5

From Sequencers to Computers: Exploring the Moon and the Inner Planets

One organization more than any other has dominated the exploration of deep space: the Jet Propulsion Laboratory (JPL) of the California Institute of Technology. JPL was responsible for the Ranger and Surveyor series of lunar exploration spacecraft, the Mariner and Viking Orbiter explorers of Mercury, Venus, and Mars, and the Voyager and Galileo probes of the outer planets. As a result, the evolution of on-board computers for deep space operations took place at JPL.

JPL's chief contribution to computing on unmanned spacecraft was in leading progress from hard-wired sequencers to programmable sequencers to digital computers. The Pioneer spacecraft developed mostly at NASA's Ames Research Center and the Lunar Orbiters used to map the moon in the 1960s did not carry on-board computers. Like their earth-orbiting cousins and the first JPL probes, they used sequencing devices to activate and command experiments. Later the Mariner spacecraft acquired more autonomy and flexibility by using machines that stored command sequences in changeable software. Finally, sophisticated spacecraft flew with special-purpose digital computers.

Unique in its relationship to NASA, JPL is *not* solely a government installation in the same way as, for example, the Johnson or Marshall Space Flight Centers. JPL's personnel receive their pay-checks from Cal Tech, yet almost every piece of equipment on the site has a NASA property tag, since, for over a quarter of a century, Cal Tech has administered contracts that have paid for all research and development of the many spacecraft originated at JPL.

Another way in which JPL is unique is its products. Whereas thousands of earth-orbiting satellites have been launched, less than a dozen each of Rangers, Surveyors, and Mariners were constructed, and just two Vikings and Voyagers and one Galileo were sent into space. Not only were few spacecraft built, but the interplanetary launches were separated by years and had to be on strict deadlines due to the realities of celestial mechanics. This created a completely different development environment than that at other NASA centers. The emphasis on basic research at JPL has perhaps been stronger than at any other NASA installation. This orientation and its application in spacecraft forms a special part of the story of JPL.

JPL's computer development activities were shaped by its organizational structures. When a project is started at the Laboratory, an office is established to house the project manager, key systems managers, and staff. Offices have come and gone with the projects themselves. The Ranger office, for example, has been closed for nearly 20 years, whereas the Voyager office is likely to be open for as long as that. Most personnel are housed in divisions and sections relating to specific discipline or system functions, as, in 1984, the "Technical Divisions" contained sections on "Guidance and Control" and "Spacecraft Data Systems." When a project office needs a component or service, it "subcontracts" it to the appropriate technical sections. For instance,

Spacecraft Data Systems supplies on-board computers, whereas the Navigation Systems Section does the trajectory calculations needed for a specific mission. In this way, specialists can be kept busy on a series of projects over a period of years without depending on a specific project for their jobs. Competition between sections to develop related components can also exist, as on the Voyager project, when the attitude control staff wanted to make their own computer for their system while the data systems people claimed sole domain over computer development. Within this setting, JPL has produced high quality on-board computers that have demonstrated outstanding reliability*.

FIXED SEQUENCERS: "COMPUTERS" ON RANGER, SURVEYOR AND THE EARLY MARINERS

Whether the final mission destination is as close as the moon or as far as Neptune, probe spaceflights consist of the same milestones and activities: launch, mid-course maneuver, cruise, and encounter. Spacecraft are launched in a stowed position dictated by the geometry of the booster vehicle. Most space probes look like multiarmed Hindu gods in flight due to the need to expose solar panels, point antennas, and deploy imaging equipment, but they must be folded to fit into the nose fairing of a rocket. During the launch period the spacecraft is injected into its transfer orbit to intercept the target, deploys its various appendages into their proper positions, and orients itself. A decision was made early at JPL to build spacecraft that would be stabilized in three axes during flight[1]. Spacecraft would be oriented by using the sun, earth, and/or a star as a reference. If kept from tumbling they would always be pointed in a specific direction. A key advantage of this plan is that a directional antenna could be used for earth-space communications, reducing power requirements. Imaging equipment could also be more stable than on a spin-stabilized spacecraft such as a Pioneer. A disadvantage of three-axis stabilization is that a fairly sophisticated attitude control system must be carried, including a sensor system to find the sun and a guide star. Part of the launch phase, then, is spent scanning the sky for Campus, Vega, or whatever star has been chosen for aligning the spacecraft.

The mid-course maneuver phase often comes only a day or two after initial transfer orbit insertion in order to correct relatively large injection errors. Consisting of a timed burn of the spacecraft's

*JPL's roots and its role in NASA receive excellent treatment in Clayton Koppes' *The Jet Propulsion Lab and the American Space Program*, Yale University Press, 1982.

propulsion system in each of three axes, it serves a number of purposes. Early launches could not depend upon the launch vehicle to establish an adequate flight path. Later, as booster guidance improved, probes were purposely aimed to miss the target so as to avoid contaminating planetary atmospheres with earthly bacteria hitching a ride on a spacecraft if the spacecraft ceased to function during launch and could not change its path to miss the planet. Therefore, the mid-course burn took place to correct the path of a "live" spacecraft. On long-duration missions with several targets, such as the Voyager probe to Jupiter, Saturn, Uranus, and Neptune, this maneuver might be repeated before and after each encounter. Engine firings are made before encounter to improve the accuracy of the trajectory to achieve a better gravity assist from the target planet to the new trajectory and reduce the size of the post encounter maneuvers.

Less is done on the spacecraft during the cruise period than in any other mission phase. However, recent larger and more complicated spacecraft have particle and fields experiments that run constantly and engineering calibrations that need periodic attention. If the spacecraft attitude is disturbed, reorientation may be necessary. This period of relative quiet ends when the encounter sequences begin as the spacecraft nears its target. Instruments must be turned on, calibrated and aimed. Imaging instrument pointing must be programmed and controlled. Data must be recorded and transmitted to earth. Of course, these activities are repeated during multiple encounter missions.

Initiating the functions done in each phase requires on-board control. This was unnecessary for Ranger missions to the moon, which were simple impact flights with televised imaging during the last minutes. Because maximum speed-of-light delay in radio signals to the moon is less than a second, near-real-time commanding could be done. Ground commands could fire engines, point the spacecraft, and turn on cameras. Ranger flights used a voice/manual commanding system for this. Desired instructions were developed and formatted at JPL and then delivered by telephone to the Deep Space Network station currently in contact with the spacecraft. An operator would thumb-wheel the octal codes into a panel called the "Read-Write-Verify Console," sending them to the spacecraft after verification[2]. Such care was not always enough. On Ranger III, a guidance error caused the spacecraft to miss the moon by 23,000 miles. Although JPL flight controllers were able to get images during the flyby, a documentation discrepancy between the command set developed during the ground testing of the spacecraft and the flight set caused Ranger to point the wrong way, returning images of open space[3].

Ranger carried a "Central Computer and Sequencer" to back up the direct command system. Activated before lift-off, it counted the hours, minutes, and seconds until a specified mission event was to occur and then executed a set of commands that performed the required

functions. If the uplink radio channel failed, the mission would proceed according to a prepared plan. This assumed optimum performance, turning on the cameras regardless of where the spacecraft might be actually pointing. Still, it provided a bit of insurance for the mission.

At the same time that the Rangers were being built, JPL designed and flew the first Mariners. Mariner's initial mission was a Venus flyby launched in 1962. In the case of this spacecraft and its later brethren, the Central Computer and Sequencer was the prime source of commands, at least for cruise and encounter portions of the mission[4]. The time delay for commands to travel to Venus and Mars defeats real-time control from the ground. For Mariner II, at launch time minus 15 minutes, the clock was set so that the encounter sequence would begin at 12 hours from the closest approach to Venus. The sequencer's clock, a very accurate oscillator similar to computer clocks today, started at launch time minus 3 minutes[5]. Direct commanding capability was maintained. When the star tracker got confused and locked onto the wrong target, ground controllers could reinitiate a search[6]. Direct command could also be used for midcourse maneuvers. As a complement to direct command, "quantitative" commands could be sent to the sequencer for later use[7]. For instance, times such as "51 seconds of minus roll" and "795 seconds of minus pitch" or burn times could be inserted into the memory for later execution[8]. Mariners could abandon direct command and go to automatic command if a radio failure was detected. On the Mariner Mars 1964 spacecraft the sequencer contained a cyclic command that checked for such a failure at 66 2/3 hour intervals, effecting an auto switch-over[9].

The Mariner II spacecraft to Venus (1962), Mariner IV to Mars (1964), and Mariner V to Venus (1967) carried the same Central Computer and Sequencer. Just one flew on each mission, due to space and weight restrictions, even though the machine weighed in at 11.5 pounds[10]. However, with the direct command capability intact, each had essentially the same level of redundancy as the Gemini and Apollo spacecraft, with their single-processor on-board computer systems and ground control computers. Plans for Mariner Mars 1969 called for a larger spacecraft and a more ambitious mission: two picture-taking flybys of different portions of the "red planet". JPL's Neil H. Herman, who had headed development of the Sequencer, saw an opportunity to improve the device for the upcoming flights[11]. One aim was to give the new spacecraft more flexibility. If the first flyby turned up something special, it would be very useful if the second spacecraft could be reprogrammed in flight to take advantage of lessons learned on the initial pass[12]. This actually happened during the missions when reprogramming was accomplished for Mariner VII's August 5,

1969 flyby in response to Mariner VI's July 31 passage[13]. Another reason for more on board autonomy is that command sessions for the Mariners lasted as long as 8 hours! Mariner's command rate was 1 bit per second, so long sequences were expensive both in personnel time and Deep Space Network time[14]. The availability of more space and weight plus the desire for flexibility and greater autonomy caused JPL to change the Sequencer to make it more of a computer and less of what it really was, a fixed-program counter.

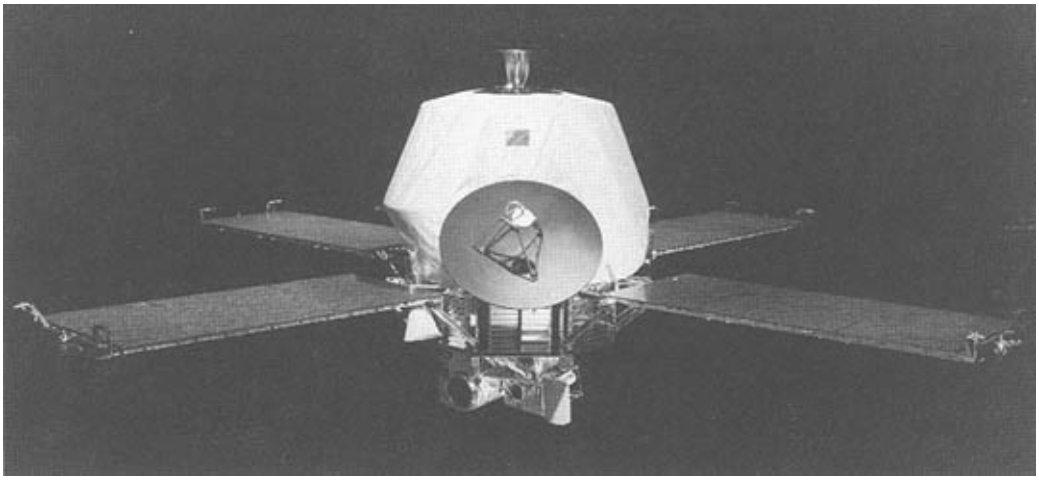

Figure 5–1. Mariner Mars 1971 carried a programmable sequencer with an expanded memory. (JPL photo P12035)

PROGRAMMABLE SEQUENCERS: MARINERS TO MARS, VENUS AND MERCURY

Mariner Mars 1969 carried a 26-pound, programmable Central Computer and Sequencer designed by Herman and his team at JPL and built by Motorola[15]. The machine originated in studies done in 1964–1965 for a Mars orbiter and lander called "Voyager" and Mariner Mars 1966, neither of which flew[16]. The major difference between the fixed sequencers and the programmable sequencer is that it had a memory of 128 words that could be altered in flight. Although this device had far greater flexibility and capabilities than the fixed sequencers, one of the older sequencers traveled on the spacecraft as a backup. During critical maneuvers, the two sequencers would run in parallel, a disagreement causing an abort of the maneuver. The Sequencer commanded all spacecraft systems, including the Attitude and Pointing System and Flight Data System, each of which evolved to include their own computers by the time JPL designed the outer planets Voyager in the 1970s.

Original requirements for Mariner Mars 1969 called for 20 words of memory, making the 128-word version more than enough. Yet the memory was quickly exceeded, necessitating the use of "creative" programming techniques for the duration of the mission. Fortunately, the Sequencer was reprogrammable in flight. Memory locations used for terminated mission phases could be given to tasks scheduled for later. Edward Greenberg of JPL, who did most of the programming, replaced the launch and mid-course burn routines with new code after they had been executed[17].

Despite the autonomy and flexibility gained by using the programmable Central Computer and Sequencer, the two Mariner Mars 1969 missions were the "most commanded" to that date. Mariner VI received 946 radio commands, Mariner VII got 957; either number exceeded the total number of commands sent to all the three previous successful missions combined[18]. One of the reasons for this was the memory restrictions; another could be that the engineers on the project downplayed the full capabilities of the Sequencer, not realizing what was possible[19]. However, the full capabilities of the device were more than exercised on the last Mariner missions.

Box 5-1: Programmable Central Computer and Sequencer Architecture and Software

The new Central Computer and Sequencer had no accumulator or central registers common to standard computers. Each memory location could be used as a register, and all operations began at a location, acted on the contents of another location, and ended in a third memory location[20]. Memory consisted of 22-bit words stored in magnetic core, with destructive serial readout[21]. Three types of words could be stored. An instruction word used the first 4 bits for one of the 16 operation codes, the next 9 for the address of the memory location to be acted on, and the last 9 for the address of where to go afterward[22]. Instruction DHJ meant "decrement hours and jump," so the computer would subtract one from the time portion of the event word stored in the location specified by the first address in the instruction and then jump to the location specified by the second address. An event word contained a 13-bit time, scaled to hours, minutes, or seconds, and a 9-bit address of where to go to start the event being timed when the time part zeroed[23]. For instance, if the event word was timing the mid-course correction burn, when the time portion reached zero, a branch would occur to the specified address, the first address of the mid-course maneuver subroutine. The last type of word was a data word, containing 22 bits of data.

An instruction set of 16 operation codes contained mostly counting-type instructions: five scaled decrementing instructions (countdown hours, minutes, seconds, variables), and an incrementing instruction (count and jump). There was an ADD and a SUBtract, each requiring 27 milliseconds of machine time, by far the slowest instructions. Programmers used those very rarely, as the other instructions were better suited to sequencing. Subtraction was in one's complement form.

The sequencer executed the software by making a scan of the first seven instructions each hour[24]. Those instructions constituted the entire executive! They contained sufficient decrement and branching instructions to check if anything needed to be done during that hour of flight. As an example, the exec might contain a counter that kept track of the time to an imaging session. The resulting routine might look like this:[25]

1. Count 123 hours from start.
2. Count 45 minutes.
3. Activate camera and start frame count.
4. At 29th frame, start sending images.
5. At 32nd frame, rotate filter wheel to blue.
6. At 93rd frame, stop scan and stow platform.
7. Resume cruise mode counting.

After resuming cruise mode, the spacecraft clock would activate a scan at hour intervals again. Mission control could interrupt a scan, or a quiet time, and cause a jump to a specified subroutine[26]. (The entire Mariner Mars 1969 flight program is reproduced in Appendix IV.)

A memory location could be changed by issuing two consecutive commands from earth stations. JPL called these commands CC-1 and CC-2. CC-1 sent the address and the least significant 7 bits of the new word, and it caused an interrupt in the receiving Sequencer. CC-2 relayed the most significant 15 bits and released the scan inhibit[27]. A related command, CC-3, caused the Central Computer and Sequencer to read out the contents of a specified memory location, 1 bit per second[28]. Input was even slower, requiring an average of 2 minutes per word, compared with a ground-loading time of all 128 words in less than a minute[29].

EXPANDED MEMORY AND EXPANDED FUNCTIONS

The new sequencer had a 9-bit address field, providing a 512 address limit. Expanding the memory to 512 words did not require a change in the logic. So JPL added the extra memory for the Mariner Mars 1971

orbiter missions. Still, the old fixed sequencer remained in charge of the Mars orbit insertion burn. After the spacecraft established orbits, however, the ground control center used the new sequencer to control the imaging of Mars and its moons. The expanded memory proved sufficient. Preflight estimates for Mariner VIII specified 150 words of memory and 225 words for Mariner IX, yet both grew to over 400 words in flight[30].

The mission that used the sequencer to its limits was Mariner Venus Mercury 1973, or Mariner X. Mission profile called for the spacecraft to turn its imaging equipment on the earth as it flew toward deep space, do some studies of the moon in flyby, and then research in the area of Venus during a gravity assist maneuver that would send it toward Mercury, where JPL planned three separate encounters with the innermost planet.

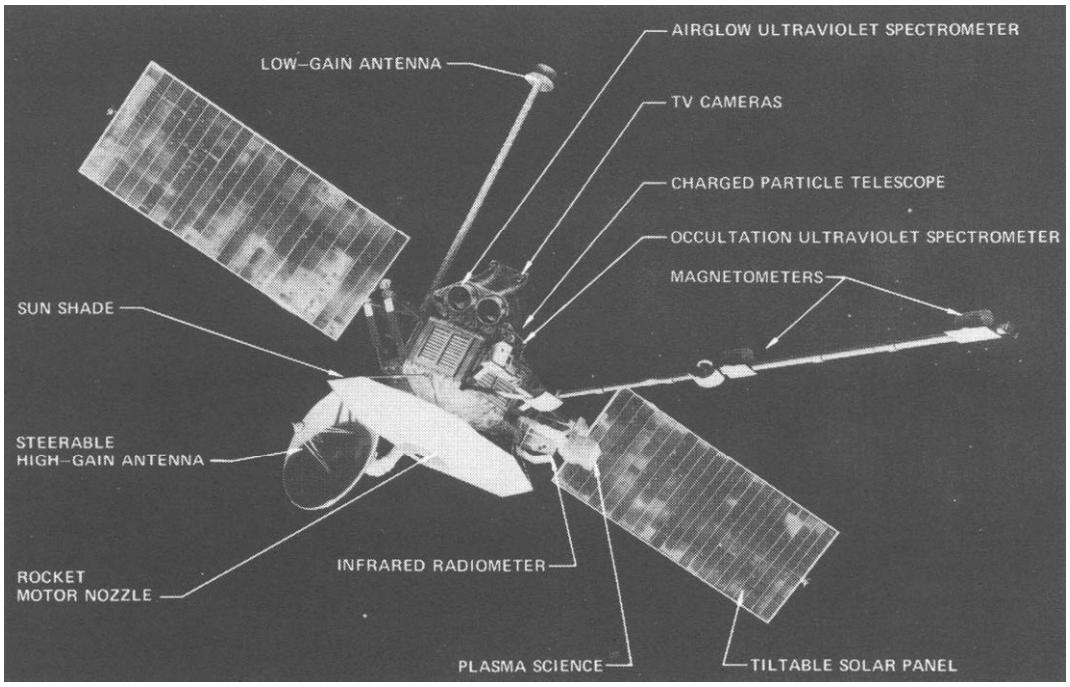

Figure 5–2. Mariner Venus/Mercury 1973 made the most use of the programmable sequencers. (JPL photo 251-135AC)

Due to the more complex mission requirements, the design team wanted a bigger and better sequencer, but cost constraints killed any chance of building a new machine[31]. Adrian Hooke of JPL, one of the project's managers, decided to use planned memory updates at regular intervals. He also instituted a "suspenders and belt" approach to reliability. The sequencer would not only carry a detailed program for the next mission phase but also a constantly updated bare minimum program to complete the mission if the spacecraft lost contact with the ground. If a command was not received for a certain time, then the sequencer would follow whatever commands were in the backup program. Thus, software moved ahead in leap frog fashion. During the earth-moon phase the Venus backup was loaded, during the Venus encounter the backup Mercury encounter sequence was on board, and so on[32]. Software development was assigned to three programmers. Ronald Spriestersbach of JPL wrote the near-earth and post-Mercury sequences, George Elliot of the Boeing Company did the Venus encounter, and Larry Koga of JPL wrote all three Mercury encounters[33].

During the 1969 missions, most changes and subroutines were hand-coded and used once. By 1971, the COMGEN ground computer program that produced memory loads for the Sequencer could develop blocks of commands that functioned much like subroutines in a standard computer program or macros in an assembly language program[34]. In 1973, COMGEN resided in an IBM 360/75 computer that generated the commands and sent them via the NASA communications net to the appropriate Deep Space Network station for transmission. By this time, each station had a command computer, thus ending the voice/manual era[35]. Another improvement to the Sequencer was that engineers could do memory checks by comparing a sumword stored in location 512 to the result of summing the first 511 locations. If a miscompare occurred, *then* a location-by-location check for error could be made[36].

The improvements both in the Sequencer and in programming and ground control techniques were not enough to ensure its use beyond the Mariner series of spacecraft. In spite of the success of the long and complicated mission of Mariner X, JPL's Hooke complained that memory limits were too costly due to excessive need for optimization and constant relocation of subroutines[37]. Besides, the sequencers, regardless of their full name, were *not* computers. Spacecraft needed to do on-board computations, to have more room for software (and, thus, increased flexibility), and to use the central computer for other functions such as spacecraft health and safety monitoring done on other manned and unmanned spacecraft. Some missions intrinsically needed computers, as, for example, the Viking Mars orbiters and landers and the Voyager outer planet probes. The computer eventually designed, built, and used for the Viking Orbiter had its roots in the programmable sequencer, but it also owed some concepts, at least

in comparison, to a computer built in the research side of JPL and aimed at the long-duration, complex missions of the future. The story of that computer research project adds a necessary perspective for understanding the direction JPL's on-board computer development took in the 1970s.

THE STAR COMPUTER

Researching the Reliability Problem

In 1961 a Lithuanian-born computer scientist named Algirdas Avizienis, employed at UCLA, began research on a highly fault-tolerant computer system for use on long-duration space missions. The nonprogrammable version of the Central Computer and Sequencer would soon make its first flight on Mariner Venus 1962. Even at that early date, JPL expected to use computers on board the "Grand Tour" spacecraft planned for the 1970s. A favorable alignment of the outer planets would make possible a mission that could fly by Jupiter, Saturn, Uranus, and Neptune, thus having encounters with all the gas giants in one sweep. Such a mission would have to last for years, with the spacecraft operating autonomously for long periods of time. Inconvenient speed-of-light communications delays in the exploration of the inner planets would become crippling in an outer planets mission, requiring a spacecraft to carry its own "brain," because the earthbound brains of its makers would be hours away in an emergency.

Avizienis' chief interest was in computer reliability. Computer failures occurred much more frequently then than in today's world of ICs. A computer entrusted with the successful completion of a deep space mission could not afford to fail before or during its long awaited encounter, so JPL and Avizienis' interests came together at just the right time. During the period from 1961 to 1965, the Laboratory sponsored his search for a more fault-tolerant computer. In 1965 the reliability scheme was settled and construction of a prototype began. The breadboard version first ran a program in March of 1969, after a 2-year effort at software development[38]. Avizienis named the computer STAR, for self testing and repair, and the name gives a clue to the architecture. JPL's Flight Computers and Sequencers Section of the Guidance and Navigation Division paid for the work. Avizienis was responsible for the concept; David A. Rennels, later a colleague at UCLA, for the hardware; John A. Rohr, for the software. F.P. Mathur did the reliability calculations, and the MIT Instrumentation Laboratory developed the read-only memory, which was basically a core rope type of memory[39].

Avizienis used selective redundancy to achieve reliability. On the Space Shuttle, the on-board computers are complete redundant versions of each other and are considered multiple computers. In the STAR, the computer is considered a single entity with its separate components replicated. Thus, each subsystem of STAR had several duplicate versions of itself in the computer as spares. The key advantage is that the spares were unpowered as long as the primary component ran successfully. Only when there was a failure would the spare come to life, and then power to the failed component would be cut off. Thus, the total power consumption of the STAR equaled, but did not exceed, that of a similar computer without the spares, making it attractive to power-conscious spacecraft designers[40]. In the 1960s, all spacecraft computers were simplex systems. The only ultrareliable system was the Launch Vehicle Digital Computer used on the Saturn IB and Saturn V boosters. Its reliability was achieved by using triple modular redundant (TMR) circuits such as those in the Common Section of the Skylab computer system. Avizienis evaluated TMR circuitry and found that the number of independent failures a TMR system could tolerate before failing was much smaller than a component-redundant computer such as STAR could tolerate[41]. Also, reliability theoretically increased through dormancy[42]. Mean-time-between-failure (MTBF) figures for a component begin when the component is turned on; thus, a subsystem with a MTBF of 1,000 hours, backed up with two identical spares, yields a MTBF of 3,000 hours. That was the theory behind STAR.

Avizienis reasoned that failures were either caused by transient conditions or permanent component failures. In order to check for transient faults, STAR would repeat the program segment in which a fault was first detected. If the fault repeated itself, the affected component would be turned off and its spare activated, with the program segment repeated again. All fault detection was by hardware techniques, with error-correcting codes included in the software[43]. Potentially, STAR could be an "automatic repairman" for the entire spacecraft, if other spacecraft systems used the same concepts[44].

Applications for STAR

In 1969, JPL began designing a Thermoelectric Outer Planet Spacecraft, or TOPS. In previous inner planet probes, the flight paths were close enough to the sun to enable the spacecraft to use solar cells for power generation. Outer planet missions ranged so far from the sun that solar cells would be inadequate. TOPS would carry radioisotope thermoelectric generators to provide electrical power.

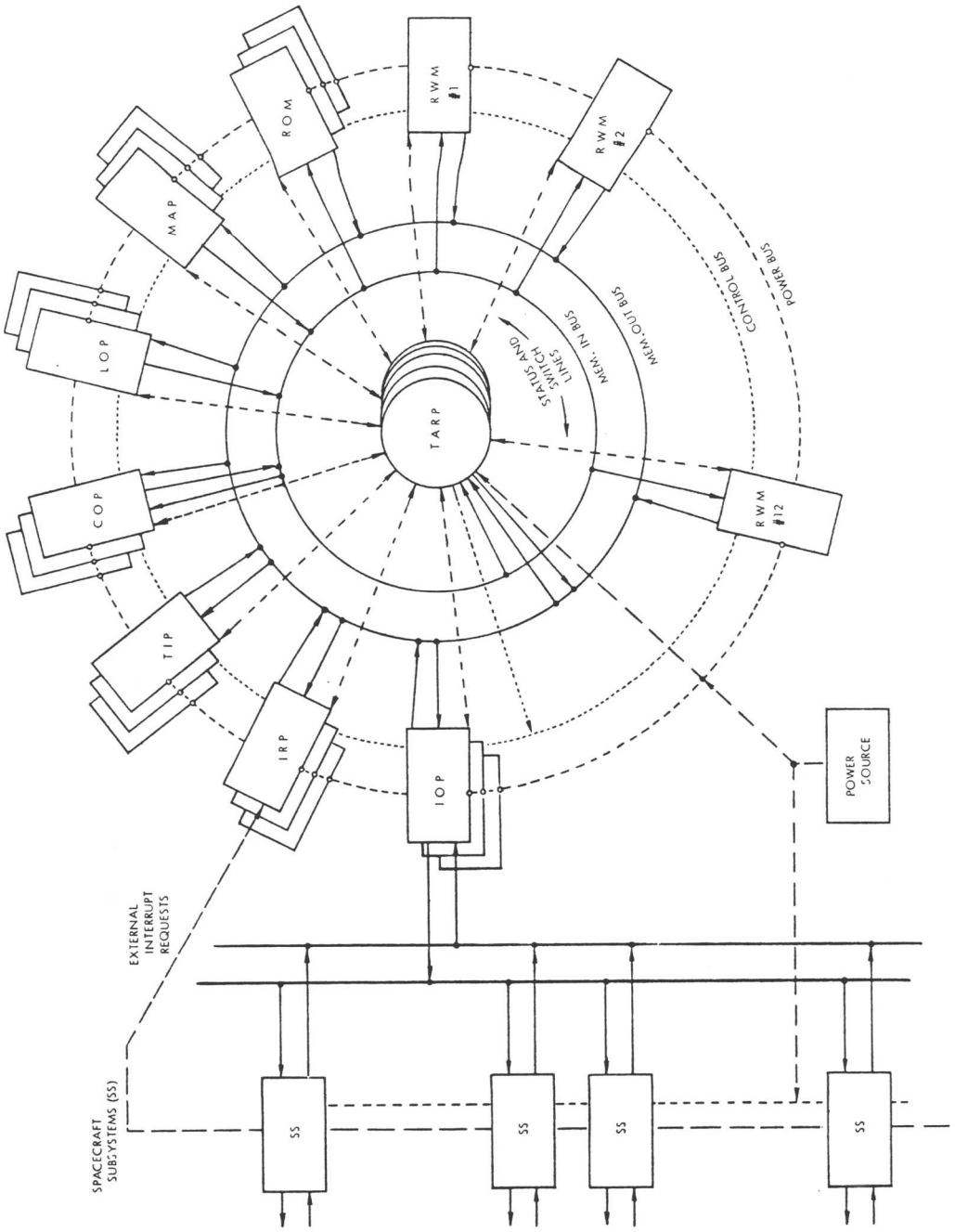

Figure 5–3. The STAR computer configuration. (From Avizienis, "Design Methods for Fault-Tolerant Navigation Computers," JPL TR-32-1409)

Box 5-2: STAR Architecture and Software

STAR was a fixed-point machine with a 32-bit word. Using separated components for redundancy meant that they had to be connected on a bus, which had 4-bit bytes as the basic transfer block[45]. There were 16K words of read-only memory, which Avizienis said consisted of a "braid" of transformers and wires[46]. Since MIT built the device, the description almost certainly indicates that it was a core rope similar to that used in the Apollo Guidance Computer (AGC). The basic version used two copies of 4K of random-access memory, with up to 12 units attachable. Avizienis forsaw that the memory would have to be reprogrammed in flight on a mission like Grand Tour, so provision was made for that function[47].

Use of a large word size was not to increase arithmetic power as much as to provide space for error-checking codes. A STAR address consisted of a 16-bit field for the address and a 4-bit check field. The address would be multiplied by 15 (yielding 20 bits), and then stored or transmitted along the bus during an operation. At the receiving end, the address would be evaluated according to the following equation:

$$C(a) = 15 - 15|a$$

where $15|a$ is the modulo residue of a. Numeric data were handled similarly; the 28-bit operands multiplied by 15 to get a 32-bit word. If the result of the check operation was zero residue, the data or address was correct. If not, STAR issued a fault signal[48].

STAR had three control signals. One was the common 1-megahertz CLOCK signal. RESET indicated a return to a standard initial state. SYNC signaled the beginning of a new 10-step instruction cycle[49]. If a fault was detected, the computer would return to the last SYNC point and begin executing instructions from there. If the 10 instructions after the SYNC were executed successfully, STAR sent a new SYNC signal.

STAR's read/write memory units were different in that they would recognize either their hard-wired name or an assigned name[50]. In this way, if a memory unit and its backup copy failed, another memory unit could be assigned its name, loaded with the appropriate data, and then act like the original memory unit, thus avoiding the necessity of changing all the addresses in the software. When an instruction appeared on the memory in (MI) bus, the memory unit that had that address put its contents on the memory out (MO) bus, and the Arithmetic Processor or other component loaded it in for processing[51].

The heart of the STAR was the Testing and Repair Processor, or TARP. Whereas the other components of STAR had either one or two unpowered spares, the TARP had three active versions and two inactive spares. Functions of the TARP were to maintain the rollback points to which the software returned after a failure detection, to diagnose failures, and to check itself. Each time an error check was made, TARP's three units would vote. If all three or two of three indicated a failure, then the TARP issued an unconditional transfer to the rollback point. In the case of a 2-to-1 vote, the dissenting unit was considered failed, and was shut off as a spare was activated[52]. Another TARP disagreement caused the last spare to be activated. On the third TARP failure, one of the previously shut down units would be reactivated, so that there were always three TARPs in action at any given time. Avizienis thought that since most failures would be transients, it would be safe to reactivate a unit. After all, if it disagreed again, it would be shutdown.

John A. Rohr's software group did not begin work until 1967. An assembler, loader, and simulator were developed on a UNIVAC 1108 mainframe computer owned by JPL[53]. Software was all done in assembler, with a rich set of 180 single address instructions[54]. The assembler did allow some types of higher level statements, mostly for arithmetic. For instance, COMP Y=Y + 1 was directly compiled into the several machine instructions necessary for execution[55]. In this way, some of the tedium associated with assembly language programming was avoided. A floating-point subroutine to extend the calculating power of the machine was planned, but there is no evidence it was ever implemented[56]. It would have had to have been done in software. The STARlet, a limited breadboard version, ran its first program on March 24, 1969[57]. The full system, save the timing processor, was on the breadboard by April 1970[58].

STAR was considered as the on-board computer for TOPS[59]. A control computer subsystem for the TOPS would use STAR technology, the full 32-bit word, but just 4K of read-only memory and 8K of the read/write memory[60]. The chief physical obstacle to using STAR on a spacecraft was size. The breadboard version filled 100 cubic feet. Avizienis wanted to reduce it to 2 cubic feet and 50 watts[61]. By 1971, the requirements reduced to 1 cubic foot, 40 pounds weight, and 40 watts power[62]. Even though progress was made in this area, STAR never flew on a spacecraft. Components built to STAR specifications found their way into the NASA Standard Spacecraft Computer 1 (NSSC-1), used in earth orbital operations, but the concept of selective redundancy was not incorporated into flight computers to the extent desired by Avizienis.

STAR did not find its way to the outer planets for two reasons. One was budget cuts[63]. Even though the Voyagers were launched

in the late 1970s, the original TOPS program and the Grand Tour were canceled due to budget constraints. The fact that Voyager 2 is essentially executing the Grand Tour is a bonus. On-board computers used on Voyager developed from a different line. So, even though Avizienis designed a Super-STAR with a microprogrammable processor using large-scale integration technology, which seemed certain to fulfill the requirements of size, power, and weight, he never sold it to JPL[64]. A second reason STAR never flew was that engineers were concerned that the STAR's TARP and its failure switches were a weak point. The concept of a TARP, as with TMR, is always limited by the question of "who tests the tester?"[65]. The actual switches entrusted with powering down a failed component and charging up another are the weakest link in the system. At one point, JPL subcontracted to the Stanford Research Institute for work on a magnetic switch, but apparently the results were not satisfying[66].

The STAR research program was not a waste even though the computer itself did not fly. It contributed to the development of new, reliable electronic components, such as those used on NSSC-1. It also provided a contrast to the development track being taken on the Mariner and Viking Orbiter spacecraft. One engineer involved in Viking Orbiter computer development said that STAR-type hardware was considered but deemed too complex. He thought that a two machine system turning in parallel would be simpler and as reliable for a Mars orbiter/lander[67]. Even though the technology of computers was not ready for STAR, it remains an innovative design and one of the few computer research projects funded by NASA. The principles developed remained valid for possible future applications that JPL was about to begin.

By far the most direct and far-reaching contribution of the STAR program to the future of JPL projects was John Rohr's work on the assembler/linker/loader for the software. It was the basis for the command sequence translators used through the present. Though extensively reworked and redesigned, the fundamental concepts were established by Rohr during the STAR development[68].

VIKING COMPUTER SYSTEMS

Viking missions to Mars were among the most complex ever executed by an unmanned spacecraft. Two probes were launched in 1975, with landings planned for the Bicentennial Summer of 1976. The project was controlled by the NASA Langley Space Flight Center, making it unique among deep space projects. Major work began in 1970, with a planned 1973 landing put off until 1976 because of budget cuts[69].

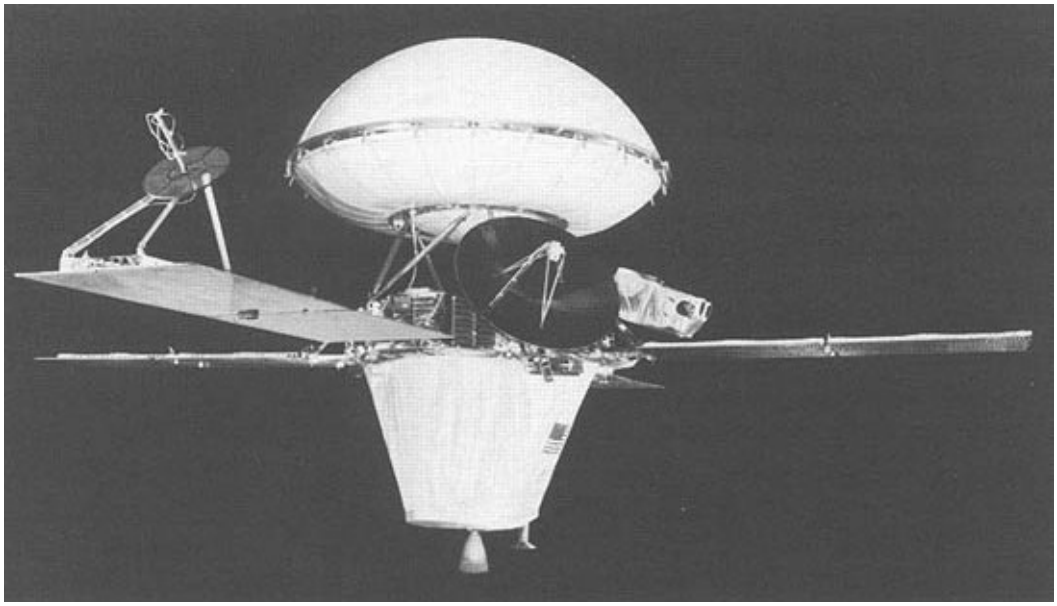

Figure 5–4. The Viking Orbiter and Lander each carried dual redundant computer systems. The Lander is in the elliptical shroud. (JPL photo 293-9157)

The Viking mission profile was a combined orbiter/lander. NASA had successfully orbited Mars with two Mariners in 1971. The Viking Orbiters were to conduct much the same science and imaging experiments as their smaller predecessors. But the Lander was a dramatic addition: it would be the first spacecraft to land on a planet that had a chance of harboring life as we could understand it. JPL got the contract to develop the Orbiter as a result of its Mariner experience. Because JPL maintained the Deep Space Network and an existing control center, it also got the mission support contract. The surprise for JPL was that the Martin Marietta Corporation's Denver division received the contract for the Lander. JPL had built the only U.S. unmanned landers, the Surveyor moon probes. Despite that experience and the difficulty of coordinating work on the Orbiter, Lander, and mission support in sites ranging from California to Denver to Virginia, Martin Marietta was chosen**.

Both the Orbiter and Lander carried dual redundant computer systems. JPL had evolved past the programmable sequencer stage

**Edward and Linda Ezell make the point in their book *On Mars* (NASA SP-4212) that part of the reasoning for choosing Martin Marietta was that the project management team at Langley felt that JPL would be overtaxed handling responsibilities for two spacecraft. Also, the difficulty of integrating the Lander components was greater, and a large aerospace contractor such as Martin had more extensive experience with such activity.

and flew a device called the Command Computer Subsystem (CCS) on the Orbiter. The Lander carried the Guidance, Control, and Sequencing Computer (GCSC). On both systems JPL and Martin demonstrated exceptional competence in software engineering in the areas of documentation and configuration control. JPL was essentially programming its first flight computer. The standards and practices used during the Viking project surpassed all but the Shuttle on-board software in quality. Somehow JPL avoided the trial and error learning process Johnson Space Center went through with the Gemini and Apollo flight software. On the other hand, Martin Marietta typically used good software development practice. Along with other defense contractors such as Boeing Military Airplane Company and TRW Corporation, it was among the leading producers of software in the world. Whereas commercial computer companies such as IBM, Honeywell and Digital Equipment have generally written systems software for their own products, large-scale applications software has been the domain of vendors supplying the military services with command and control systems and embedded software in weapons. Martin Marietta is such a vendor and subscribed to military contract specifications that required the use of strict software engineering principles. That experience carried over to Viking, prompting an innovative method of developing the flight program that holds promise for future space systems.

Viking Orbiter CCS

The Viking CCS made it possible to increase the results of the Orbiter mission many times over the Mariners of 1971. According to one designer, the 512-word Central Computer and Sequencer would have returned less than a hundredth of the data received from the Orbiters[70]. JPL considered several designs for the Viking computer, finally settling on the eventual Command Computer because of its simplicity. It had the least number of parts and was similar to prior systems in concept[71].

Viking's CCS was the first JPL command device to be fully redundant. Mariner missions that retained the original hard-wired sequencer to back up the programmable sequencer were redundant in the same way the Apollo lunar excursion module (LEM) had computer redundancy: two different systems could accomplish some, but not all, of the other's functions. The dual redundancy of the Viking subsystem was more like Skylab's computer system, with two power supplies, two processors, two output units, two discrete command buffers, and two coded command buffers. Interrupts and level inputs to the system were split and thus delivered identically to both processors. Processors and

output units were cross strapped so that in case of failures they could be reassigned. The hardware requirements document generated by JPL called this type of redundancy "single fault tolerance," in that each component had a backup, making possible extensive redistribution of functions[72]. In practice, the two sets of computers were useful because, at times, there was too much for one computer to do[73]. Although the designated secondary processor and memory were rarely on line, certain operating modes called for dual processing. Requirements specified three operating modes: individual, where each computer could be working on different events; parallel, where each computer worked on the same event; and tandem, where each computer worked on the same event and the output units were voted in a manner similar to that used on the Mariners when the two sequencers were in action together[74].

In general, the design of the processor was exceedingly simple, yet fairly powerful, as indicated by the use of direct addressing, a minimal set of registers, and a reasonably rich set of 64 instructions. The key is that the design placed relatively light demands on spacecraft resources while replacing both the programmable sequencer and the command decoder used in the Mariners. The fact that the processor was later adopted by the Voyager project as its Command Computer and modified for use as the attitude control computer is not only a statement of JPL's frugality but also a testament to the versatility of the design.

Software Development Practices for the Viking Command Computer Subsystem

By the time Viking was under development, JPL had over a decade of ground software experience, with resulting institutional development standards. Most space-related software done at JPL in the 1960s was for the Deep Space Network and the large computers in the mission support area. Viking was the first flight software project, so it was remarkable that effective software standards were in effect from the beginning.

JPL's project organization assigned each subsystem a Cognizant Engineer responsible for the overall development of the component. For the CCS, Wayne H. Kohl was the "Cog Engineer." Samuel G. Deese and T. K. Sorenson also signed the hardware and software requirements documents and were heavily involved in the development of the computer. Significantly, JPL also assigned a Cognizant Software Engineer, R.A. Proud. JPL's project management apparently believed that software could be engineered, like hardware. Both hardware and software had requirements documents that set forth the functional

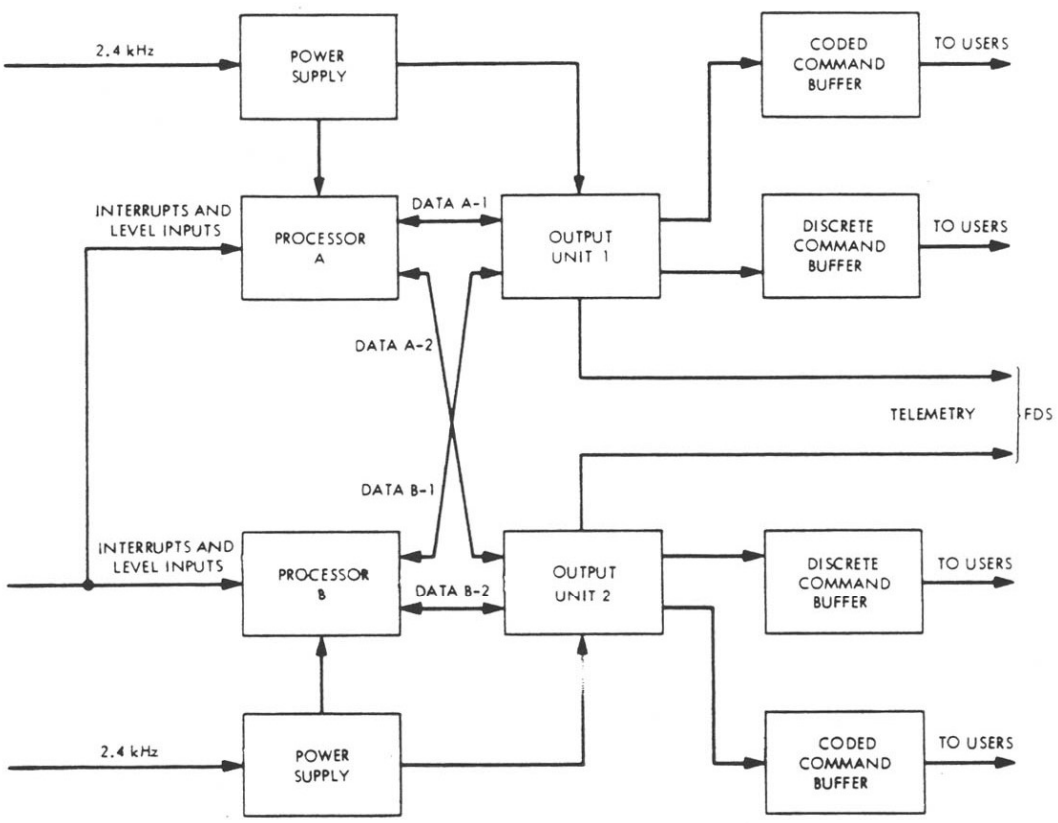

Figure 5–5. A block diagram of the Viking Orbiter Command Computer Subsystem hardware. This basic dual computer configuration was used for both Viking computers and all three Voyager computer systems. (From Kohl, *Viking Orbiter Computer Command Subsystem Hardware*)

specifications for the Command Computer and its software load[75]. These were followed by detailed design documents.

Design documents generated for the Viking software and available to programmers were based on a software design description. That volume contained an overview to the mission and the software architecture, and, for each routine, detailed process descriptions, entry/exit points to other routines, variables and their descriptions, constants, and other relevant notes. A flowchart followed each routine's narrative description. Appendices to the document included a hardware description and a reference guide to the instruction set. Programmers were expected to use the design description as a manual. Volume two of the document contained the assembly listings of the resulting flight routines. By opening both volumes to the same routine, it was possible to easily follow the logic of the programs by reading both the narrative and the comments on the listing.

Box 5-3: CCS Hardware

The Command Computer's central processor contained the registers, data path control and instruction interpreter[76]. The machine was serial in operation, thus reducing complexity, weight, and power requirements. It had 18-bit words and used the least significant 6 bits for operation codes and the most significant 12 for addresses, as numbered from right to left. This permitted 64 instructions and 4K of direct addressing, both of which were fully utilized. Data were stored in signed two's complement form, yielding an integer range from $-131,072$ to $+131{,}071$. Average instruction cycle time came to 88 microseconds. Thirteen registers were in the Command Computer, mostly obvious types such as an 18-bit accumulator, 12-bit program counter, 12-bit link register that pointed to the next address to be read, and a 4-bit condition code register that stored the overflow, minus, odd parity, and nonzero flags[77].

Timekeeping on the Orbiter was in three units. The clock issued interrupt pulses every hour, second, and 10 milliseconds[78], similar to the sequencer clocks used in Mariner, save that the 10-millisecond pulse provided finer timing. Pulses entered an interrupt processor that collected and interpreted them before transmission to the central processor. The interrupt processor had 32 interrupt levels, and constantly scanned for the highest priority task being requested[79]. Thus, the Command Computer had the same interrupt-driven concept used in the Apollo and Shuttle manned spacecraft computers and the NSSC-1, but it was accomplished in hardware rather than software.

Viking's Command Computer used 4K of plated-wire memory[80], divided into four equal parts. The first three could be set as either read only, write protected, or read/write, but the last 1K was always read/write[81]. On Viking the first 2K was specified as read only, and the program instructions stored there. The second 2K was read/write, and the data resided in that segment.

Software development was guided by the "Viking 1975 Orbiter CCS and Support Equipment Software Development and Control Plan," which set the standards for production of the flight software and software for ground support and testing equipment. The Cognizant Software Engineer, Cog Engineer, Subsystem representative to the Systems Engineer, and all software design team members reviewed each routine as it was designed and coded[82]. Coding was assisted by the Orbiter Sequence Translator Program, or OSTRAN. Code produced by the programmers was verified by running it in both the CCS Breadboard and the CCS Programming System. The former was a complete hardware version of the Subsystem, and the latter a software simulation. The Command Computer Subsystem Technical Manager,

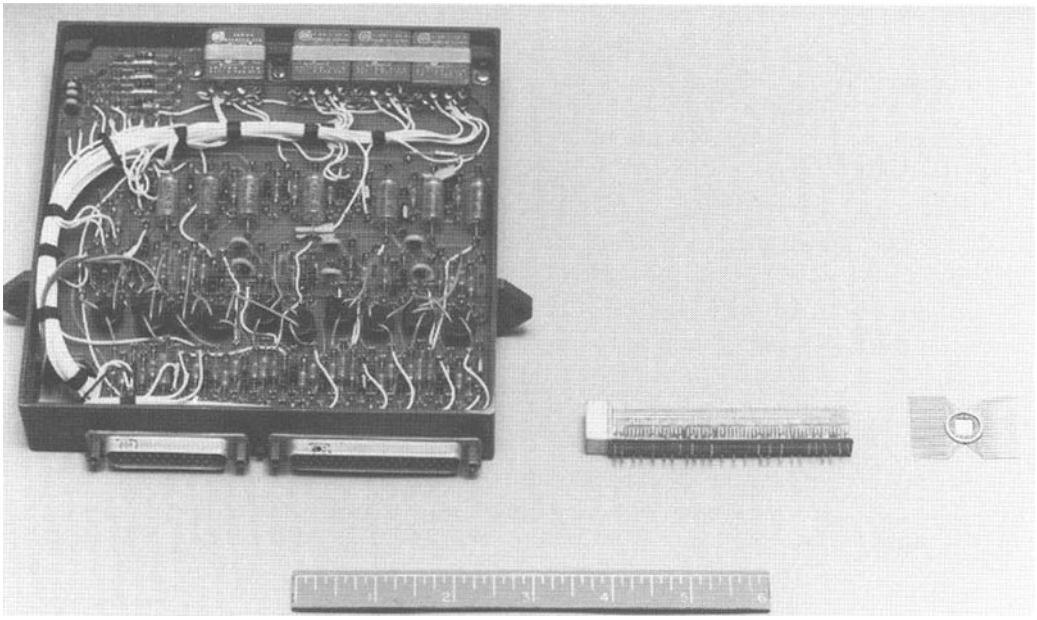

Figure 5–6. Different types of packaging used in the Viking computer system. Note the discrete components in the leftmost device. (JPL photo 360-276-AC)

hardware Cog Engineer, and Cog Software Engineer each compared the performance of the routines on these devices[83].

Testing and system integration was done from the bottom up. Programmers tested individual routines through all options at expected times, all expected branches, and all expected interaction with other routines, and then through selected failures[84]. As with any real-time system, it was impossible to test for all possible failures. After this unit testing, the programmers integrated the routines with related code and ran it on either the breadboard or the flight hardware.

As with the most successful software development projects, JPL exercised strict configuration control. Even though the memory was eight times larger than that on the programmable sequencers, so many functions were transferred from hardware to software that memory was constrained from the beginning. Viking Orbiter Data Management Office handled changes to the documents. The Configuration Control Board, consisting of the Subsystem Technical Manager, the CCS Support Equipment Tech Manager, the hardware Cognizant Engineer, and the CCS Software Engineer, decided on software changes or what to do about discrepancies between the design and code or performance[85].

Box 5-4: CCS Software Structure

The Viking Command Computer software structure appears different from others described in this volume because of the apparent lack of an operating system or executive program. The functional block diagram used in the CCS requirements document (reproduced here) shows that all inputs, either interrupts, or "level" inputs, enter a software block that contains conditioning routines. The TRAP routine maintains 32 memory locations that correspond to the 32 levels of the interrupt processor. Each location contains an instruction to be executed or an address to branch to if the appropriate interrupt occurs[86].

After clearing the input conditioning block, signals are either routed to the command decoding software or to the generation software. The command decoder does just what its title implies: examines the bit streams of commands routed to it for specific orders and then routes them to either the event generator, the output unit driver, or the telemetry processor.

The event generator block contains the most complex software in the system. Its chief routine is the Master Table Driver. Software requirements documents specified that the Master Table Driver handle all time sequenced events, maintaining up to 20 tables at once[87]. Thus, it was the replacement for the programmable sequencer carried on previous missions. Implementation of the Master Table Driver was the TARMEX routine: Timing and Region Management Executive containing many of the common executive functions centralized in other machines. TARMEX is referred to as a "time-sharing executive" in the software documentation, but that is perhaps too ambitious a title[88]. It did regularly scan through the event tables and maintain the time countdowns for a number of mission events. At 454 statements, it was one of the largest routines on the spacecraft. Functionally, it acted like any other sequencer JPL built, the difference being that it was implemented in software and thus highly flexible, which contributed to its success on the Viking mission and later on Voyager.

Other routines in the event generator were used less comprehensively than TARMEX. The Data Acquisition and Playback Routine controlled science instruments, imaging, and data storage until broadcast to earth. The accelerometer control routine was needed because for the first time an unmanned spacecraft would have active control over engine burns, rather than depending on precalculated timed firings. In the past, the maneuver and insertion firings were made based on calculations done before the flight and implemented as timed sequences in the Central Computer and Sequencer.

Viking carried accelerometers and a computer, making it possible for the spacecraft to fire its engines and calculate when to turn them off in real time based on velocity figures uplinked in advance from navigation computers. A Launch/Hold/Reset routine handled spacecraft functions as a fixed sequence during the prelaunch, launch, and early cruise phases of the mission, with the capability to reset its timers if holds occurred in the countdown[89]. This was a more robust version of the sequences carried for the first phase of previous missions. An Error Recovery routine included a programmable version of the 66 2/3-hour command loss sequence implemented in Mariner missions. The computer could be programmed to check for commands at varying times. During cruise, the command loss routine could be set to check just once a week or more, and changed to check at much closer intervals near encounters[90]. Deep Space Network resources were thus less tied up during relatively dormant periods of the mission, as commands did not need to be sent just for the reason of keeping the command loss sequence from starting.

Remaining software blocks were the output driver, which transferred output signals to the appropriate output unit for distribution to the command buffer and eventually the affected systems, and the telemetry processor. The telemetry routine took over some of the functions previously done by the hard-wired Flight Data System. The Flight Data System on Viking had its own dual 1K memories of 8-bit words. Command Computer software helped manage that memory and prepare data streams for transmission. The Checksum routine was similar to that used in the Central Computer and Sequencer, except that a range of addresses could be specified, instead of the entire memory being summed.

Viking Orbiter software had to be written in an assembler, which fortunately had relocatable addresses, simplifying the maintenance task. The 64 instructions were mostly common to other computers, but there was no multiply or divide. There were two sets of loads, stores, increments, and subroutine calls: one used during independent operation and one aimed at dual operation, so that the two memories could be kept equivalent[91]. Even though many interrupts were available, most routines as coded had all but the internal error and counting interrupts disabled[92]. Many routines were free to run out without being interrupted, in contrast to the highly interrupted Apollo and shuttle software. Programmers avoided the memory and processing time overhead required to preserve the current accumulator and register contents during an interrupt.

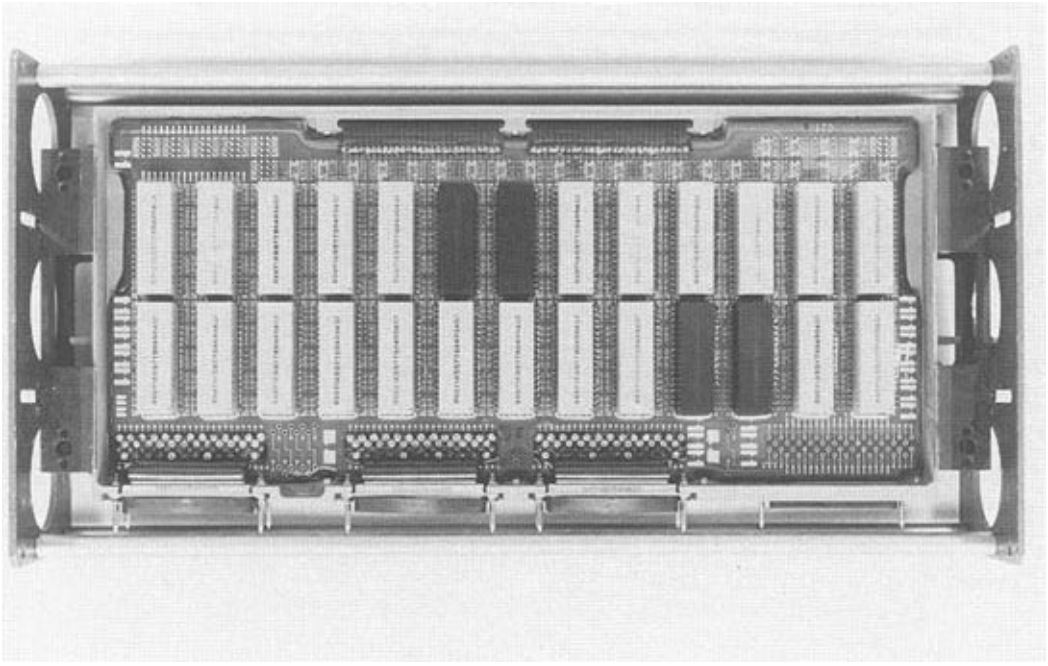

Figure 5–7. A circuit board with integrated circuits used in the Viking Orbiter computers. (JPL photo 360-371-AC)

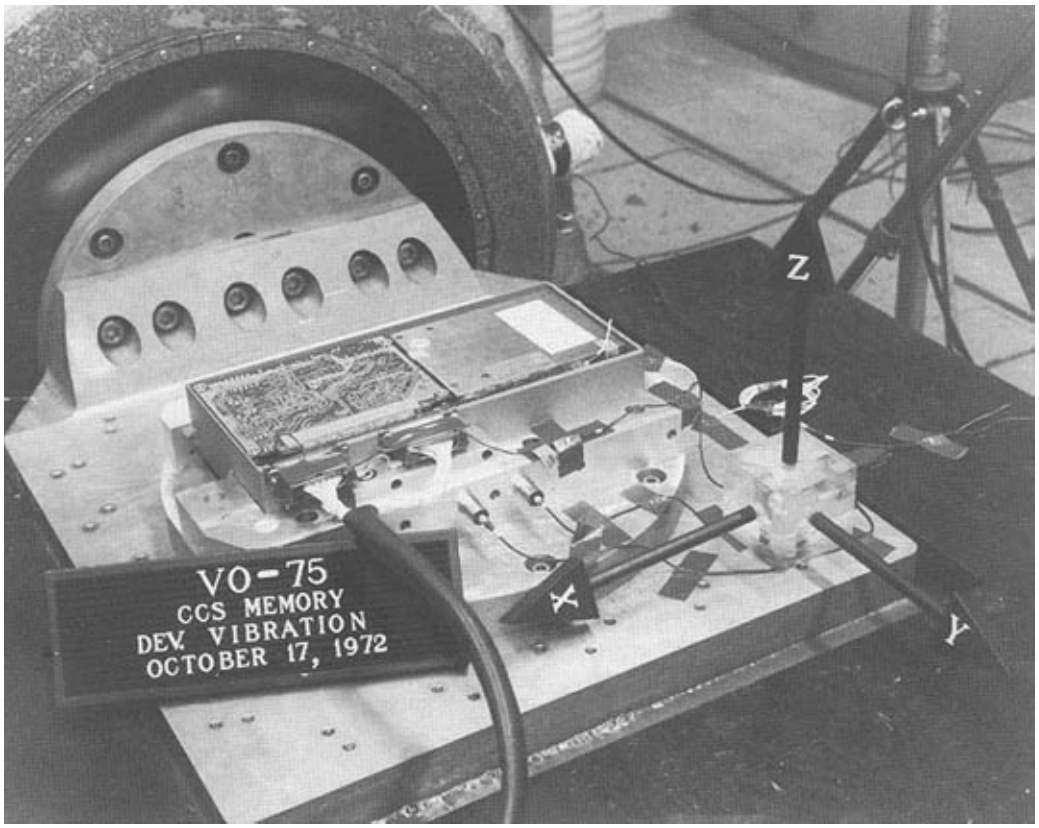

Figure 5–8. One of the Viking Orbiter plated wire memories in a vibration test device. (JPL photo 360-276-AC)

Instituting concepts of full documentation, configuration control, and engineering principles, such as modularization in software development, made it possible to have a successful flight program at launch and to remain successful throughout a long mission. Some of the people involved in Viking left the project before the Orbiters reached Mars, either to other projects, or to leave the Laboratory. With the materials and techniques available to maintain the computer software, it was possible to bring new people into the project and have them make necessary updates and upgrades to the flight program. This capability is as important to a long-term mission as the reliability of the hardware. JPL's concern for Grand-Tour-length reliability in hardware, exemplified by STAR, also extended to software. Without such an attitude the later Voyager would be much more difficult to maintain as an active project.

The Viking Lander GCSC

Martin Marietta's Denver division developed the computer system for the Viking Lander in an innovative way. To this point the stories of the development of various on-board computer systems have a similar theme: Project managers determine the expected specifications of the system, choose the hardware, and develop the software. By the 1980s, the danger of this approach became evident to computer and software engineers and to some of their customers. Choosing the hardware first and then developing the software for an embedded computer system runs the risk of the eventual software exceeding the hardware's capabilities or capacity. If the hardware is chosen before the full requirements of the mission are known, which was often the case, then the software is written in such a way as to compensate, thus exceeding the memory size because the compensating programs were not in the original software estimate. If the hardware turns out to be more powerful than needed, the software is expanded to take advantage of the additional capability, so it pushes the hardware to its limits. Either way, the development of the computer system and its software becomes more complex, expensive, and late. The Gemini, Apollo, Shuttle, NSSC-1, Mariner X, and Galileo projects all suffered because of insufficiencies in computing power or memory, largely because of poor specifications.

Martin Marietta did a number of military projects that repeated the same mistakes that the space program had made in regard to onboard computers. In 1970, when the company received the Viking Lander contract, it determined to follow a different course of development by adopting a policy of "software first"[93]. This was one of the earliest attempts to break the paradigm of specification/ hardware selection/ software development/ reaction to changed requirements, a

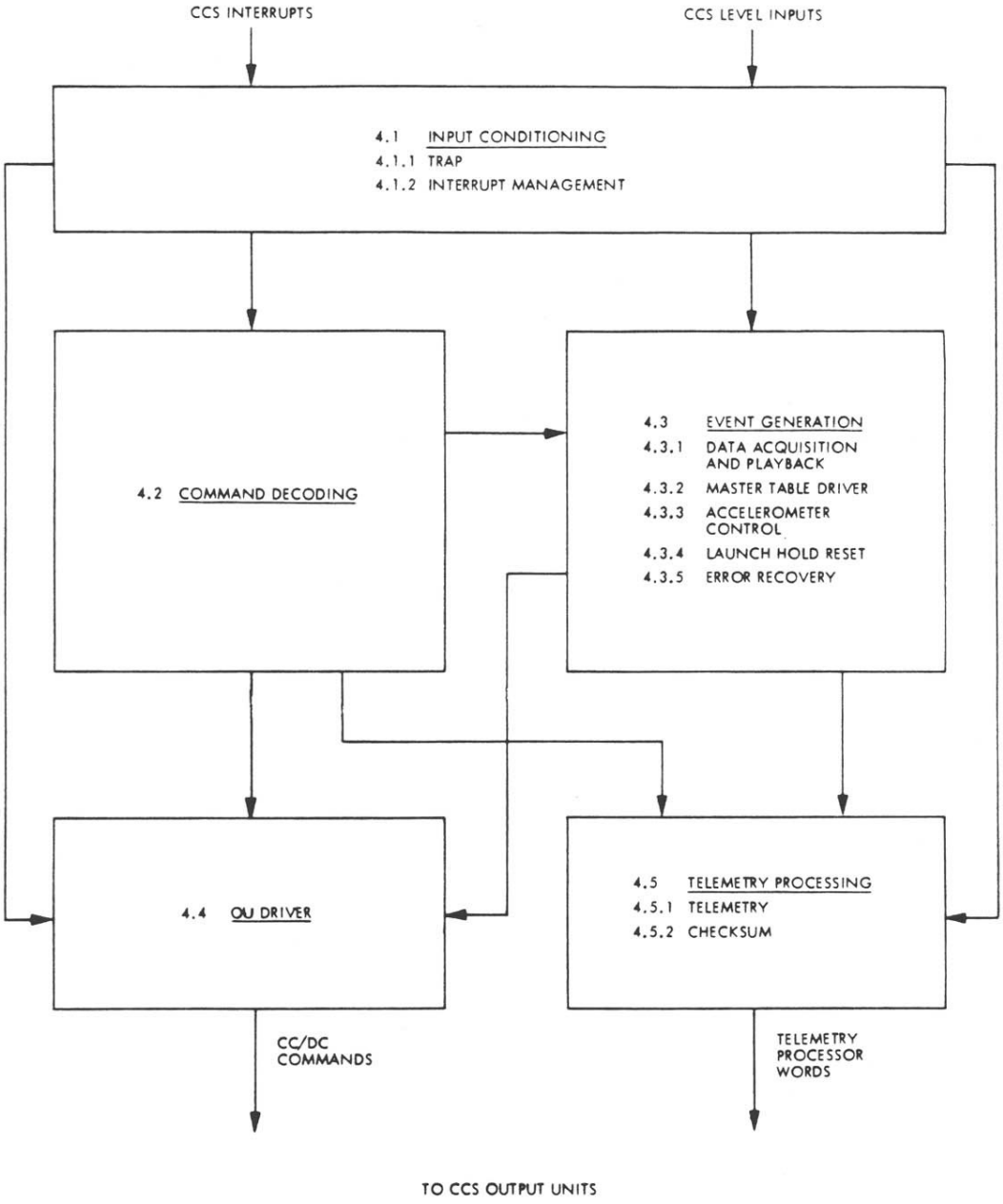

Figure 5–9. A block diagram of the Viking Orbiter software. This same software structure was used for the Voyager Command Computer Subsystem software (From Kohl, *Viking Orbiter Computer Command Subsystem Software*)

cycle that crippled many projects. The decision turned out to be highly

successful. The Air Force studied the results and disseminated the technique. Thus, it contributed to a shift in attitude which, though only barely established 15 years later, is still a turning point in the history of computing.

"Software first" techniques make it possible to compensate for the severest deficiency of most projects, incomplete or incorrect requirements specification. Hardware decisions and software sizing are based on the requirements document developed early in the program. Users of the eventual product must be careful in contributing to this document. In the Apollo program, NASA gave MIT a broad statement of requirements, basically to develop a guidance system capable of navigating a spacecraft to the moon and back. Painful lessons learned as a result of the Apollo project included a greater appreciation for more detailed specifications. As a result, the requirements for the Shuttle were among the most outstanding written to that time. Still, memory estimates were far off, stretching the computer system to its limit. In contrast, the Viking Lander software developed contemporary to the Shuttle kept to its original boundaries, staying within the hardware capabilities of its computer. Both Shuttle and Viking had extremely high change traffic, and both project management teams anticipated many changes; but the software first philosophy handled change differently. Martin Marietta recognized that it is easier to change software independently of hardware than to react to revised specifications. More importantly, it recognized that if it is necessary to change hardware requirements, it is easier to change before it is purchased than afterwards. If the software is developed first, the hardware can be bought to fit it. Martin Marietta completed the flight software for the Lander 1 year before the hardware was delivered, which was only 2 months before the launch[94]! The company accomplished this feat because detailed timing and sizing experiments gave confidence in the eventual hardware selection[95].

To implement the concept of software first, Martin developed a Viking Controls Mock-up Unit (VCMU) using two Standard Computer Corporation IC-7000 computers. IC-7000s had a two-section CPU. The Viking team microprogrammed one processor as an emulator of the proposed GCSC on the Lander; the other processor acted as a simulator of the other spacecraft systems[96]. This system could be linked to an IBM 360/75 used at JPL for mission control, thus providing simulations of flight operations[97].

Differences between an emulator and a simulator are rather fine but very important in this case. A simulator imitates a computer using software that functions interpretively. For example, a simulator running a program written in the machine language of the target computer executes a set of instructions in its own machine language that has the same effect. The problem with this method is that it has variable results. Even though most simulations are done on

computers more powerful than the target machine, performance is far less, and a real-time simulation is virtually impossible. Even such simple instructions as an ADD, which can usually be simulated using a single instruction, run much slower because of the software overhead involved in maintaining pseudoregisters and fake memory. Programs that run in a minute on the target machine might take as long as an hour on a simulator.

An emulator runs the target machine's program in near real time. In fact, its performance on some instructions is likely to exceed that of the target due to the performance difference between the emulator and the actual hardware, but other instructions run slower, creating something of a balance. Microprogramming makes it possible to achieve these results. Older computers had the control unit that handles the flow of signals in the computer permanently hard-wired during manufacture; therefore, the way a particular computer executed instructions was fixed throughout its operating life. As early as 1950, Maurice Wilkes of Cambridge University suggested representing the control paths in the form of special software. He called these control programs "microprograms" and their instructions "microinstructions" to differentiate them from higher level programs and code[98]. Such "microcode" could not be implemented in the 1950s because suitably cheap and permanent memory was not available. The IC-7000 was a microprogrammable machine, so its microcode could be changed by Martin Marietta to make the processor execute instructions like another computer. Martin started with a reasonable set of instructions and tried to write the software. If a problem or change arose that would be better handled by hardware or a new instruction or two, the microprograms for the new operation codes were installed[99]. In this way, the hardware evolved along with the software, and when a fully functioning software load was complete, the hardware requirements were also complete.

Ironically, other constraints eventually thwarted Martin's plan for the computer hardware. Developers working on other subsystems of the Lander had trouble delivering hardware that could accomplish all the mission goals without increasing its weight. So when the time came to purchase the computer, the weight gains by the other systems had to be compensated for by the only system without hardware. Therefore, the computer that flew on Viking was actually the "third best" of those available, its chief deficiency being a poor instruction set, but it weighed less than the first choice[100]. Martin changed the software affected by the less powerful computer. Even though the optimum computer did not fly, the principle of software first was demonstrated. Additionally, Martin introduced the concept of using off-the-shelf equipment for unmanned spacecraft projects. Despite the care taken to anticipate problems, some of the most common development difficulties occurred.

In an environment created to anticipate change, the lack of detailed software requirements and a large number of change requests still caused serious problems[101]. At one point, testing came to a standstill, which turned out to be fortunate in that the Systems Engineering Director began to take software seriously and to treat it like hardware, a lesson painfully repeated on project after project[102]. And again, memory sizing, though controlled, posed difficulties. Martin completed the prototyping for the Lander software in July 1971. Its actual size at that point was 13K. Martin engineers specified 18K, anticipating inevitable growth to accommodate new requirements and set up a control group to ensure that the memory stayed under that size. Twice during development the software exceeded memory limits, first in March–May 1973, when it topped off at 18.5K, and then in June 1974, when it hit 19K. Both times the flight program was reduced to the correct level[103]. These overruns are minimal compared to those of the Apollo and Shuttle programs.

Eventually the software for the Lander reached 20,000 words and required 1,609 man-months to produce (the reason more than 18,000 words are shown here is that some routines used after landing overlaid landing software). Over 200,000 instructions of emulator and simulator software were produced, requiring just 494 man-months[104].

Differences in the proportion of development time to instructions are because the Lander software was hand-coded, whereas the simulators could be written with the aid of assemblers and higher level language compilers. Langley Research Center project managers determined that the flight software would be verified by an independent organization, so TRW Corporation was contracted to provide such services on site at Langley[105]. Such completely independent verification is somewhat more useful than an “independent” quality assurance group within a company, as it has a more adversary relationship.

Box 5-5: Viking Lander Computer Characteristics

The GCSC consisted of two Honeywell HDC 402 processors, each with 18K of 2-mil plated-wire memory. These processors had the capability of eight levels of interrupt and an average 4.34-microsecond instruction cycle time[106]. Original plans for the Viking Lander specified a single computer for the landing phase and another for on-surface operations, but when the project was delayed this changed to a dual redundant system similar to the Orbiter CCS[107]. Honeywell's computer had a 24-bit word size, with 47 instructions, and used two's complement representation for data. Compared to the NSSC-1 and Viking Orbiter computer, it is slightly faster than the former and much faster than the latter, with better numerical precision than both.

Lander software structure reflected common short-cycle real-time control concepts such as those used in the Space Shuttle Main Engine Controllers. During descent, the software executed a 20-millisecond control loop, cycling through a set of routines[108]. Martin claims that the executive was a "virtual machine" facility, in that each process "thought" that it had its own machine and was not sharing resources with other processes[109]. Galileo Command and Data System software developers used the same terminology, but on that spacecraft the virtual machines resided on several microprocessors and were more truly "virtual." Martin's system is more like the cyclic time-sharing executive found in the Shuttle Backup Flight System.

One problem Lander software developers had was that no adequate assembler was ever written for the computer, perhaps because of the changing nature of the instruction set[110]. Patches had to be hand-coded in octal, with many jumps to unused memory space because of the lack of an assembler with relocatable addressing. A programmer trying to trace a routine thus had to contend with having to go back and forth on the memory map to follow the logic. JPL's Viking programmers could keep their routines in contiguous memory locations by reassembling the code after changes. The assembler would automatically move the data around to accommodate the modifications.

Lessons learned in the Viking Lander computer system development program influenced Martin Marietta's future work. After the VCMU outlived its usefulness, the organization and equipment were renamed EMULAB to reflect what takes place inside it. The Air Force requested that Martin study its software development practices during its participation in the space project, resulting in a report entitled *Viking Software Data*[111] and issued by the Rome Air Development Center at Griffith Air Force Base, New York. This report and the experience gained influenced the continuing shift from "hardware first" to "software first" among some contractors in the late 1970s and early 1980s. As microprocessors become military- and space-rated, it will

become easier to adopt such a sequence because readily available microcomputers can be adapted to specialized functions. Users are also becoming more likely than before to adopt microprogramning to tailor instruction sets, as in the shuttle general-purpose computers and the Galileo attitude control computer.

ON TO THE OUTER PLANETS

Experience and the appreciation of the flexibility of computer processors are the legacy of the computer systems development for the inner planet probes. Consistent and detailed documentation, simple, reliable, and reusable hardware designs, and the practice of many missions contributed to the later and continuing success of Voyager. Just as management experience gained during Apollo applied to the shuttle, JPL's success with Viking made the concurrent development of Voyager and Galileo easier. People like Samuel Deese, who gained practical experience in the 1960s, led subsystem management in the 1970s. Viking's Wayne Kohl went on to Galileo after the Mars landings in a position similar to the one he held on the former project. Both Voyager and Galileo are better projects because of the continuity of techniques and personnel.