

Emulation Benchmarks

The Chart

Legend:

- Green: The SD-8516 is capable of emulating this level of performance on an i7-12700 (Geekbench 6 baseline).
- Light Green: Capable, but only with extensive use of the PPU and APU.
- Light Gray: The SD-8516 cannot reach this level of performance, but can reach at least 60%, so ports are possible but may be heavily constrained. Optimization in Assembly may be required.
- Dark Gray: The SD-8516 cannot emulate this system. This is almost always because it is designed as a single threaded system with no 3D acceleration. As an emulator, modern computers can often run emulated systems at near-native speeds. But even if we pass-through the GPU via SDL2 we are unlikely to be able to emulate this class of system because we don't have multi-threading. Given time it is likely computers will continue to increase in speed and we will be able to emulate these systems performance; if I ever add the ability to multitask, these limitations will almost certainly disappear.

| Year | System | CPU | Width | Approx MIPS | RAM | Graphics | Audio | Notes |
|------|---------------|----------------|--------|-------------|-------------------|-----------------------------------|--|--|
| 1977 | Atari 2600 | MOS 6507 | 8-bit | 0.30 | 128 B | 160×192, ~128 colors (TIA tricks) | | No framebuffer; cycle-exact emulation needed |
| 1979 | Intellivision | CP1610 | 16-bit | 0.35 | 1 KB | 159×96×16 | | 16-bit CPU but very slow clock |
| 1980 | VIC-20 | MOS 6502> | 8-bit | 0.45 | 5 KB | 176×184×16 | | Simple video chip; CPU-bound |
| 1981 | BBC Micro | MOS 6502 | 8-bit | 1.00 | 32 KB | 640×256×2 | | 2 MHz CPU; very tight timing |
| 1982 | ColecoVision | Z80 @ 3.58 MHz | 8-bit | 0.58 | 1 KB + 16 KB VRAM | 256×192×16 | | Same VDP as MSX |
| 1982 | C64 | MOS 6510 | 8-bit | 0.36 | 64 KB | 320×200×16 | SID | VIC-II steals cycles-slower than VIC-20 CPU-only |
| 1983 | NES | Ricoh 2A03 | 8-bit | 0.50 | 2 KB | 256×240×25 | 2×PWM, 1×triangle, 1×noise, 1× DMC (4-bit PCM samples) | Heavy PPU timing-emulation harder than C64 |
| 1983 | Apple IIe | MOS 6502 | 8-bit | 0.50 | 64 KB | 280×192×6 | | No sprites; CPU-driven graphics |
| 1985 | C128 | MOS 8502 | 8-bit | 0.75 | 128 KB | 320×200×16 | SID | Faster CPU but still VIC-II limited |

| Year | System | CPU | Width | Approx MIPS | RAM | Graphics | Audio | Notes |
|------|------------|------------------|--------------|-------------|------------------|------------------------|---------------------------------|--|
| 1987 | Amiga 500 | 68000 @ 7 MHz | 16/32 | 4.5 | 512 KB-1 MB | 320×256×32/64/4096 | Paula (DMA-driven PCM playback) | Custom chips dominate emulation cost |
| 1991 | SNES | Ricoh 5A22 | 16-bit | 1.5 | 128 KB + VRAM | 256×224×256 | Advanced Sample Playback | Slow CPU; complex PPU & DMA |
| 1991 | 386SX | 80386SX @ 25 MHz | 32-bit | 10 | 4 MB | 320×200×256 VGA | SB Pro | Software rendered; cache key for speed |
| 1992 | Amiga 1200 | 68EC020 @ 14 MHz | 32-bit | 14 | 2 MB | 320×256×256 | Paula (DMA-driven PCM playback) | Much easier CPU than SNES to emulate |
| 1992 | 486 Gamer | 486DX2-66 | 32-bit | 54 | 8 MB | 640×480×256 (S3 ViRGE) | SB16 | ~20 FPS software Quake; 3D accel emerging |
| 1994 | PS1 | MIPS R3000A | 32-bit | 30 | 2 MB + 1 MB VRAM | 320×240 | SPU | Geometry-heavy; no GPU T&L |
| 1994 | Pentium 90 | 586, 90 MHz | 32-bit | 90 | 32 MB | 640×480×16M (PCI VGA) | SB AWE32 | Smooth Quake @ 50+ FPS; Win95 ready |
| 1996 | N64 | MIPS VR4300 | 64-bit | 125 | 4-8 MB | 320×240-640×480 | | Hard due to RSP/RDP synchronization |
| 1998 | Dreamcast | SH-4 @ 200 MHz | 32-bit | 360 | 16 MB | 640×480 | AICA | Very emulator-friendly architecture |
| 2000 | PS2 | MIPS R5900 | 128-bit SIMD | 6000 + 40 | 32 MB | 640×448 | SPU2 | Emotion Engine 6k MIPS; +40 MIPS for PS1 compat.; VUs dominate |
| 2001 | GameCube | Gekko @ 485 MHz | 32-bit | 1125 | 24 MB | 640×480 | Flipper DSP | Dolphin emulator gold standard; clean PowerPC arch |
| 2017 | Switch | ARM Cortex-A57 | 64-bit | 12,000 | 4 GB | 720p-1080p | | GPU & OS dominate emulation cost |

What happened?

Looking at the above chart, you will see “what happened”. In the early 90s, the 486dx-66 was the last great consumer CPU before the 3d revolution really took off. The death of the Amiga, the rise of the accelerated playstation and the absolute dominance of the Pentium changed how home computing worked. After that moment, graphics acceleration was a must and MIPS did not matter anymore. Thus, the PS-1, even though it is only 30 mips, had enough graphics acceleration to do things a similarly priced PC of the day could not.

The original Pentium (P5, 1993–1997) provided several key architectural accelerations that Quake's software renderer exploited via hand-tuned x86 assembly (primarily by Michael Abrash), delivering ~2–3x the performance of a 486DX4-100 in rasterization-heavy scenes.

The Days before GPU Acceleration

The Pentium was really the key that launched the GPU acceleration wars.

- Superscalar dual integer pipelines (U/V): Allowed two integer instructions per cycle, allowing quake to draw pixels in bursts
- Pipelined FPU: pipelined floating point ops let Quake fire FDIV for perspective-correct texturing in parallel with integer rasterization (r_alias.s), making slow DIV “free” (~1c effective).
- 64-bit FPU data path: Enabled fast 64-bit FP loads/stores and near-free FXCH (stack rotates, 0-1c), optimizing matrix/dot products for geometry and lighting
- Branch prediction: predicted branches (e.g., bottom-of-loop) increased throughput.

These Pentium-specific traits were exploited via Abrash's hand-tuned ASM (id386.asm) delivered a 3x speedup over 486DX4-100 and AMD/Cyrix 5x86-133 style CPUs, crushing the clones' weaker floating point pipelining and marginalizing them in gaming. Pentium began to dominate the 1996 PC market as Quake's “minimum viable” software 3D benchmark, shifting devs from CPU raster hacks to hardware offload. Next, GLQuake/Voodoo (1996) hit 60+ FPS by rasterizing on GPUs, birthing the 3D acceleration era.

Lesson learned, the biggest improvement besides raw MIPS is a good PPU/GPU.

Profiling Experiments

Here are the results of tight-loop experiments featuring benchmarks of one instruction. They are intended as relative results only. Taken on an i7-12700k.

The basic “unrolled LDA \$1000” example executes at 95 MIPS in the WASM version and at 675 MIPS in the C version. However, there's an issue if we go beyond this relative benchmark.

MIPS isn't useful

The following program illustrates a baseline:

```
.address $000100

    LCD $0FFFFFFF          ; choose a number to make the test take ~10
seconds

loop:
    DEC CD                 ; Decrement CD
    JNZ @loop              ; Jump to loop if CD != 0
```

HALT ; Halt when done

- WASM version 55 MIPS.
- C version was 560 MIPS.

But here's the problem with MIPS. LSTEP, a command that does DEC CD and JNZ in one step, performs at 360 MIPS (in the C version). This shows that MIPS is somewhat deceptive as a measurement. The LSTEP command is performing the work of both DEC and JNZ in less time than each; but since it is a relatively slow command in and of itself it lowers the MIPS of the system as a whole. In reality if it was counted as two instructions it would show over ~700 MIPS. If we use a dual stage LSTEP (on two 16 bit registers) it runs in 470 MIPS (940 mips equivalent).

Conclusion: CISC vs RISC

Time spent on the hot path is slow, while time spent in the hot path is fast. That is, just like the WASM version, the C version does best with CISC instructions. MIPS itself, is not as important as it seems. What matters is the quality of the instruction set; even at 1 MIPS, the WASM version can handle over 2,000 sprites per frame at 60fps, utterly destroying even the SNK Neo Geo (1990) in terms of arcade performance.

Using a RISC-like ISA is only a requirement if you are emulating a particular architecture. It is not a good idea for a fantasy computer in general. A fantasy computer does better with CISC instructions.

A final example; I had benchmarked kernal 0.7.2 at 750 MIPS, then I switched kernals to from 0.7.2 to 0.8.3. This had the effect of putting CASETAB into the hot path. This meant that instead of performing hundreds of JZ and CMP instructions per system loop, we performed one CASETAB. MIPS dropped to 500. But the system is running much faster! As it turns out, 10 seconds at 750 MIPS is slower than 5 seconds at 500 mips. That's the real takeaway; despite having a lower number of MIPS, the system runs measurably faster.

From:
<https://www.appledog.ca/wiki/> - **Appledog**

Permanent link:
https://www.appledog.ca/wiki/doku.php?id=sd:emulation_benchmarks&rev=1778889708

Last update: **2026/05/16 00:01**

