ISA: Hot, Dead, Missing

What the instruction set actually gets used for — the hot, the dead, and the missing. Census taken 2026-09-07 against the vc4 kernal.

The SD-8516 has **163 live opcodes**, or 150 distinct mnemonics once the ``_IMM`` and ``_REG_IMM`` encodings are folded into their base instruction. This page counts how often each one appears in the kernal source, and argues from that about what to add and what to drop.

Why this matters: dispatch is the bottleneck

Measured on an unthrottled build, the interpreter runs at:

workload	branch density	speed
mixed — loads, stores, ALU, moves	1 branch per 8 instructions	580 MIPS
tight loop — INC/CMP/BNZ	1 branch per 3 instructions	470 MIPS

The tight loop is **slower**, which is the whole argument for CISC on this machine. What limits a switch-dispatch interpreter is not how complicated an instruction is — the jump table absorbs that — it is **how many instructions you execute**, and especially **how many of them are branches**. Every guest branch costs a mispredicted indirect jump on the host plus a non-sequential instruction fetch.

So an instruction earns its place by the number of *dispatches* it removes, not by how little work it does. At roughly 6-7 host cycles per dispatch, an instruction that replaces a five-instruction loop body saves about 30 cycles every time round the loop.

This is the same conclusion the benchmarks page reached from the other direction: kernal 0.7.2 measured 750 MIPS and 0.8.3 measured 500, and 0.8.3 was **twice as fast**, because CASETAB had replaced a chain of CMP/JZ. MIPS is a poor proxy for speed on a CISC.

Hot — the workhorses

Occurrences in the kernal source:

Instruction	Uses	Instruction	Uses
CALL	1203	INC	477
POP	1147	ADD	301
CMP	1000	JNZ	279
PUSH	840	INT	271
JMP	751	JZ	201
MOV	682	JC	185
RET	681	SKPC	181
BZ	628	DEC	176

Two patterns dominate everything else:

- **Call and return with register save/restore.** CALL + RET + PUSH + POP is roughly 3,900 occurrences. The kernal is full of `PUSH3 A, B, C` … `POP3` prologues.
- **Compare then branch.** CMP appears 1,000 times and is followed by a branch almost every single time; the conditional jumps and branches together come to about 1,400.

Those two patterns are where any new instruction should aim.

Dead — never used

Of 150 mnemonics, **56 appear nowhere in the kernal**. (A further 15 that look unused — LD_MEM, LD_IDXR, ST_FS, ST_PD and so on — are not mnemonics at all. They are encodings the assembler selects from bracket syntax: you write `LDA [$1234]`, not LD_MEM. Those are alive and well.)

The genuinely unused, grouped by what they were for:

- **Compound comparisons** — JA, JAE, JG, JGE, JL, JLE, BA, BAE, BG, BGE, BL, BLE, BNA, BNAE. Fourteen opcodes, zero kernal uses.
- **Multi-precision arithmetic** — ADDC, SUBC, SHLC, SHRC, ROLC, RORC.
- **Forth and loop experiments** — LIT, TTOS, LSTEP, LSTEPM, DPUSH, DPOP.
- **Register-indirect control flow** — JMPR, CALLR.
- **Single-flag set/clear** — most of the SEx/CLx pairs (SED, SEI, SEN, SEV, CLI, CLN, CLV, …), plus CLRF and SSI.
- **Odds and ends** — MOVSX, NOP, BREAK, PUSHF, POPF, CMPC, CVTNA.

> **Do not remove the compound comparisons or the carry-chain arithmetic without checking the LLVM backend first.** Those landed alongside the backend work, and compiled C — not the hand-written kernal — is their likely consumer. A 16-bit machine doing 32-bit arithmetic needs ADDC/SUBC whether or not the kernal ever writes them.

The safest removals are the single-flag SEx/CLx opcodes, since SETF/CLRF with a mask already covers the same ground, and the Forth and loop experiments, which were tried and not adopted.

Dormant — built, but never called

Not dead, exactly. These exist, work, and are simply not being called:

- PCIRC, PCIRCF, PRECT, PRECTF, PBLIT, PCLEAR — **INT 18h uses none of them.** It draws circles in software, with its own ellipse arithmetic. The accelerated opcodes sit idle beside it.
- CVTNA — converts a nibble to an ASCII hex digit in one instruction. Zero uses; the hex monitor formats digits by hand.
- PPIXEL (3 uses), PLINE (1), CASETAB (2), SCANQUE (2), INSQUE (1), REMQUE (1).

The cheapest performance available on this machine is not a new instruction. It is calling the ones that already exist.

Missing — what is not there

Ranked by what the census says would pay, not by what would be fun.

1. Fused compare-and-branch

```
CBZ  reg, imm, target
CBNZ reg, imm, target
```

CMP is the third most common instruction and is followed by a branch nearly every time. Fusing them halves about 1,000 dispatch pairs, and — more importantly given the measurements above — it **reduces branch density**, which is the thing actually limiting the interpreter. ARM has CBZ/CBNZ; RISC-V folds the comparison into every branch. This is the single highest-value addition.

2. Call with a register-save mask

```
CALLM mask, target
RETM  mask
```

CALL/RET/PUSH/POP is the most-executed pattern in the kernal, and the PUSH3/POP3 prologue-epilogue is written out by hand in hundreds of routines. Folding the mask into the call and return removes two to four dispatches from every function call. The LLVM backend emits the same shape for every function it compiles, so both sides of the machine benefit.

3. MEMSET / FILL

There is MEMCOPY but no fill. Every screen clear, buffer initialisation and structure zeroing is a hand-written loop. PCLEAR covers the framebuffer only.

4. Widening multiply

MUL truncates to the destination width and only records overflow in the carry flag, so the high half of a product cannot be recovered. That makes **16.16 fixed-point arithmetic impossible**, which is the format games want for rotation, scaling and physics. Either MULH (high half) or a 16×16→32 widening form into a register pair would open that up.

5. Scaled index addressing

```
LD reg, [base + idx*2]
```

Every walk over a table of 16-bit entries currently needs a shift first, and the machine is full of such tables — the interrupt jump tables, BASIC's variable storage, the palette.

6. Conditional move

`CMOV` turns a compare-branch-move sequence into one instruction and removes a guest branch. Lower value than the above, but it is on the same theme: fewer branches, fewer dispatches.

Caveat: these are static counts

Everything here counts **occurrences in source**, not **executions at runtime**. An instruction inside a hot inner loop matters far more than one that appears in fifty cold error paths, and this census cannot tell the difference.

A dynamic profile would be strictly better and is not hard to produce: a 256-entry counter array in `cpu_step()`, incremented per dispatch, dumped at `HALT`. Running the kernal boot, a BASIC program and a game under that would give a real histogram — and might well reorder the recommendations above.

Reproducing the census

Parse the `OP` enum from `arch.h`, ignoring commented-out lines, fold the `_IMM` and `_REG_IMM` suffixes into their base mnemonic, then count occurrences of each at the start of a line across `kernal/*.sda` and `kernal/int/*.sda`, with comments stripped.