=7200` (connections left after each pass: 253, 80, 28, 12, 3, 4, 0). 6 layers, 617 nets, 0 unconnected, 0 differences from the schematic; 13,582 track segments, 137.6 m (about 35 m on each of the four signal layers), all 0.2 mm, and 385 vias. KiCad's DRC: no errors, and three notes, one for each front-edge LED, whose round pads (D90) differ from the library's footprint. A fourth, the upper display's type printed 0.06 mm into its own pin 1 tick, went by moving a TIL311's labels 0.5 mm further out and relabelling the routed board (`p32kicad_pcb.py --label`: silk only, the tracks untouched). The pre-order check (`pcb-check --layers 6 --tests`) passes with no failures: the copper rebuilt from the gerbers is the schematic's 617 nets, with no shorts or opens. Left to do by hand before an order: S1's ends, the displays the right way up, the fab's gerber viewer.

D102 — The debug card cannot drive against a card that is in: `-CPUIN`, `-MMUIN` (2026-11-11; Ken: "if debug is on drive mode is there protection thsat

it wont drive bus if CPU is?"; "yes add the interlock, both cards"; "go ahead as above")

The question. Beside a running CPU the answer was already yes: the card's bus drivers come on only in its own DMA cycle, which starts on the arbiter's grant; the clock, `-RES` and the local link's groups need ALONE; the rest is open drain. But ALONE itself was only a bit the host writes. With the switch at DRIVE, a wrong command or a firmware fault that set it with the CPU card in put a second push-pull driver on `CLK` and `-CLK`, and with a group bit on `-RES`, the bus or the local link. Nothing on the card knew which cards were in.

- **Two lines, on pins that were spare.** `-CPUIN` on J3 B28 and `-MMUIN` on J3 B30: two of J3's eight spare pins, which the backplane joins between slots 0, 1 and 2 (D99), each between two of row B's ground pins. **The CPU card ties B28 to its ground plane, the MMU / data card B30**: no part, so nothing can be left off a card, and no jumper that could be pulled. The debug card pulls both up with 10k (R6, R7; the POWER LED's resistor is now R8). A line is high only with its card out. The backplane's copper does not change, only two nets' names (`L_-CPUIN`, `L_-MMUIN`); J3 C25–C30 are still spare.
- **What GAL DBDRV now refuses** (it makes every enable; two more inputs, one more output):
  - **a CPU card in:** ALONE counts for nothing. No `CLK`, no `-CLK`, no `-RES`, no group. There is no use of ALONE with the CPU card in that this takes away;
  - **an MMU / data card in** (the CPU card out, ALONE set): the grants `-BGNT3–10` are not driven, which is what was asked; and, found in that card's netlist while doing it, **neither are D and the bus cycle's lines, and an ungranted GO cycle drives nothing**. That card's address drivers and strobes are on the bus whenever no grant is out and it drives D in a write, so GBUS or GD alone beside it was the same fight on some seventy lines. Ken was told before it was built, and said to go ahead. `CLK`, `-RES`, `-BREQ3–10`, `-FAST` and the CPU's side of LR and J3 stay: what testing that card alone needs;
  - **beside a running CPU** (ALONE clear): nothing new. A granted cycle is the arbiter's to give.
- **The grants get a slice and an enable of their own**: slice 11 is `-BGNT3–10` (`GNTOEN`), slice 10 `-BREQ10`, `CLK`, `-CLK`, `-FAST`. They were spread over slices 10 and 11 with `-FAST`. The host's bit for them is still GSYS.
- **Both lines are sampled** (slice 21, bits 4 and 5; two inputs that were grounded): 167 lines now. The host can read which cards are in, and say why a command did nothing: the hardware refuses silently.
- **A card's own "in" pin is a ground pin on that card**: `p32bus.PRESENT` names the card that grounds each line; the netlist code wires and checks the pin as ground there (`cardnet()`), `p32bus.py --pins --card NAME` prints it so for `pcb-check`'s bus pin check, and the gate-level system (`sim_sys.System`) makes the line ground for every card when that card is in the system.
- **What it does not do.** A line broken on the backplane, or a pin not soldered, reads as "card out": it guards against a mistake at the keyboard, not against a fault in the wiring. And it does not settle the arbiter granting in the middle of a stretched cycle (backlog, "to settle before fabrication", item 1), which is no fight between drivers and is still Ken's to decide.
- **Verified** (`sim_debug.py`, in `make test`): alone, with every group and ALONE set: no card in, every enable on, and a GO turns the bus cycle's drivers on; `-MMUIN` low, the enables of D, the bus cycle and the grants off, the grants floating, `-FAST` and `CLK` still driven, and the same GO turns nothing on; `-CPUIN` low, every enable off and `CLK` floating; let go, all back. listen (the CPU, MMU / data and memory cards in): both lines low in all 768 samples. dma: the granted cycles as before. The GAL's fuse file regenerated (DBDRV), each of its outputs checked against its spec over all 2,048 input combinations.
- **144 ICs still; 306 parts** (two resistors), **620 nets** (617: the two lines and `GNTOEN`).

- **The boards** (all four pass `pcb-check --layers 6` with no failures, the copper rebuilt from each one's gerbers matching its schematic's nets):
  - **the debug card**, placed and routed again with the fixed Freerouting: 1 h 25 min (connections left after each pass: 267, 87, 34, 20, 5, 1, 0; the first pass slow, the Mac busy with the other boards' renders), 13,846 track segments, 137.3 m, all 0.2 mm, 425 vias, 0 unconnected, 0 differences from its schematic. R6 and R7 stand beside GAL DBDRV in the bottom row;
  - **the backplane**, regenerated: the same 1,651 tracks and vias, no pad moved, two nets renamed (compared with the board as it was);
  - **the CPU and MMU / data cards**: their schematics regenerated, and on their routed boards the one pad joined to the ground plane (`p32kicad_pcb.py --rework`, which learned two things: an unconnected pad the netlist now ties to a plane goes onto that plane's net and the planes are filled again; and a part's value follows the netlist, so the CPU card's CLOCK switch S4 is a slide switch on the board as in its schematic, D101). No track moved on either.
- **debug-card-theory §15's item on the grants beside the arbiter is settled by this.** Still open there: RESET for XCLKEN with the CPU card in (now firmly a button: ALONE does nothing then), which end of S1 is LISTEN, the firmware, the LEDs' colours.

D103 — The video card on the HD63484 ACRTC, from YACC1-D's V3 design (2026-11-12; Ken: "The video card changes again: base it on YACC1-D's V3 design (the HD63484 ACRTC, VGA) in place of the 6845 revision (D98). I am not building V3 for the YACC1."; "Keep the memory side and the dot path as they are: they are simulated and timed. Design the bus side new for P32W's bus. Write the design note and the list of choices for me before any netlist")

- **What:** revision 2, `docs/video-card-acrtc.md`: a design note and six choices for Ken. No netlist, no model, no driver yet.
- **Kept from YACC1-D's V3** (`~/Developer/YACC1-D/docs/cards/video-v3.md`, `v30_netlist.py`, `v30_netsim.py`), part for part and net for net: the chip's memory side and the dot path, 35 ICs with text method A's eight sockets. They are simulated gate for gate and timed at eight corners; a changed part or a gate moved into a GAL is a new circuit to that check.
- **New, for P32W's bus:** a register page only (the chip at `+00` and `+04`, the PS/2 section at `+08` and `+0C`, the ID, the interrupt status); nothing of the picture is on the bus, and the card's 64 KiB at `0xFE00_0000 + n × 64 KiB` is not used. About 23 ICs, 5 of them GALs: the card about 58.
- **The cycle proposed: four clocks, no handshake.** The chip takes CS with its own clock (2CLK, 6.29 MHz). Hitachi's application note allows a synchronous bus to leave DTACK open if CS lasts three 2CLK cycles (477 ns). CS is low 2.5 clocks for a read and 2 for a write: 500 ns at the least, at 4 MHz.
- **Read again from the manual** (its figures T-2 and T-3): a read's data comes 80 ns after a 2CLK edge, up to 358 ns after CS falls, not 80 ns after CS as the YACC1 note's sections 5 and 13 have it. No result of YACC1's changes at its 1 MHz clock. YACC1-D is not edited from here.
- **For Ken** (the note's §7, each with a recommendation): the chip's host bus, 16 bits or 8; the fixed count or the chip's DTACK; the chip's interrupt wired or polled; the console's text by method B first with method A's sockets, or A; 25 rows or 30; the ROMs as timed or P32W's flash.

- **Carried over:** PS/2 on an ATmega328P with YACC1-D's firmware (D98), VGA on a DE-15, window 5, level 4, card ID 3.
- **Until the software is done** the emulators and the kernel model revision 0, and its files stay as they are, as D98 left them.
- Supersedes D98's 6845 card (`docs/video-card-6845.md`, a description only).

D104 — The ACRTC video card's six choices (2026-11-12; Ken: "agree with all your recommendations")

D103's note put six choices to Ken, each with a recommendation. All six as recommended (`docs/video-card-acrtc.md` §7):

1. **The chip's host bus is 16 bits**: lanes 0 and 1, a command word or a register in one access. DACK is pulled high.
2. **The cycle is a fixed four clocks**, three of them waits, counted in a GAL. DTACK is wired to a spare input of that GAL and a test point, and not used.
3. **The chip's interrupt is wired**: device 1, its own level jumpers, vector n × 16 + 1.
4. **The console's text is method B first** (glyphs copied in the bitmap); the board has method A's eight sockets and tracks.
5. **25 rows**, as D50, on lines 40–439 of the 480.
6. **The two ROMs are 27C256s, as timed.**

Next, by the note's §9: the software in the emulators (the chip's model, the kernel's driver, the debugger's Screen, the tests), then the netlist.

D105 — The ACRTC video card in the emulators and the kernel (2026-11-12; step 2 of D103's plan, after Ken's "agree with all your recommendations")

- **What:** both emulators and P32W Unix now have the video card's revision 2 (D103, D104). Nothing models revision 0 any more.
- **The chip's model**, `emulator/p32acrtc.h`: YACC1-D's `acrtcmodel.h` moved to the chip's 16-bit bus (registers as words, FIFOs of eight words, WPTN in words), with IRQ from the command control register's enables. Its own test, `emulator/test/acrtc_test.c`: 55 checks.
- **The card**, `emulator/p32io.h`: the chip at `+00` / `+04` (halfwords), the PS/2 section at `+08` / `+0C` as the ATmega328P's firmware shows it (ASCII from the keyboard, the mouse's packets, its commands), the ID 3 and revision 2, two interrupt devices (vectors `0x50`, `0x51`). No video memory on the bus. `p32uemu -C`: three waits for an access to the chip. A `WAIT` ends when the chip finishes its work and interrupts. New option `-Km B0,B1,B2`, a mouse packet.
- **The kernel**, `kernel/video.c`: the console by text method B. A character is one graphic copy from a picture of the font; bold, underline and inverse are seven more pictures the chip makes at start; a scroll is one copy and one clear by the chip; the cursor is in XOR. The command words go to the chip from its "write FIFO empty" interrupt, eight at a time; kernel messages wait for it. The keyboard arrives as ASCII, so the scan-code table is gone.
- **The font** for the card's EPROM and the emulators: `generators/gen_font.py --acrtc` and `--acrtc-header`.
- **The debugger** (`p32dbg.h`, `tools/p32dbg`): the Screen draws the chip's picture; `screen` prints the console's text, read from the dots.

- **Tests:** the model; `video_test` rewritten (12 cases, both emulators identical, and with the card's timing at 3 MHz and 4); `session23`, P32W Unix on the video console, passes as it was recorded before the change; the debugger's two.
- **Revision 0** (D67): its emulator model is gone, and with it its two gate-level runs (`vidcard_test`, `ps2_test`); `sim_sys.py --video` refuses. Its netlist, its GALs' checks, its board and its place in the timing pass stay until revision 2's netlist replaces them (step 3).
- **Asked on the way** (Ken: "what would be involved in making bitmap available on the bus as well"): a second port stealing the chip's idle memory cycles; about 12 more ICs, the kept memory side reopened, a two-clock handshake, about twice the speed. Described in `docs/video-card-acrtc.md` §6a with the recommendation to leave it off; for Ken to decide before the netlist.

D106 — The ACRTC video card's netlist; the picture stays off the bus (2026-11-12; Ken: "leave it off, go ahead with the netlist")

- **The second port is not built.** Ken had asked what putting the picture on the bus as well would take (D105; `docs/video-card-acrtc.md` §6a: about 12 more ICs, the kept memory side reopened, a two-clock handshake). His answer: "leave it off". The card has a register page and nothing else.
- **The netlist**, `generators/card_video.py`, step 3 of D103's plan: **58 ICs, 5 of them GALs** (351 nets, 173 parts); 50 ICs with text method A's eight sockets empty. It takes revision 0's place in that file.
- **The kept part: 35 ICs of YACC1-D's video card V3, with the oscillator, the DE-15 and their passives.** The generator transcribes them from YACC1-D's `v30_netlist.py` by that file's pin numbers and net names. A copy of that file is in `hardware/video-card/yacc1-v3/` (from YACC1-D's commit `db48aa5`), and `kept_check()` proves the two equal in `make test`: 50 parts, 765 pins, 178 nets. 11 pins differ, each listed with its reason (DACK, D8–D15, DTACK, the oscillator's pin 1). Nothing of the bus side is on a kept net except at the seam: RS, R/W, CS, D0–D15, RES, IRQ, DACK, DTACK.
- **The bus side, 22 ICs**, as the note's §4 designed it, with these things settled in the writing:
  - **CS is a flip-flop on the bus's** `-CLK`, cleared by a register of the cycle's GAL, not a decode: it cannot make a runt pulse, which the chip, sampling CS with its own clock, could take for an access. A write's CS is two clocks, both edges out of that flip-flop. Its twin opens the data buffers.
  - **The chip's data lines are the card's own data bus too**: the status, the ID, the interrupt status, the vector and the PS/2 latches are on them, so two 74HCT245s are the card's only load on D0–D15.
  - **The PS/2 strobes are registers, never a decode**: a decode that is only settling cannot lose a byte or latch a false command. FULL's clear is a pulse in the clock after a read of `+0C`. **A command written to** `+08` **takes two clocks** (one wait): the latch takes the byte as the second begins, the master holding it a clock either side. `p32uemu -C` adds that wait (the only change to the emulators here).
  - **Two buffers, not three** (A8–A15 go to one GAL only), **no 7416** (two spare open-drain gates of the 74HC05 hold the ATmega in reset), a 74ACT74 for CS: 22 ICs where the note estimated 23.
  - Five GALs: VSEL (the page; the revision's bit), VDEC (which register, `-ACK`, the lanes), VWT (the cycle, the PS/2 strobes), VRD (status, ID, interrupt status, the vector), VINT (who asks, the acknowledge cycle).
- **Checked:** every GAL equation against its specification over every combination of its inputs, and its fuse file through the fuse map; the netlist's electrical rules; the kept part equal to YACC1-D's (six planted faults each caught); **the bus side at gate level, as wired** (`generators/sim_video.py`, new): the four-clock cycle half by half, every register

and its size, 12 bus errors, the PS/2 handshake, both interrupts and their vectors, the jumpers (22 planted wiring faults each caught). All of it in `make test`.

- **Revision 0 leaves the tree** (D67; never built): its generator, its 18 GALs' fuse files (88 GALs now, from 101), its gate-level helper and its runs in the timing pass. They are in the history, last in commit `78473d7`. Its schematic and board in `hardware/video-card/kicad` stay until revision 2's layout replaces them.
- **The timing pass leaves the video card out for now**: docs/timing.md's runs are the machine without it (0 failing, as before). The kept part is timed by YACC1-D's own check; the bus side joins the machine's runs in step 4.
- **The power budget**: the video card is 1.2 A typical, 2.3 A at most (revision 0: 2.4 and 4.2, most of it its 18 GALs); the machine with four memory cards 16 A typical, 29 A at most (17 and 31 before). The figures of seven of the card's new parts are estimates, the chip's among them (Hitachi's manual gives no supply current).
- **A small addition to the GAL builder** (`GB.com(..., inverted=True)`): an active-low output of a wide decode is written as its active-high sum, with the 22V10's polarity fuse. The fuse-file writer, the simulator and the timing pass already knew such outputs.
- **Not done here, by the plan:** step 4 (YACC1-D's timing check run on this netlist; the bus side against a model of the chip's host timing at every phase of its clock; the card in a machine at gate level against `p32uemu -C`; the static timing pass), then the schematic and the layout.
- **To check** (the note's §8): the five fuse files against GALasm, which is not on this Mac; the estimated supply currents; the 27C256 in the programmer.

D107 — The ACRTC video card simulated and timed (2026-11-12; Ken: "go ahead with step 4 when the tests pass")

Step 4 of D103's plan, on the netlist of D106. Four checks, all in `make test`.

- **The memory side and the dots: YACC1-D's own timing check, run on P32W's netlist.** `generators/sim_video_timing.py` loads YACC1-D's `tests/video/v30_netsim.py` (a copy beside the netlist's, in `hardware/video-card/yacc1-v3/`, from YACC1-D's commit `12565a7`) and runs its two scripts unchanged on the kept part: 20 runs at eight corners, all pass. The netlist's copy moved to the path it has in YACC1-D, which is how that file finds it.
- **The bus side with delays, against the chip's host timing** (the manual's AC table, figures T-2 and T-3), on the same engine: a write and a read of the chip at every 4 ns of the chip clock's phase against the bus clock, 3 and 4 MHz, six corner sets, then the card's registers and the PS/2 section: 516 runs, all pass. The least to spare: CS rises 62 ns after DTACK; CS is low 12 ns over three 2CLK cycles; RS and R/W are steady 70 ns over their 50 before CS; a read's word is on the bus 196 ns before the cycle ends, where the machine needs 140. Another master's address passing through the chip's, `+0C`'s or `+08`'s moves nothing.
- **The static timing pass** has the video card back in the machine's runs (its bus side; the chip and the ATmega have clocks of their own): nothing fails, the fastest clock is unchanged (3.09 MHz worst case). New there: **how early the machine needs read data on the backplane's D lines, 140 ns at worst case** (`READ_SETUP`), which a device answering from its own clock is checked against; the pass fails if the machine comes to need more.
- **The card in a machine at gate level** (`sim_sys.py --video`): the CPU, MMU / data, memory, I/O and video cards as wired, `p32io.h`'s models at the chip's and the ATmega328P's pins. `vidcard_test.asm` (new: the chip's cycle, registers and FIFO, the PS/2 latches, both interrupts taken with their vectors, bus errors) gives the same trace and the same 833 cycles as `p32uemu -C`, at 3 MHz and at 4. `video_test` whole, run once by hand (ten minutes a run): identical too.

- **Found on the way, and put right:**
  - `p32uemu -C` **saw the chip's interrupt an instruction too soon.** The CPU card samples the `-IRQ` lines in the middle of a clock, and the chip takes a write at that moment of its cycle's last clock; a request the write causes is seen in the next microcycle. The emulator's `-C` timing now has the chip's IRQ as the microcycle began.
  - **A write's CS is two clocks give or take 11.5 ns**, not "a few nanoseconds" as the note said: the flip-flop's rising delay against its falling one at the data sheet's extremes. 488 ns at the least at 4 MHz, still over the 477 asked.
- **Not modelled, noted for the bench and the backlog:** the chip's delays where the manual gives none; the PS/2 firmware's own time (the emulators reload at once; on the card FULL stays low some microseconds after a read, so `video_test`'s back-to-back reads would need a wait; and after the firmware's flush command the byte in the latch still waits).
- **Next** (the note's §9): the schematic and the layout, `pcb-check`, a design review; then Ken orders.

D108 — The ACRTC video card laid out; its timing checked with the board's tracks, and the kept part placed for it (2026-11-12; Ken: "go ahead with step 5 when the tests pass")

Step 5 of D103's plan: the schematic, the board, `pcb-check`, a design review. The note's §4.9 has the tables.

- **The schematic:** eleven sheets from the netlist of D106 (`hardware/video-card/kicad/video-card.kicad_sch`). KiCad's rule check finds nothing; the netlist it reads back from the drawing is the generator's (351 nets). Revision 0's schematic and board, which had stayed in that folder, are replaced (in the history, last in commit `5d447f7`).
- **The board:** 340 × 366.7 mm, six layers (D77), all through-hole (D21). 730 connections routed in under five minutes, 59.5 m of track; DRC's one warning is the LED footprint's (D90). The layout rules: the power LED, the DTACK test point, the RAM-size jumper, the DE-15, the two mini-DINs and the FTDI header at the front edge, the cable connectors facing out and named (VGA, KEYBOARD, MOUSE); the ten jumpers along the bottom edge, each named; every reference and every type or value on the silkscreen outside its part; `cottageworker.com` beside the name and revision.
- **The review found the first layout wrong for the kept part's timing, and the layout was done again.** YACC1-D's check leaves 0.2 ns at the least (D107), and a track is about 0.07 ns a centimetre. With the first routing's tracks as delays one run in twenty failed (text method A, the left character's latch 0.7 ns late), and the dot flip-flop was down from 7.2 ns to 0.7. The router's tracks were as short as the placement allowed (within a tenth of the pads' shortest trees). The placement was the fault: rows by the netlist's groups, and the chip with its memory side away from the memories.
- **How the tracks are checked, from now on in `make test`:**
  - `generators/board_tracks.py` (new; KiCad's Python) measures the routed board: from every pin that can drive a net to every other pin of it, the length of track, and how much of it on an inner layer. It writes `video-card-tracks.json` with the board file's SHA-256; `build_board.sh` runs it for a card that asks (`card.track_check`).
  - `sim_video_timing.py` runs its memory-side and bus-side checks a second time with those lengths as delays between the pins of each net (5.9 ps a mm outside, 7.0 inside; a net over 50 pF slows its driver by 0.03 ns a pF, the track at 0.12 pF a mm and a pin at 5 pF: estimates). It fails if the file is not the committed board's. With tracks of length nothing it gives D107's figures exactly. YACC1-D's engine is still used unchanged: the tracks are a subclass of its `Sim`.

    - `--cost` takes each family of nets' tracks away in turn and prints what each thin margin gains; `board_tracks.py --estimate` gives a tracks' file for a board that is only placed, so a placement is tried in a minute. That is how the new placement was found (nine tries).
- **What the margins hang on** (not the address path, whose tracks cost none of them anything): the hand-over of MAD from the buffers to the chip, on the dot clock's way to the buffers' flip-flop and from there to the 74ACT245s against its way to the counter and `2CLK`'s to the chip; text method A's latches, on `2CLK`, on the RAMs' chip enable (`CHR`, its inverter, `RAMCE#`) and on the RAMs' data lines; one memory off before the next is on, on `MOE#` reaching the three memories alike.
- **The new placement** (`card_video.py`): **the chip turned end for end**, its memory side towards the memories and its host side towards the bus logic; the counter under its `2CLK` pin; the oscillator between the counter and the buffers' flip-flop; that flip-flop's gate straight below it, between the RAMs and the buffers; a RAM with its text latch under it; the `MOE#` gate between the font and the RAMs. The two address latches and the two text latches are turned as well (their D pins towards what drives them). **Five chips therefore have pin 1 at the right**; each says `PIN 1 >` on the silkscreen.
- **The board as routed:** every run passes with its tracks. The least to spare, without the tracks and with: the MAD hand-over 0.2 and 0.5 ns; one memory off before the next is on 2.0 and 1.9; text method A's latches 2.9 and 1.3; the dot flip-flops 7.2 and 4.9; on the bus side a read's word 196 and 190 ns before the cycle ends (140 needed). Half a nanosecond is still thin, and the dot clock's way to three parts decides it: **a board routed again must be measured again**, which the build and `make test` now do.
- `pcb-check --layers 6`**:** no failure. Its two warnings: files not yet committed, and two footprints not in its table (the 27C256's, and **the HD63484P8's: a 64-pin DIP, rows 22.86 mm apart, to be held against a chip before ordering**).
- **The board generator, for every card** (each takes effect on a card only when that card is built again): a part can be turned end for end (`place_turn`, with the `PIN 1 >` mark); rows can be given their share of the room (`place_gaps`); a row that runs off the card's front edge stops the build (it collided with nothing and went unnoticed before); a connector's label stays on the card, with what plugs in there above it (`panel`); a header's or a jumper's two ends are named (`name: first|last` in its value); a line of text about a row of chips (`silk_note`).
- **Not covered:** reflections and crosstalk; the chip's own outputs against their load; what D107 lists.
- **Next:** Ken's checks by hand (the note's §8), then the order: the pre-order check on the committed files with `--tests`, and `--archive --fab F --order NUMBER` when it is placed.

D109 — The PS/2 firmware's time in the emulators, at gate level and in the kernel (2026-11-13; Ken: "go ahead with the PS/2 firmware timing")

The backlog's item from D107. The emulators' PS/2 section answered at once: the next byte in the latch the moment one was read, a command taken as it was written, the ATmega ready from the reset. YACC1-D's firmware (`embedded/video-ps2`, an Arduino sketch on an ATmega328P) is a loop that does one thing at a time, and the card's two flip-flops show it.

- **The model** (`emulator/p32io.h`, "the firmware's time"): what the firmware does next and when, one thing at a time, a command before a byte. The figures, from the sketch's own delays and the Arduino core's, rounded up: **FULL clear about 50 µs after a byte is read**; **BUSY about 30 µs after a command is written**, or until the command before is done; **3 ms for each command to a device** (the mouse off one, the keyboard's LEDs two), **half a second for a device's reset and self-test**; and **about 2.1 s from a reset**: the ATmega's start-up and its bootloader's second,

then the sketch's `start_all()`, a reset of the keyboard and of the mouse. Estimates for a part that has run on no hardware: software depends on FULL and BUSY, on none of the figures.

- **What the card and the sketch do, that the old model hid:**
  - a command written while BUSY replaces the one waiting (the card's latch simply takes it): the first is lost;
  - BUSY's flip-flop has no reset, so the sketch's `setup()` reads the command latch and drops what it finds: **the first command after a reset may be lost**;
  - **06 (flush) does not empty the card's latch**: a byte there stays, with FULL;
  - **03 (acknowledge) keeps the interrupt quiet until a device sends something new**, so bytes the firmware already holds arrive with FULL and no interrupt at all. Harmless while the model handed over every byte at once; on the card a handler that used it would leave the rest of an arrow key's sequence waiting for the next key.
- **One time base for all three models.** The delays are microseconds, counted in the timer's microseconds (an instruction each) in `p32emu` and `p32uemu`, and in microcycles at 3 or 4 MHz with `p32uemu -C`. At gate level (`sim_sys.py --video`) the same model decides in which clock the stand-in moves the ATmega's pins (LDDATA, -RDCMD, MD0-7, SRC, INTREQ), every bus clock, a wait state's too; FULL and BUSY are the card's own flip-flops there, and **after every clock the model's must equal them, or the run stops**. `vidcard_test` is identical to `p32uemu -C` instruction for instruction and cycle for cycle at 3 MHz and at 4, and so are four variants with one to four instructions more before the reads and the interrupts (run once by hand).
- `-Kr`: the section ready from the reset. The gate-level machine cannot run two seconds (six million clocks), so its ATmega always is, and `p32uemu -C` gets `-Kr` where the two are compared.
- **The kernel** (`kernel/video.c`): it wrote its two commands at once and waited for BUSY with the machine stopped, which is up to two seconds at boot and for ever with the ATmega's socket empty. Now three commands (02 interrupts off, the one that may be dropped; 05 the mouse off; 01 interrupts on) go out from the clock's tick, one each time BUSY is clear (`kbdtick`). It never used 03, and it reads a byte only with FULL: both were right already. P32W Unix on the video console runs as recorded.
- **The tests:** `video_test` waits for FULL and for BUSY as a driver must, and has two new cases (a command written over another is lost; the flush leaves the latch), 14 in all, on both emulators from the real start-up and with the card's timing at 3 and 4 MHz. `vidcard_test` (gate level) has the gap after a read, the overwritten command at the card's own latch, and a second PS/2 interrupt that comes when the firmware loads the next byte by itself. `ps2boot_test` (new) is the start-up: BUSY through the bootloader's second and the command dropped, the first byte after two seconds, a device's command holding the next one up.
- **Noted for YACC1-D** (its firmware, not changed here): 03 and 06 as above. A driver there that acknowledges with 03 would see the same.
- **On the bench:** the real figures (the note's §8).

P32W backlog

docs/backlog.md

Work Ken has asked for that is not yet scheduled, and smaller follow-ons. The next software milestone is **M7, the native toolchain** (D24, os-design.md). The hardware's next step is **the card netlists**, the CPU card first. Items move into a design document (and a decision, D-number) when they are taken up.

Hardware still to design

The board layouts (status 2026-10-31: every card routed)

All on 6 layers (D77), from `sh generators/build_board.sh <card>` (Freerouting 2.5 at `~/freerouting`: FRJAR, FRJAVA, else the system's Java 25+). Each run places the card fresh; a try that leaves connections open is improved only by placing again (ROUTE_TRIES) or another direction cost (`KT_AGAINST`, D80), not by routing the same board again: Freerouting cannot finish an already-routed board (D77), and it stops by itself once ten passes bring nothing.

- **Backplane:** done, placed and routed by `p32backplane.py` (D79): 0 DRC violations, 0 unconnected. Verify the sockets' 180° turn with a real card and socket before ordering.
- `docs/pdf/p32w-schematics.pdf` **is older than the cards' own schematic PDFs** (D90 changed the LEDs' pins): `sh generators/build_schematics.sh` joins them with `pdfunite` (poppler), which this Mac does not have. Run it on the M6.
- **I/O card:** routed (D80: a row per function), **0 unconnected**; its 14 DRC "violations" are cosmetic (DB9 outlines past the front edge, silk clipped by pads, the P32W footprint library not registered). Routed at `KT_AGAINST=1`.
- **Memory card:** routed (D80: its grid spread across the card), **0 unconnected, 0 violations**. Routed at `KT_AGAINST=1.5`.
- **MMU / data card:** routed (D80: a row per register), **0 unconnected, 0 violations**. Routed at `KT_AGAINST=1`, 2 h tries (`FRTIME=7200`).
- **Disk card:** done (D81, D82): **0 unconnected, 0 DRC violations**, matching its schematic (its seventh try at the D81 layout).
- **Video card:** revision 2 (the ACRTC card) done, D108: **0 unconnected**, its 1 DRC warning the LED footprint's (D90); matches its schematic. Its kept part is placed for the nets its thin margins hang on, and **a board routed again must be measured again**: `build_board.sh` rewrites `video-card-tracks.json` and `make test` runs the timing with it (docs/video-card-acrtc.md §4.9). (Revision 0's board, D81, D82, is in the history.)
- **Debug card:** done (D81, D82): **0 unconnected, 0 DRC violations**, matching its schematic. Its rows put the bus side left (sample latches, drive registers) and the capture side right; the first layout, two whole slices a row, stalled at 94.
- **CPU card:** done (D83): **0 unconnected, 0 DRC violations**, matching its schematic, on 6 layers with the general packer (its 195 chips at the tightest spacing), routed by Freerouting with its fixes for issues #1017 and #1054 built in. Rows by function and 8 layers had routed worse (D82).
- **Freerouting:** back to a release once one has the fixes for its issues #1017 and #1054 (pull requests #1021, #1055; D83); until then `generators/build_freerouting.sh` builds the one P32W uses.
- Route with `PARITY=` while a card has no generated schematic; `FRTIME=7200` gives a try 2 hours; one router at a time.

Found by the theory documents: to settle before fabrication (2026-10-30)

Writing the theory of operation of the I/O, disk, video, debug cards and the backplane (`docs/*-theory.md`), against the netlists and the gate-level simulations, turned these up; each document's last section has the detail. Ken's decisions; nothing is changed yet.

1. **The arbiter grants in the middle of a stretched cycle** (GAL ARB on the MMU / data card grants at any `CLK` edge with `-LOCK` high). A CPU cycle held by `-WAIT` is taken off the bus and run again, so a disk port's register sees two strobes: at gate level a PIO read during a DMA transfer got 7 extra strobes; a write would be done twice. Both emulators pass the same program. bus.md says a grant comes at a cycle boundary. (disk-card-theory §10, backplane-theory §11)
2. **The video card's `+F4` word read: bits 16-31 float** (no driver and, unlike XD0-15, no pull-downs). Two more 10k networks would fix it. (video-card-theory §14) **Settled with revision 2's netlist, D106 (2026-11-12):** that was revision 0; revision 2 drives zeros onto D16-D31 for that read (two 74HCT541s), and its gate-level check sees the whole word.
3. **The debug card's capture SRAMs: a write race**, and its GO / ARM / STEP flip-flops not cleared at power-on: **settled, D101 (2026-11-11)**: the write is in the sample clock's high half, and CRESN clears them. (debug-card-theory §8, §5)
4. **GAL IOSTB (I/O card): a possible glitch on `-RD` / `-WR`** between the two clocks of a byte access (a term `BCLK & !W1` in both). Nothing checks hazards. (io-card-theory §17)
5. **The disk card's internal bus CD0-15 has no pull resistors**, and IORDY reaches the sequencer GAL's registers unsynchronised. (disk-card-theory §10)
6. **The emulators against the hardware:** a TRELOAD write at the reload edge (the gate level reloads the new value, p32io.h the old one); the clock's interrupt (status bit 5, vector +5) is not modelled; a DMA bus error in DIR 0; BUSY in the PS/2 status. (io-, disk-, video-card-theory)
7. **The backplane:** with no card in slot 2 nothing drives the grants or holds the requests (their drivers and pull-ups are on the MMU / data card); `XCLK` is not terminated. bus.md's signal table says "7 highest" for `-IRQn`, against everything else (0 first). (backplane-theory §11)
8. **The I/O card's DB9s stand 2.84 mm further out than KiCad's footprint intends.** card_io.py puts pin 1 7.70 mm from the edge ("the shell through the panel"); the footprint's 7.7 mm "pin-PCB-offset" is the inner row (pins 6-9): its mounting holes, 9.12 mm from the edge, put pin 1's row at 10.54 mm, the flange at the board's edge. If the flange should sit at the edge, set 10.54 and route the I/O card again. (The video card's DE-15 had the same mix-up worse, its outer row off the board; fixed, D81.)
9. **The debug card's documents have drifted from its netlist**, and DRIVE cannot drive `-BGNT3-10`, `-FAST` or `-IRQ0-7`: **settled, D101**: the documents follow the netlist, the window-4 registers and the in-place ROM writing are dropped, and DRIVE reaches those lines. **D102**: the card alone no longer takes the host's word that the CPU and MMU / data cards are out; each ties a line to ground (`-CPUIN`, `-MMUIN`) that turns the drivers for its lines off. Still to do: the Pico's firmware and the host tool. (debug-card-theory §15)

A master pre-PCB-order check, for every system (Ken, 2026-11-05)

One command and one report for P32W, P8X, YACC1-D, System-5 and any later system, in a **new repo** (it makes "all 7" into "all 8": its name and visibility still to settle, then update `~/.claude/CLAUDE.md` and its master copy in `private-resources/claude`). To start after the clock counter and the kernel Makefile. FAIL blocks an order, WARN needs Ken's eye; a small settings file per board (name, rev, layers, exemptions); it ends with the by-hand steps (the layer pictures, the 3D render, the fab's viewer, the order form). It only reports: it never changes a board, so built and ordered cards are safe.

It gathers what is scattered today: `elf/tools/preorder_check.py` (10 checks, YACC1-D already calls it), `check_order_zip.py`, `tools/kicad/plane_check.py` and `compare_netlists.py`; `YACC1-D/tools/design_review.py`; `p8x/generators/check_card.sh`.

Ken's checks:

- every layer is in the gerbers;
- the gerbers match the schematic's netlist;
- no VCC-GND shorts;
- a **big warning** if VCC and +5V both exist and are not joined;
- every part's reference and value on the silk, and as far as possible both visible with the parts fitted;
- the card's name and rev level, large enough to read;
- `cottageworker.com`, and FAIL on any `onepagecomputer.com`;
- a warning if there is no POWER LED.

And, all accepted:

1. A copper-level proof: connectivity rebuilt from the gerbers and drill file alone, against the schematic. Today the zip check reads only the net names KiCad writes into the gerbers, not the copper. This closes the netlist and shorts checks above. The biggest piece (numpy is not installed: plain Python or C).
2. The rest of YACC1's review: unpowered parts, floating inputs, two drivers on a net, open-collector with no pull-up, too few bypass caps.
3. One-pin nets (how `-IRQ` against `IRQ` was caught on P8X).
4. Footprint against the real part: substitutions listed and compared with a table of known parts (P8X's Pico was a 2x20 header, its CR2032 holder the wrong one, and both passed everything).
5. The bus connector's pinout against that system's bus definition.
6. Inner layer order on 4- and 6-layer boards, and a tidy zip (P32W's zips also carry adhesive, paste, courtyard and fab layers).
7. Polarity: capacitors and diodes the right way round, `+` and pin-1 marks visible with the parts fitted.
8. Mechanical fit: board size, connector position, mounting holes, part height against the slot pitch.
9. Fab limits: track, gap, drill, annular ring, silk size; a closed outline; plated and non-plated drill files.
10. Power track width: no 0.2 mm track carrying a card's supply.
11. Soldering: a warning when plane pins have no thermal reliefs. **P32W's generators connect every plane pin solidly** (`ZONE_CONNECTION_FULL`), which will be hard to hand-solder on 6 layers: look at it before any P32W order.
12. A clean committed tree, the zip made from that commit, and the rev the same on the silk, the schematic, the README and the zip's name.
13. The system's own tests pass (P32W's `make test`, timing and GAL files; P8X's gate simulation).
14. The BOM complete, sockets included.
15. The order recorded in `FABRICATED.md` with the commit hash, and a warning if an ordered board goes again with no rev change.
16. Ken's layout rules as warnings: the front edge, connectors facing out, jumpers at an edge.

The order archive (Ken): when a board is set for order, what was sent to the fab is stored in that board's revision directory, with the matching schematic PDF (the good ones, chips connected), the layout PDF and the PNGs.

The card netlists

The cards are specified (architecture.md, D50) and the bus is generated (docs/bus.md, `generators/p32bus.py`); no card's netlist is designed yet. The order:

1. **The CPU card** (slot 1): sequencer and control store, instruction register and decode, register file, PC, the ALU module, the clock, the front panel.
2. **The MMU / data card** (slot 2): MAR, the split I/D TLB, the fault checks, MDR and byte steering, BADADDR, the bus master interface, the arbiter, `-LOCK`.
3. **The memory card**: 16 MiB of SRAM, the boot ROM, the window and ROM jumpers.
4. **The backplane**: the netlist is done (D68, docs/backplane.md): slot 0 and ten slots, the radial `-BREQ` / `-BGNT` lines to slots 2 and 0, J3 in slots 0–2 only, the power entry (D78), the pull-ups. Its layout to do.
5. **The device cards:** disk, I/O, video, then debug.

Each card follows the board rules (D19, D21, D47, D48): every part through-hole, bypass caps on 0.1" footprints, switches, LEDs and cable connectors at the front edge.

Gate-level simulation of the cards

Done for the CPU card (D61: `generators/sim_cpu.py`, in `make test`). Next the MMU / data card the same way, then the two together over the local link (J3), and a 5 MHz CPU card (§ "5 MHz without a wait state").

A jumper for the clock source: oscillator or an external clock

The CPU card's 15 MHz oscillator drives the clock GAL's clock input (D17). Add a jumper choosing between that oscillator and an **external clock brought in on a pin** (a header near the card edge, D48), labelled on the silkscreen with each position (INT OSC / EXT CLK). Uses: a slower or faster crystal for bring-up, a signal generator, or a clock from the control panel (below). To decide with the CPU card: the input's buffering (a 74HCT14 Schmitt input, with a pull so an open pin cannot float), whether the external pin is on the front edge or a header, and how it relates to the debug card's fast clock (that one drives the bus `CLK`, after the GAL; this one feeds the GAL).

A probe header on the CPU card for the debug card

The debug card's logic analyzer reaches only what is on the backplane. A **probe header on the CPU card's front edge** (D48) lets it see signals inside the card: the microword, the sequencer's state, the internal data bus. To decide with the netlist: which signals, the header's size, and how the cable runs to the debug card.

5 MHz without a wait state

The memory microcycle (MAR → TLB → physical address → memory → R, about 200 ns) does not fit a 200 ns clock with the control store's share. At 5 MHz the MMU / data card asks for **one wait state on every access** (`-WAIT` on itself, no microcode change); 3 MHz needs none. Measured with `p32uemu -W N` on the three-bus datapath (P32W Unix booted, root logged in, `halt`; 7,049,876 instructions with the address adder and the D59 instructions; 47% of all microcycles are bus cycles):

Clock, waits	Cycles	Cycles / instruction	Speed
3 MHz, 0	22,427,463	3.18	0.94 MIPS
5 MHz, 1 on every bus cycle	32,868,985	4.66	**1.07 MIPS**
5 MHz, 0 (translation earlier)	22,427,463	3.18	1.57 MIPS (+47%)

Moving the address translation earlier, into the microcycle that loads MAR, removes the wait state and is worth another 47%. It changes the MMU / data card and the microcode; to design after the wait-state version works.

The ALU's division at 5 MHz

The ALU core's worst latency is 18 of its 108 MHz clocks (167 ns, `fpga/alu/tb_p32alu.v`), set by division; every other function is a few clocks. At 3 MHz the result is on R in time; at 5 MHz (200 ns, less the operand and result transfers) a division needs its result cycle held a clock (`busy` into the sequencer, as `STALL`), or the divider made faster.

The VGA timing parts (done, D67)

Counters (74F163) and fast GALs on the 25.175 MHz dot clock; the card's RAM 16 bits wide, shared in four slots of each character time (text, the bus's two halfwords, the overlay). docs/video-card.md.

The power budget

Done (D69, docs/power-budget.md, generated by `generators/p32power.py`): about 14 A typical, 26 A at most for the whole machine on +5 V, plus the drives. Left: measure the Tang Nano 20K and the Picos, and replace the estimated rows with datasheet figures as they are checked.

An external control panel

The machine's controls are on the CPU card's front edge (D50): the RUN / instruction-step switch, the RESET and STEP buttons, and the RUN and HALT LEDs. A separate control panel, cabled to the card, would have room for more: an address and data display, a clock-speed selector, and the front-panel ports' switches and lights (the I/O card's ports at `FF20` / `FF21`). To decide then: the cable (a header on the CPU card in place of the parts), what the panel shows, and whether the debug card's probes and fast clock join it.

For the MMU / data card: a bigger or 2-way D-TLB

A hardware question for Ken, not decided. The D-TLB (D12, D13: 64 entries, direct-mapped) costs about 8% of the time of a compile (see "Memory-system statistics" below). Doubling it to 128 entries (one more index bit on the D-TLB's SRAMs) cuts its misses about elevenfold; making it 2-way (a second bank, two comparators, an LRU bit per set, and TLBW choosing the way) about eightfold, with the same 64 entries. The I-TLB can stay as it is. The emulators, the microcode and the kernel's refill would change with it (a decision of its own).

A DRAM memory card (later; Ken: "later, for now stick to Static")

For density: 30-pin or 72-pin SIMMs (4 or 16 MiB a module, DIP-free but through-hole sockets) or DIP DRAMs, with a refresh controller and the RAS / CAS timing on the card, answering in the same 16 MiB windows (D52) and stretching its cycles with `-WAIT` as needed. To weigh against D60's static cards (simple, no refresh, 16 MiB a card) when memory runs short.

Faster disk transfers: DMA from the disk card

The CPU moves every 16-bit halfword through the CF data register: about 140 KB/s, and the CPU does nothing else meanwhile. That bounds swapping (below) and all file I/O.

- **Software first (with swapping):** the sector loop in hand-written assembly (about 2×, ~280 KB/s; docs/bench.md), interrupt-driven waits (the CPU runs another process while the card is busy), READ/WRITE MULTIPLE. No hardware.

- **Then DMA: the engine is on the disk card (D75, 2026-10-28)**, verified at gate level and modelled in `p32emu` / `p32uemu`; per sector, the driver starts it once the drive has the data (docs/disk-card.md §4.1). To do: the driver (`kernel/`), using it for whole sectors and the interrupt instead of the copy loop; measure swap and file-I/O times. Later, if wanted: the drives' own multiword DMA, by reprogramming the card's GALs.

Analysis: component speeds for a faster clock

Done as an analysis: docs/speed.md (2026-10-27): the clock jumper, the SRAM control store, 5 MHz at worst case, DMA, the overlapped fetch. The clock jumper is built into the design (D74, 2026-10-28): 4 MHz at typical delays.

The tool exists now (D71, docs/timing.md, `generators/p32timing.py`): it gives each path's delay and the fastest clock the machine allows, worst case and typical; trying faster parts is a change to `p32delays.py` and a rerun. First the 3 MHz failures it found (D71). (Analysis only, nothing to change yet.) The clock is 3 MHz, with 5 MHz the design goal (D50; microarch.md §8). Find the critical paths card by card and what each faster part would buy: 74HCT → 74AC / 74ACT / 74F / 74ABT (through-hole availability, edge rates and ground bounce, drive into the backplane), the microcode flash (55 ns, D18; or copied to fast SRAM at reset), the register-file SRAM (15 → 10 / 12 ns), main memory SRAM and ROM (55 ns → 10–25 ns SRAM, ROM shadowed in RAM), the TLB SRAM and tag comparator, GAL speed grades (-25 / -15 / -10 / -7), the ALU FPGA's 5 V ↔ 3.3 V buffers and latches, and the backplane itself (propagation, loading, settling, and whether its long lines need termination). Structural options: the earlier translation (above), wait states only for slow devices, or a pipelined fetch. Result: a table of path → parts → delay → achievable clock, and the cheapest steps to 6, 8 and 10 MHz.

Idea (parked): a PDP-11/23 clone on this hardware

Could P32W's cards, or variants of them, run PDP-11/23 microcode and so the 11/23's operating systems (RT-11, RSX-11M, RSTS/E, Unix V6 / V7) and their software? First analysis: pdp11-analysis.md: feasible as a second microcode set with CPU card, MMU / data card and ALU FPGA variants; the hard parts are the 64-byte MMU relocation and DMA disk controllers. Emulator first (in `p32uemu`, checked against SIMH).

Devices: what is left

Networking

The network is a Pico 2 W with a W5500 behind a mailbox (D45, docs/net-mailbox.md), doing TLS too. **Done:** the emulator model, kernel sockets, `ifconfig`, `ping`, `telnet`, `fetch`, `telnetd`, `ftp`, `ntpdate` (D45, D46). **Left:**

- **The coprocessor's firmware:** the mailbox's other side, then the W5500 and TLS, then the bench.
- **httpd:** static files, and CGI (shell, awk or BASIC scripts writing HTML) for status pages (uptime, processes, `df`, the TLB statistics); then directory listings, downloads and uploads. **No HTTPS on P32W:** for access from outside, a proxy or VPN on another machine, never P32W exposed directly.
- `select` / `poll`, to wait on several descriptors at once.
- **SLIP** over a UART (RFC 1055), as a second interface under the same stack, if a machine without the network ever needs one.

Wi-Fi: joining a wireless network

The Pico 2 W's CYW43439 radio gives Wi-Fi beside the W5500's Ethernet. The mailbox speaks sockets, not Ethernet, so Wi-Fi changes the coprocessor's firmware, not P32W's kernel. To do:

- **The firmware's socket layer** over the Pico SDK's lwIP for the radio, the coprocessor using whichever link is up.
- **The protocol gains a few commands:** scan for networks, join one (network name and password), the link's state and signal. P32W gains `ifconfig wifi NAME` (or a `wifi` command), the password kept in a file only root reads (`/etc/wifi.conf`), and a join at boot from `/etc/rc`.
- **The emulator's model** pretends: a list of networks, any password accepted, the Mac's own network behind it.
- **Security:** WPA2/WPA3 is done on the radio; the password must not be readable by other users on P32W.

USB drives

Done (D43, D44, docs/usb-design.md): the CH376S model in the emulators, the driver (`/dev/usb`, `/dev/rusb`) and the FAT tools (`dosdir`, `doscp`, `dosrm`, `dosmkdir`, `dosformat`). **Left:** checking a real module's pinout and USB socket position against the I/O card's front edge, then trying real sticks on the bench; later, FAT mounted directly, which needs a VFS (a filesystem switch).

Graphics for programs

The video card (revision 2, D103) draws 640 × 480 dots with an HD63484, from commands; none of the picture is on the bus, so there is no memory to map into a process. The console uses the middle 400 lines. To let programs draw: **a device, `/dev/video`, that takes the chip's commands** (root or a `video` group only; the driver keeps the console's and a program's words apart and gives the screen back), a small C graphics library over it (points, lines, rectangles, fills, text; circles and arcs once the emulators' model has them), and BASIC's graphics statements (SCREEN, LINE, CIRCLE, PSET) on top of it. A second port that would put the picture on the bus as well, so that it could be mapped into a process, was looked at and left off (D106; docs/video-card-acrtc.md §6a).

A windowing system on the video console

After the overlay: overlapping windows, a terminal window running a shell (several sessions on one screen), mouse and keyboard focus, menus, and an API for programs. P8X has a resident window manager and its design (`docs/p8x-wm-design.md` in the P8X repository: chrome, painter's repaint, launching programs, per-window state). On P32W, with protected processes, it becomes a **server process** (`/etc/wm`) that clients talk to through pipes or a device.

getty discarding typed-ahead input (optional)

`getty` could discard input typed ahead on a terminal before it shows the login prompt (a flush of the terminal's input queue), so that stray keys do not reach `login`. Optional.

Operating system and tools

Swapping: whole processes to disk

When page frames run out, `fork`, `exec` or `sbrk` fails (or a growing stack kills its process). V6/V7-style swapping writes an idle process's pages (data, bss, heap, stack; text can be re-read from its program file or stay while others share it) to a swap area on disk, frees its frames, and reads it back when it can run. More processes than fit in RAM; no process bigger than RAM.

- **Software only:** the MMU, the page tables and restartable faults are there. In the kernel: a swap area (a reserved region or partition), a map of each swapped-out process's swap blocks, and a swapper that picks a victim (the longest-sleeping) when frames run out and brings processes back.
- **How long a swap takes** (each way, at 3 MHz; only pages a process touched are written):

swapped	C loop (~140 KB/s)	assembly loop (~280 KB/s)
idle `sh`, ~150 KB	~1 s	~0.5 s
a 1 MB program	~7 s	~3.5 s
`cc` on `lvm.c`, ~3 MB	~20 s	~10 s
a full 8 MiB process (D34)	~55 s	~28 s

With 16 MiB of RAM or more, everyday use fits without swapping; it is the safety net for heavy moments. The swapper should keep a process in for a good while once it is in, so two big jobs do not thrash.

- **Faster disk transfers first** (they speed up every file too): see above.
- `ps` / `top` should show swapped-out processes; `p32emu -m`'s statistics should count swaps.

Demand paging (later, if needed)

Single pages go to disk and come back on a fault, so a process can be bigger than RAM. Needs a replacement policy (the clock algorithm) and "used" and "dirty" information, which the software TLB does not keep: the kernel emulates them by mapping pages read-only or not at all and noting the faults. A process much bigger than RAM thrashes. Only after swapping, and only if a real need appears.

A debugger: source level

`db` (machine level: breakpoints, stepping, finish, backtrace, registers, memory, disassembly, symbols from the a.out) runs on `ptrace` (unix.md). To do: **source level** (`p32cc -g` emitting a line table and the locals' frame offsets / register variables, kept by the linker in the a.out; `print var`, `break file:line`), `core` **dumps** on fatal signals for post-mortem use, and a `gdb`**-style stub in** `p32emu` so the same debugger runs on the host against the emulator. Best done with M7, whose object format must leave room for the debug sections.

Measure hand-written assembly against C

Inner loops are measured (docs/bench.md): C is 1.2–2.3× slower than assembly, and `memcpy` and `memset` are in assembly. Still to do: the trap / system-call path, whole commands (`wc`, `grep`, `sort`), a boot-plus-login figure, and the `n--` test. Report the ratio per piece and per whole program, and whether the gap is in the compiler (worth improving it) or inherent (worth a few assembly routines in libc and the kernel). The benchmarks live in `software/bench/`, so they can be rerun as the compiler improves.

MicroPython

MicroPython is `/usr/bin/python` (D35, D36, D39). Left:

- `pow` **correctly rounded** (about 93%: y·log x needs more than 62 bits, a two-part log).
- `tgamma`, `lgamma`, `erf`, `erfc` **in 62 bits:** musl's code rounds at every step and is a few ulp off (`tgamma(5.5)` 1.8 ulp). Rewritten on softfp's 62-bit core, as `sin` and `atan` are (D38), they would be correctly rounded about 99.5% of the time. Rarely used: later.
- `select` **and** `asyncio` need `poll()` in the kernel.
- **Startup takes 3.1 s** (D37); the rest is the CF loop itself (DMA).

Lua

Lua 5.4 is `/usr/bin/lua` and `luac` (`software/lua/README.P32W`). Left:

- **Deep recursion is slow:** 4,000 levels (each making a small table and a string) take 36 s of P32W time. Find out where it goes (stack reallocation? the incremental GC?).
- **Startup takes 4.1 s.** Measure where the time goes.

BASIC

`/bin/basic` (D27; basic(1)). Left, from basic(1)'s BUGS:

- the graphics statements (SCREEN, LINE, CIRCLE, PSET, COLOR), with the overlay (above);
- INKEY$ (always ""), the MID$ statement, PRINT USING's `^^^^` and `**$`, CHAIN / COMMON / AUTO / EDIT, random-access files (FIELD, GET, PUT), and a DEF FN of more than one line.

vi

`software/cmd/vi.c` (unix.md). Later: line wrapping, named registers, marks, `:g`, `:set`.

The SQL database

`/bin/sql`, `sqlui` and `libsql` (sql(1), sqlui(1), sqldb(3)). Left:

- a decimal or floating-point type (BASIC uses doubles, D27; SQL has none, and AVG is truncated to an integer);
- joins of more than two tables. (Not attempted: subqueries, UNION, views, ALTER, defaults, foreign keys.)

Commands

- `dc`, and `grep -E` (grep has only -vnci).
- `ps -l` (a long listing: priority, nice, wait channel).

Known bugs (small)

- **sh:** no `until`, `trap`, `$@`, PS1 or job control; words split at spaces only (no IFS); `"a b"$X` is split at the quoted space; past the word limit the rest stays one word, silently; `sh -c`'s `$0` is the command text; `done > file` cannot redirect a loop (`{ ...; } > file` can); `wait` takes no pid; `exec > file` does nothing. (sh(1)'s BUGS.)
- **Kernel:** `#!` is in libc's execve, not the kernel (no setuid scripts); `wait` on a stopped traced child marks the stop reported before the copyout, so an EFAULT loses it.
- **du** counts blocks from `st_size` (there is no `st_blocks`), so a file with holes is overcounted.
- **init**'s back-off is a `sleep(60)` in its main loop, so for that minute it does not reap or respawn the other terminals.
- **The kermit test is intermittent:** about 1 run in 3, under load, `kermit -s` (P32W sending to the host) ends with "the line closed". It is probably timing in `tools/pykermit.py` or the test, not Kermit on P32W.

Follow-ons

- **The native cc's memory:** all the tokens exist before parsing starts; running the preprocessor and the parser together (a token at a time), or smaller tree nodes, would cut it further, if ever needed.
- **Code generation:** constant folding of addresses, compare-and-branch on immediates, leaf functions without a frame.
- **Disk writes** (10 MiB takes 143 s of kernel time): the buffer cache's lookup is a linear search (a hash), and the CF transfer loop in assembly.
- **Interrupt-driven disk transfers** (the disk card's interrupt, level 1) instead of polling.
- **LBA48** in the CF driver (and the emulator's CF model) for cards past 128 GiB.

- **Partitions** on a disk (several filesystems on one big card or drive).
- **Speed on the hardware:** measure a boot with login; hand-written transfer loops for the disk driver and `memcpy`.

Memory-system statistics: what they show

`p32emu -m` and the kernel's counts (`time -l`, `top`'s `Paging:` line) show that the D-TLB thrashes on big programs. Compiling `ls.c`, a sort and the shell tests: 24.6 M data accesses, 203,373 D-TLB misses (0.82%, 97% of them conflicts), about **8% of all instructions spent refilling**. The same accesses miss 25,826 times in a 64-entry 2-way TLB, 18,648 in a 128-entry direct-mapped one, 7,504 in 128 2-way; a hashed index (VPN ^ VPN >> 6) is worse (235,220). The I-TLB is fine (0.01%). See the D-TLB item above.

A PDP-11/23 on P32W hardware — analysis

docs/pdp11-analysis.md

Status: **an idea, parked** (Ken, 2026-10-14): analysis only. Not a decision; nothing is built. Question: could this hardware, or something very close to it, be a PDP-11/23 clone that runs the 11/23's operating systems and their software?

Short answer

Yes, and P32W is well placed for it. P32W is **microcoded**, as the 11/23 was (the F-11 chipset ran PDP-11 microcode on a general datapath). The natural route is a **second microcode set** on the same backplane, memory, disk and I/O cards, with **variants of three parts**:

- the **CPU card's decode and sequencer**: PDP-11 instruction decode, a sequencer that can chain operand sequences;
- the **MMU / data card**: PDP-11 relocation (PAR / PDR), not a TLB;
- the **ALU FPGA**: 8- / 16-bit condition codes, the PSW flags, PDP-11 MUL / DIV / ASH / ASHC / SWAB / SXT.

The hard parts are not the instruction set. They are the PDP-11 MMU's 64-byte relocation, DMA disk controllers, and making the devices look exactly like DEC's.

Three ways to do it

	What	Verdict
A	PDP-11 microcode on P32W's datapath (other control-store images, some card variants)	**Recommended**: "this hardware or very close"
B	A PDP-11 emulator in software, under P32W Unix (SIMH-style)	Easy but ~50–100 P32W instructions per PDP-11 instruction: thousands of PDP-11 instructions a second. Too slow.
C	A PDP-11 core in the Tang Nano 20K (open cores exist, e.g. pdp2011)	Works, but the TTL becomes a backplane. A reference, not the goal.

What already fits

- **Register file:** 32 × 32-bit with a banked stack pointer. PDP-11 needs R0–R7 and separate kernel / user R6 — the same idea as P32W's banked r15 (the bank rule moves from 15 to 6). T0–T7 are microcode scratch.
- **Memory:** byte addressing, byte enables, byte / halfword load steering. 16-bit words and byte operations map directly; the PDP-11 odd-address trap is P32W's MISALIGNED.
- **22-bit addressing** (4 MB) fits P32W's RAM.
- **Traps and interrupts:** microcode pushes PS and PC and loads the new PC and PS from a vector (4, 10, 14, 20, 24, 30, 34, 250, devices). P32W's eight IRQ lines map to vectors and priorities through a small ROM.
- **T-bit trace trap:** a test at the end of each instruction.
- **The tools:** `p32ucode.py` and `p32uemu` (the emulator runs the real flash images) are what a second microcode set needs.

What changes

Area	P32W today	PDP-11 needs	Change
Decode	dispatch on IR[31:24]; 16 steps per instruction	16-bit instructions, irregular layout, 8 addressing modes *per operand* (up to two memory indirections each)	a decode ROM from the instruction word to an entry; a sequencer that chains source-operand → destination-operand → execute sequences (microcode subroutines). **The biggest change on the CPU card.**
PC	a +4 counter, not in the register file	R7 *is* the PC (`(PC)+` immediates, PC-relative modes), +2	a +2 counter variant; register 7 aliased to the PC
r0	always 0 (write sink)	an ordinary register	the zero rule off in PDP mode (a gate in the CPU card's decode)
Condition codes	none: branches compare registers	PSW N Z V C at **byte or word** width, set by most instructions	**in the FPGA:** 8 / 16-bit flag modes, the flags held inside, a "branch taken" output. The FPGA ALU (D21) pays off; its pins are tight — to check.
MMU	4 KiB-page TLB, refilled by software	8 × 8 KB pages per mode; PAR = base in **64-byte units**; PDR = length in 64-byte blocks, expansion direction, access	**The main obstacle.** OSes place pages on 64-byte boundaries (Unix allocates in 64-byte "clicks"), which 4 KiB frames cannot represent. A real relocation unit: a small PAR / PDR file indexed by mode and VA[15:13], a 22-bit adder, a length comparator — ~15–20 chips, an **MMU / data card variant**. (In microcode it would cost cycles on every access.)
I/O	P32W UART, timer, RTC, CF	DEC registers at DEC addresses: DL11 console (177560), KW11-L clock (177546), a disk controller (RL11 / RLV12 for RL02, or MSCP)	**The other hard part.** DL11: a UART with a GAL rearranging the status bits. **DEC disk controllers are DMA bus masters.** P32W's bus has bus masters too (radial `-BREQ` / `-BGNT` to the arbiter, D50), so a Pico-based device card can emulate the controllers over an SD / CF image and do the transfers by DMA.
Console	the monitor ROM	console ODT in microcode; a boot ROM	ODT in microcode or ROM; a small boot ROM
Floating point	none	the optional KEF11 (FP11 instructions, six 64-bit accumulators)	optional, as on the real 11/23; RT-11 and RSX run without it; V7 Unix's `cc` uses FP11 for `float`. Later, in the FPGA.

Software

- **Operating systems** that ran on the 11/23: RT-11 (SJ / FB / XM), RSX-11M and M-Plus, RSTS/E, Unix V6 and V7, Ultrix-11, 2.9BSD. (2.11BSD prefers separate I and D space, the 11/44 / 11/73 class — to check whether it runs on an 11/23.)
- **Licences:** the old Unix versions are free under Caldera's 2002 licence; DEC's own systems have historically been available under hobbyist licences — Ken's call.
- **Disk images** for all of them exist for SIMH, mostly RL02 and MSCP: the emulated disk controller decides what boots. RL02 is the simplest; MSCP is more work and more common.
- **Checking:** SIMH's PDP-11 (set to an 11/23) as the reference, instruction by instruction (as `p32uemu` against `p32emu`); DEC's XXDP diagnostics for the KDF11 to test the CPU as DEC did.

Speed (an estimate, to be measured)

P32W averages ~6 microcycles per instruction at a 3–5 MHz clock (D50). A PDP-11 register-to-register `MOV` would be ~4–5 cycles; memory modes add ~2–3 per operand. Roughly **0.3–0.5 million PDP-11 instructions a second — the same class as a real 11/23**, with the MMU in hardware.

A path, if taken up

1. A decision (D-number): the target (11/23, 22-bit, no FPU first), the devices, dual-mode cards or separate "P11" card variants.

2. **Emulator first:** a PDP-11 microcode set in `p32uemu` over a card model with the PDP-11 MMU and devices; checked against SIMH; XXDP.
3. Boot Unix V6 / V7 and RT-11 from an emulated RL02.
4. Hardware variants last: the CPU card's decode / sequencer, the MMU / data card, the FPGA bitstream, a device card.

Steps 1–3 need no PCB work.