

# Apollo LEM computer

**Source:** <http://sci.tech-archive.net/Archive/sci.space.history/2004-07/1846.html>

---

*wboas\_at\_nyx.net*

**Date:** 07/10/04

Date: Fri, 9 Jul 2004 20:33:00 -0600

Folks,

I'm sure this file is known to NASA veterans, but as time passes, maybe posting it would be of interest to newcomers to USENET.

I found it about 3-4 years ago somewhere on the net.

Herewith, as I found it:

"Computers in the Apollo Program

The comp.realtime newsgroup has carried an interesting discussion of computers used on board the Apollo Command and Lunar modules. For those of you who haven't seen this, here's a repost of a fascinating story about these antique machines.

by Stanley R. Mohler, Jr. Dec '94

A few months ago I posted a message in sci.space.tech seeking info on the Lunar Module on-board computer and its software. I soon got some private e-mail from Allan Klumpp, the principal designer of the Apollo Lunar Module on-board descent software. He also wrote the steering system for the digital autopilot. He invited me to call him which I did. I thought I would summarize here my understanding of some of the fascinating info he gave me over a 1.25-hour period.

Allan was one of about 300 people who designed the LM's software over a 7 year period for 46 million 1967-era dollars. He did his work as a graduate student at the MIT Draper Lab during the Apollo years.

The LM and CM had identical computers on board, each the size of a shoe box. Each contained a total storage capacity of 36K of 14-bit words. This means total storage was roughly equal to the 64K bytes of a Commodore-64 computer. The LM's computer had a "memory cycle time" of 11.7 micro-seconds. However, virtually all CPU operations required at least 2 clock cycles making the effective memory cycle time 23.4 micro-seconds, i.e., it effectively ran at only about 43 kHz (0.043 MHz)! Note that the original IBM PC-XT ran at 4.77

## sci.space.history: Apollo LEM computer

MHz, and the latest PC's run at about 66 MHz. The fastest computers today run at about 300 MHz. The LM computer is probably comparable in speed to a pocket calculator. Numbers were represented using 14-bit words in double-precision (i.e., 28 bits). The 15th and 16th bit were for the sign of the number and for parity checking (i.e., to make sure the chips were all in sync with the clock pulses). Calculations were fixed-point (not floating-point).

The on-board program, named "LUMINARY", was stored in read-only core-rod memory which took months to manufacture (the program fills about 10 cm of print-out). Therefore the software had to be in final form months before launch. LUMINARY version 99 landed Apollo 11. Version 209 was the final version.

The computer also contained a small erasable area of about 2K 14-bit words to temporarily store variables in. The computer was built entirely out of integrated circuit NOR gates: one type of gate for high reliability.

Allan, his friend Don Eyles, and about 300 others wrote their programs in the first high-order computer language, called MAC (MIT Algebraic Compiler), then compiled it BY HAND into assembly language, which they typed onto punched cards (there were no terminals or text editors). Incidentally, the Shuttle's software is written in a language called HAL/S, named after Hal Lanning, the author of MAC. HAL/S is an improved version of MAC.

The LUMINARY program consisted of many subprograms which were priority driven, i.e., they took turns executing according to their priority. Each program would move data in and out of the very small erasable area of memory (2K in size). The biggest debugging challenge was to keep programs from erasing, or "overlying", another program's data at inappropriate times. If too many tasks were demanding the computer's time, it would simply delay or THROW AWAY what it had been working on, issue an alarm, and start working on the new item.

Such frightening alarms occurred during the Apollo 11 landing (first moon landing). If you listen to recordings of the landing, you will hear the Capcom say "1201 alarm" and "1202 alarm." The astronauts' checklist had erroneously called for the astronauts to turn on the rendezvous radar before initiation of the descent. Subsequently, the program that managed the radar began demanding too much of the computer's spare margin of time. The power supply for the radar was not properly synchronized with the LM's main power supply. Consequently, as the two power supplies went in and out of synchronization, the rendezvous radar generated many spurious input signals to the LM's computer. In responding to these signals, the computer delayed some of its guidance calculations and left others unfinished. This situation caused the computer to issue alarms during the landing. During a normal descent, the guidance program, which brought the LM to its target landing site using a minimum of fuel, would issue commands once every two seconds. Steering commands to the digital autopilot, which kept the LM stable, were issued every 10th of a second. Although the landing, which had an 11-minute guidance phase, was successful, a full minute's worth of guidance commands were never issued by the computer due to rendezvous radar!

For debugging, the programmers at MIT had an IBM 360 model 175 mainframe computer that acted as a simulator of the LM. Allan and his colleagues would

test their software in this simulator, which interfaced with their software just as the real LM, with its associated dynamics, would. The IBM 360 produced printed output as well as plots of the trajectories of the simulated landings.

In the real LM, the on-board computer had a digital display and a keyboard. During landing, the computer would display a number, updated periodically. The LM "Pilot", who was on the right and never touched the controls, would continuously read out updated values of this number. The Commander on the left, who was actually manipulating the controls, would find this number on a reticule painted on the window. The target landing spot, where the computer was trying to land, would be visible at that location out the window. The commander would "fly" the LM by redesignating to a new landing spot by clicking a hand controller. In this way, Neil Armstrong carefully steered the LM away from an unexpected crater full of Volkswagen-size boulders, setting the LM down with only 30 seconds of fuel left! One click of the hand controller would move the landing spot by a couple of degrees. Allan chose to program in 2 degrees left/right, and a half degree up and down (i.e., forward and backward). Later he changed it to 1 degree both ways, at the astronauts' request. The commander could also increase or decrease his descent rate by one foot/second by clicking a second hand controller.

LUMINARY was never completely bug free. Allan told me about a fascinating series of events that could have easily prevented the first moon landing and might have caused disaster. Allan was the principal designer of the LM's descent guidance program which steered the LM by gimbaling and throttling the descent engine. Whenever the computer commanded the engine to increase or decrease thrust, the engine (and LM) reacted after a short time lag. Allan's descent program needed a routine to accurately estimate the new thrust level, which could be accomplished by reading the "delta-V" (change in velocity) measured by the LM's accelerometers. He wrote a short routine that took into consideration, i.e., compensated for, the engine's lag time, which TRW's "interface control document", full of useful information for the programmers, said was 0.3 seconds. It took 0.3 seconds for the LM's descent engine to achieve whatever thrust level the computer might request. The final version of the thrust routine, which was put into the LM, was written by Allan's friend Don Eyles. Eyles was sufficiently enthusiastic about the programming challenge that he found a way of writing it which required compensating for only 0.2 of the 0.3 seconds. The IBM 360 simulator showed Eyles' program worked beautifully. His routine was aboard Apollos 11 and 12 which landed successfully. However, telemetry transmitted during the landings later showed something to be very wrong. The engines were surging up and down in thrust level, and were barely stable. A guy at Johnson Space Center called Allan and informed him that the LM's engine was not a 0.3-second-lag engine after all. It had been improved some time before Apollo 11's launch such as to lower the lag time to only 0.075 seconds. Correction of this item in the interface control document had simply been overlooked. Once this discrepancy was discovered, the IBM 360 simulator was reprogrammed to properly simulate the actual, faster engine. Running on the simulator, Don Eyle's thrust program, with the 0.2-second compensation, exhibited the surging that had occurred on the real flights. But here's the most interesting fact: the simulator also showed that had Allan Klumpp chose to "correct" Don Eyles' program by compensating for the

full 0.3 seconds that was printed in the document, the LM would have been unstable and Apollo 11 would never have been able to land. By pure luck, Don Eyles was creative enough to write the thrust routine in a way that kept the LM just inside the stability envelope and allowed successful landings!

Allan's descent program called "P64" periodically computed a polynomial function to describe the optimum descent trajectory. This polynomial would smoothly merge the LM's current position and velocity vectors into the target point position and velocity vector. The "target point" for P64 was just above the landing point (When the LM reached the target point with a small vertical descent rate, P64 would cease execution and the landing phase would be handled by a program called "P66"). The computer would then make the LM fly the trajectory, which would be recomputed every 2 seconds. An opportunity for disaster presented itself here. Many sci.space.tech readers may know enough mathematics to understand the undesirable "wiggles" that can be generated by high-order polynomial curve fits. Under conceivable circumstances, the polynomial function computed by P64 could droop down, go beneath the lunar surface, rise out of the surface, then descend to the target point! If such a trajectory were computed during a real landing, and the LM were allowed to follow it, the LM would crash. There was no logic coded in to detect this situation and prevent it. No programming solution was ever found. An example scenario where this disaster could have happened follows. If the LM was off course, away from the terrain model stored in the computer, and flying over a deep crater, the landing radar would fool the computer into thinking the LM was higher relative to the mean surface than it previously assumed. This could cause a newly computed polynomial trajectory to "droop" down sharply, unintentionally intersect the real lunar surface, then rise back out of the surface, inviting the LM to crash! Allan said this problem could conceivably be remedied by an astute astronaut retargeting the landing point beyond the fuel range (at least for a while!).

What would the computer have done if the LM's descent engine quit cold a mile above the moon? The computer would not have initiated any automatic solutions. Allan said the astronauts simply would have pressed an abort button, which would have jettisoned the descent stage and ignited the ascent engine for return to the CM.

I would like to thank Allan Klumpp for the time he spent explaining this stuff to me. It was absolutely fascinating to hear him talk. I hope sci.space.tech readers have enjoyed reading my description of Allan's comments."

-eof-

Bill  
wboas@nyx.net