3

The Skylab Computer System

Skylab, America's first orbital workshop, carried a highly successful computer system. For much of the operating life of the space station, the computer was not just the fourth crew member but the *only* crew member. It made a large contribution to saving the mission during the 2 weeks after the troubled launch and later helped control Skylab during the last year before re-entry. The entire system functioned without error or failure for over 600 days of operation, even after a 4-year and 30-day interruption. It is significant as the first spaceborne computer system to have redundancy management software. The software development for the system followed strict engineering principles, producing a fully verified and reliable real-time program.

The record of the computer system stands in contrast to that of the workshop itself. NASA launched Skylab on May 14, 1973 on a Saturn V booster. The first two stages put the modified S-IVB third stage into orbit. The S-IVB contained the workshop, which included a solar telescope mount and living and working quarters. The plan was to launch the first crew the next day aboard a Saturn IB carrying an Apollo command and service module. However, shortly after achieving orbit, telemetry from the unmanned Skylab indicated that one of the two wings of solar panels was missing and the other had not deployed. The panels on the Apollo Telescope Mount (ATM) had opened properly but they were too small to supply power for the whole workshop. In addition, the gyros were drifting and the thermal shield was damaged. These failures caused concern that the interior of the space station would overheat and destroy the equipment. The damage was so serious that for the first 3 or 4 hours the ground controllers felt that NASA would be fortunate if the systems were to function for 1 day[1]. However, by using the computer system that controlled the workshop's attitude, the ground controllers were able to keep the Skylab at angles to the sun such that the equipment would be exposed to tolerable temperatures in the laboratory in concert with generating adequate power from the remaining solar panels. At times these were conflicting requirements. This had to be done for 2 weeks while engineers prepared repair materials for the crew to fix the workshop. Controller Steven Bales remembered that time as "the hardest 2 weeks I have ever spent," since a 24-hour watch had to be maintained on the attitude and temperature[2].

The computer system again served as "captain" during the entire Skylab reactivation. The workshop systems were shut down on February 9, 1974, after the last crew left. NASA expected that the Skylab would stay in orbit until the mid-1980s. By that time the Space Shuttle would be operational and, it was thought, could be used to bring up rockets to boost the laboratory into a higher orbit. However, unexpected solar activity in the mid-1970s resulted in an increase in the density of the atmosphere, so the Skylab's orbit decayed at a much faster rate than projected[3].

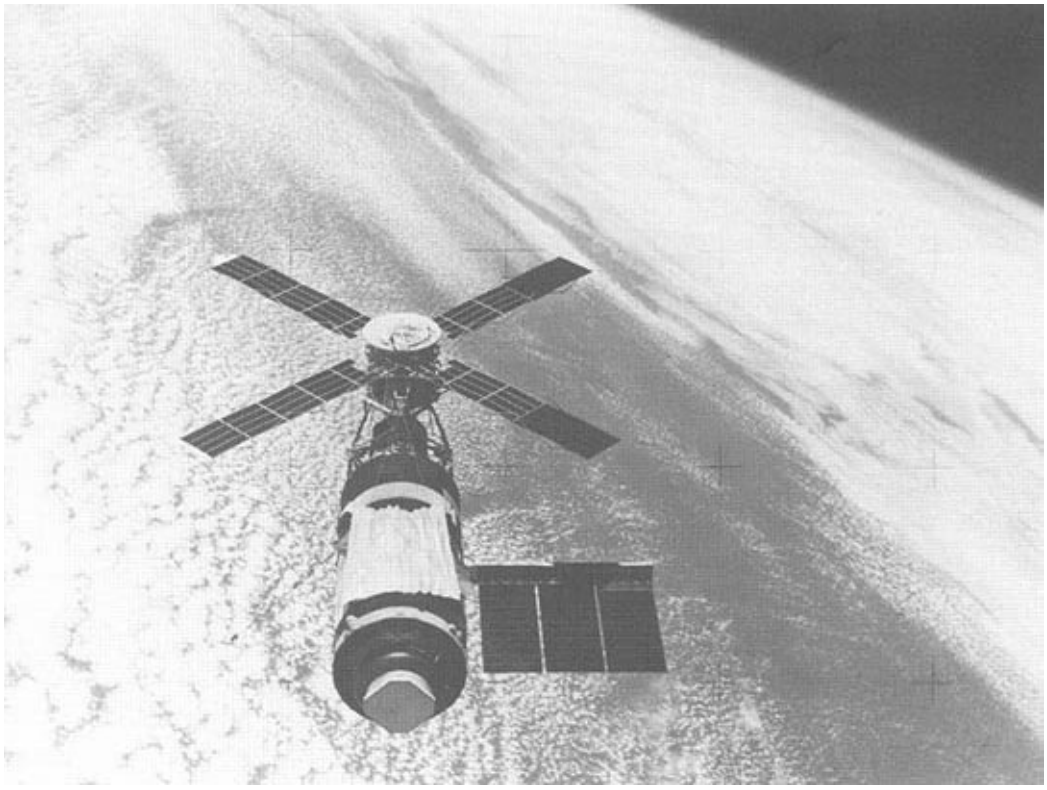

Figure 3–1. Skylab in orbit. Note the foil sun shield above the center section and the missing large solar panel. The Apollo Telescope Mount is the section with the "windmill" solar panels. (NASA photo 74-H-98)

By 1978, the predicted re-entry time was to be late that year or in early 1979. NASA decided to attempt to change the attitude of the workshop so that minimal drag would ensue. In this way, the orbit might be maintained until the Shuttle could rescue the space station. Engineers reactivated and reprogrammed the computer to maintain the proper attitude and, later, to control the re-entry when NASA abandoned the attempt to maintain orbit. They accomplished this over 4 years after the computer was shutdown.

The need for the computer system that served Skylab so well was not apparent until the original "wet workshop" concept (the laboratory to be assembled in space inside of the empty propellant tanks of the last stage of the launch vehicle) had progressed through more sophisticated designs to the eventual "dry workshop"[4]. In December 1968, NASA decided to acquire a dual computer system to help control attitude while in orbit[5]. Attitude control was crucial to the success of the solar experiments. In fact, the name of the computer reflects this: Apollo Telescope Mount Digital Computer (ATMDC). Two of these computers were a part of the Skylab Attitude and Pointing Control System (APCS), which consisted of a number of other components, such as an interface unit, magnetic tape memory, control moment

gyros, the thrustor attitude control system, sun sensors, a star tracker, and nine rate gyros[6].

Marshall Space Flight Center devised this complex system—a pioneering effort because it represents the first fully digital control system on a manned spacecraft[7]. Its mission-critical status led to the use of extensive redundancy in its design, in both hardware and software. The computer system not only managed its own redundancy, but all redundant hardware on the spacecraft[8]. The uniqueness and complexity of the control laws associated with the control moment gyro attitude system led one NASA engineer to refer to it as "a crazy animal"[9]. It was up to the Skylab computer system to tame it.

HARDWARE

The choice of a central processor for the Skylab computer system marked a break from NASA's previous practice. The Gemini and Apollo computer systems were custom-built processors. Apollo did have an immediate predecessor, but the number of changes necessary before flight negated most of its resemblance to the Polaris system. To the contrary, Skylab and, later, the Shuttle, used "off-the-shelf" IBM 4Pi series processors, though they both needed the addition of a customized I/O system, a simpler and necessarily idiosyncratic component. By using existing computers, NASA avoided the serious problems associated with man-rating a new system encountered during the Apollo program.

The 4Pi descended directly from the System 360 architecture IBM developed in the early 1960s. Some 4Pis were at work in aircraft by the latter part of that decade. The top-of-the-line 4Pi is the AP-101, eventually used in the F-15, B-52, and Shuttle. The version on board Skylab was the TC-1, which used a 16-bit word, in contrast to the AP-101's 32 bits. A TC-1 processor, an interface controller, an I/O assembly, and a power supply made up an ATMDC[10]. Each flight computer had a memory of 16,384 words[11]. This memory was a destructive readout core memory, which means that the bits were erased as they were read and that the memory location had to be refreshed with the contents of a buffer register, which saved a copy of the bits before they were passed on to the processor. The memory was in two modules of 8K words each[12]. Addressing ranged from 0 to 8K, with a hardware switch determining which module was being accessed[13]. The redundant computer system was composed of two processors attached to a single Workshop Computer Interface Unit. The unit consisted of two I/O sections (one for each computer), a common section, and a power supply[14]. Only the I/O section connected to the active computer was powered. The inactive computer and its

I/O section of the interface unit were not powered. The common section contained a 64-bit transfer register and timer associated with redundancy management[15]. The transfer register and timer were the only parts of Skylab that consisted of triple modular redundant (TMR) circuits[16]. Basically, TMR circuits sent signals in triplicate on separate channels and then voted. The single output from a TMR voter represented either two or three identical inputs.

The final component of the computer subsystem was the Memory Load Unit. The original design did not contain one, but, like the Gemini Auxiliary Tape Memory, engineers later added it. Whereas the Gemini tape unit was useful in handling memory overloads, designers included the Skylab tape unit to further increase the reliability of the system. It carried a 16K software load and an 8K load that could be written into either module of either memory of the ATDCs. If up to three modules failed, the mission could continue with reduced capabilities with an 8K program loaded into the remaining module. This raised the total reliability of the system from a factor of 0.87 to 0.97[17]. The tape load would take a maximum of 11 seconds[18].

NASA decided to add the Memory Load Unit in the summer of 1971, when both IBM and Marshall realized that a Borg–Warner tape unit, like the two already used as telemetry recorders, could be upgraded for program storage. IBM imposed some manufacturing changes on the recorders (primarily piece part screening) to make the process more nearly match the care taken in constructing the computers[19].

NASA awarded the contract for the computer system to IBM on March 5, 1969[20]. By October, designers froze the choice of processors and their configuration, a decision heavily influenced by the concern for redundancy and reliability[21]. The first computer was delivered on December 23, 1969. IBM eventually built 10, the final 2 being the flight versions, which went to NASA on February 11, 1972, over a year before launch. Two of the ATMDCs and an interface unit were turned over to IBM for use in testing both hardware and software, ensuring that the final verification would be on actual equipment rather than simulators[22].

IBM took great pride in delivering on time without sacrificing reliability. In applying Saturn development techniques to the Skylab equipment, for example, IBM required all piece parts to exceed expected stress levels[23], and prepared the ATMDC for thermal conditions, the most dangerous stress to electronic components[24]. A number of design problems, including thermal and vibration difficulties, analog conversion inaccuracies, and interconnection failures, had to be overcome[25]. To make up time lost handling these problems, IBM sometimes went to a 7-day, three-shift debugging cycle[26].

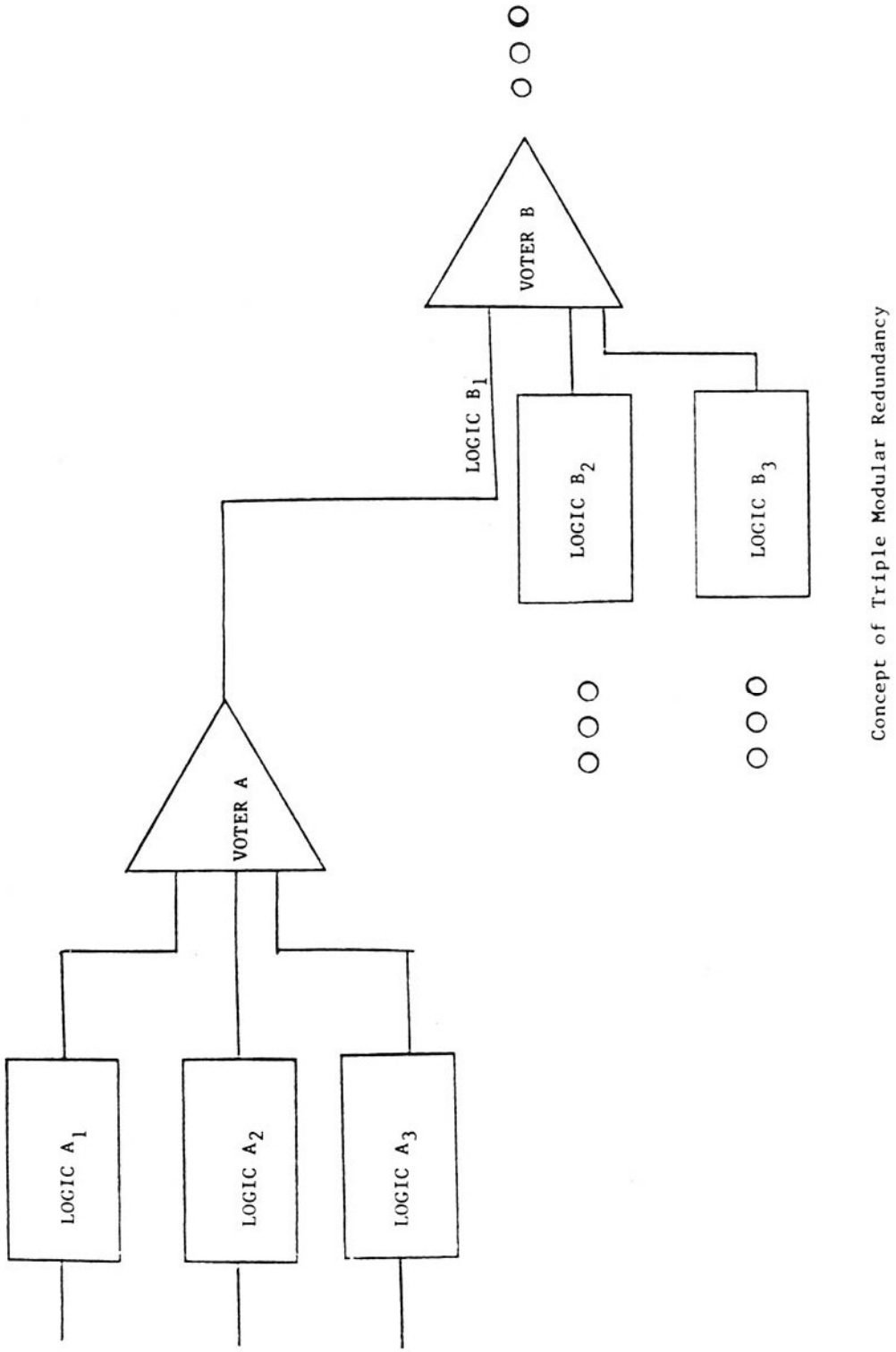

Figure 3–2. The concept of Triple Modular Redundancy.

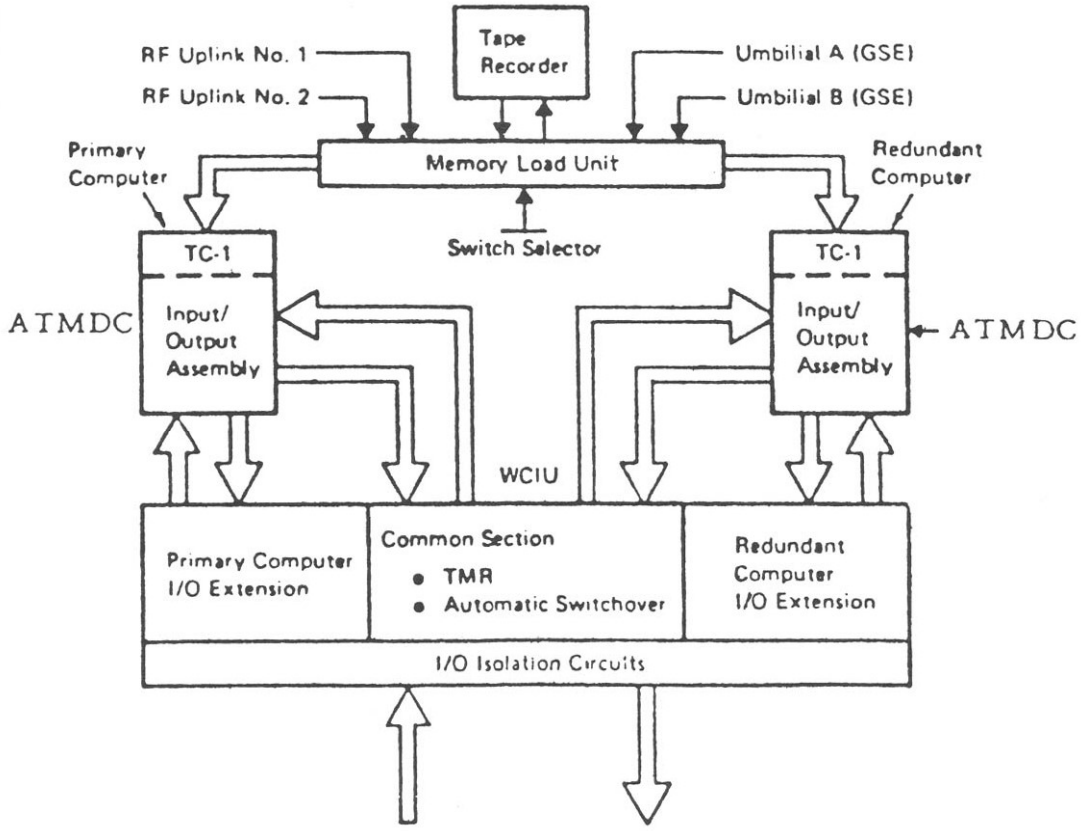

Figure 3–3. A block diagram of the Skylab Computer System with the dual ATMDCs, tape memory, and common section shown. (From IBM, *Skylab Operation Assessment, ATMDC*, 1974)

Probably due to the care taken in manufacture, the computer system had no failures. A planned ground-initiated switch-over from the primary to the secondary computer occurred after 630 hours of orbital operations. The second computer then ran the remainder of the 271-day mission[27]. On the final day, the system did another switch-over and used the tape unit for the first time, primarily to prove that it would work. A transmission of software from the ground to the computer was also practiced. IBM's reports of the performance of the hardware are quite self-congratulatory but, based on the actual record, justified.

SOFTWARE

IBM wanted to do a careful job on the software for Skylab. In the late 1960s and early 1970s, the company internally pushed the development and implementation of software engineering techniques. IBM learned many lessons from the creation of the OS/360 operating system, and various government-related projects. Two IBM software management experts, Harlan Mills and Frederick Brooks, circulated these lessons both within IBM and to the computing public[28]. The small size (16K) of the Skylab software and correspondingly small group of programmers assigned to write it (never more than 75 people, not all of whom were programmers, and only 5 or 6 for the reactivation software), meant that the difficulties in communication and configuration control associated with large projects were not as much of a factor. Also the IBM programmers were specialists. MIT assigned engineers to the programming of the Apollo computer, assuming that it was easier to teach an engineer to program than to teach a programmer the nuances of the system. This turned out to be a mistake, which MIT acknowledged[29]. Thus, the stage was set for IBM to produce a superb real-time program. However, the complexity of the control moment laws, the redundancy management needs, and the inevitable memory overrun kept the development from being simple.

Requirements Definition and Design

IBM and NASA jointly defined the requirements for the Skylab software. Marshall Space Flight Center delivered the detailed requirements for the control laws, navigation, and momentum management, leaving lesser items such as I/O handling to the contractor. IBM and NASA made a parallel effort to determine if the equations actually worked[30]. The result was the Program Requirements Document (PRD), issued July 1, 1970[31].

The actual design, the Program Definition Document (PDD), was released later and served as the baseline for the software, which meant that the design could not be changed without formal review. The software resulting from these documents ranged from 9,000 words to nearly 20,000 words of memory. Since the memory size of the computer was just over 16,000 words, a “scrub” was necessary, continuing the NASA tradition of exceeding the memory size of an already-procured computer by the time the planners knew the final requirements. Managers had not yet learned that software needs should drive the hardware choices. Engineers changed the control moment gyro logic to reduce core usage and made other cuts[32]. Memory

became the prime consideration in allowing requirements changes[33].

Architecture and Coding

Skylab gave IBM an opportunity to demonstrate how to do software development right. The company carefully separated the production process into strictly designed phases. Two different flight loads resulted: one full-function program that filled the 16K memory, and an 8K version as a backup that needed only one module for storage. These two programs needed slightly different architectures, or schemes for organizing the execution of functions, which made the job tougher. Also increasing complexity was the requirement for redundancy management. An advanced development environment helped keep the complexity under control.

Production Phases

IBM developed the software load for Skylab in four baselined phases. Originally, three were planned: Phase I, Phase II, and Final, but numerous changes made during Phase I required an intermediate stage Phase IA. Crews used the software resulting from Phase IA for training in the simulators in Houston[34].

The PDD for Phase I was released on November 4, 1970, and coding began[35]. The Phase I program contained most of the major components of the eventual flight load, including discrete I/O and interrupt processing, command system processing, initialization, redundancy management, attitude reference determination, attitude control, momentum desaturation, maneuvering, navigation and timing, ATM experiment control, displays, telemetry, and algorithms for utilities[36]. IBM's programming team completed and released the Phase I program for verification on June 23, 1971. It consisted of 16,224 words, filling about 99% of the computer's memory[37].

It was this situation that led to the added phase, which was chiefly a memory scrub. Not only was Phase I a fairly extensive program, three modules still had to be coded and many changes would likely occur in the nearly 20 months remaining before launch[38]. By the time IBM delivered Phase IA on February 9, 1972, it had incorporated 45 waivers and 105 software change requests (SWCR) made after the thirteenth revision of the design[39]. This meant that nearly 40% of the original program was changed. Even with the attention to memory size, the new software amounted to 16,111 words, or 98.3% of the locations.

Phase II represented another extensive revision of the software. The baseline for it was Phase IA plus 49 approved change requests. By delivery on August 28, 1972, 102 additional changes had been incorporated and the design was up to revision 19[40]. Therefore, software engineers modified about 35% of the program. The memory usage rose to 99.7%, or 16,338 locations. The final version reduced this to 16,329 words. The difference between Phase II and the flight release was only 17 additional changes. IBM made the delivery March 20, 1973, 2 months before the launch.

Architecture: The 16K Program

The ATMDC software divided into an executive and applications modules. The executive module handled the priority multitasking, interrupt processing, supporting the interval timer and also basic timekeeping chores[41]. Applications consisted of three major groups: time-dependent functions, asynchronous functions, and utilities. Time-dependent functions were executed in three cycles, with the possibility of higher priority jobs interrupting the currently running module. The cycles were differentiated by time: There was a "slow loop" each second, an "intermediate loop" executing five times each second, and the switch-over processor running each half second[42]. Designers grouped appropriate modules in a cycle. An exception to the cycle groupings, but nevertheless time dependent, was the output-write routine, which was run between intermediate loops in order to take more efficient advantage of the system resources. The switch-over process aided in redundancy management, as explained below. Asynchronous functions could be called at any time, one of which was telemetry, which sent 24 strings of 50 bits per second. The other was the command system, which could receive signals from either the ground or the Digital Address System (DAS, the crew interface) in the workshop. Those signals resulted in interrupts. Utility functions included such common algorithms as square root, sine, and cosine, and unique functions such as gimbal angle computations and quaternion multiplication[43].

Interrupt handling was quite straightforward. Each application module had a specific priority ranking. Tasks could be requested by several means, such as interrupts, discrete signals, elapsed time, or by the direct request of another program. Any current task could be interrupted when a new task was requested. The priority of the new task was immediately entered into the priority level control tables. If the new task was of a higher priority than the current task, the computer did the new one first. When telemetry or the command system requested a task, its priority was entered on the table, just like tasks called in the other ways[44]. The standard telemetry signal

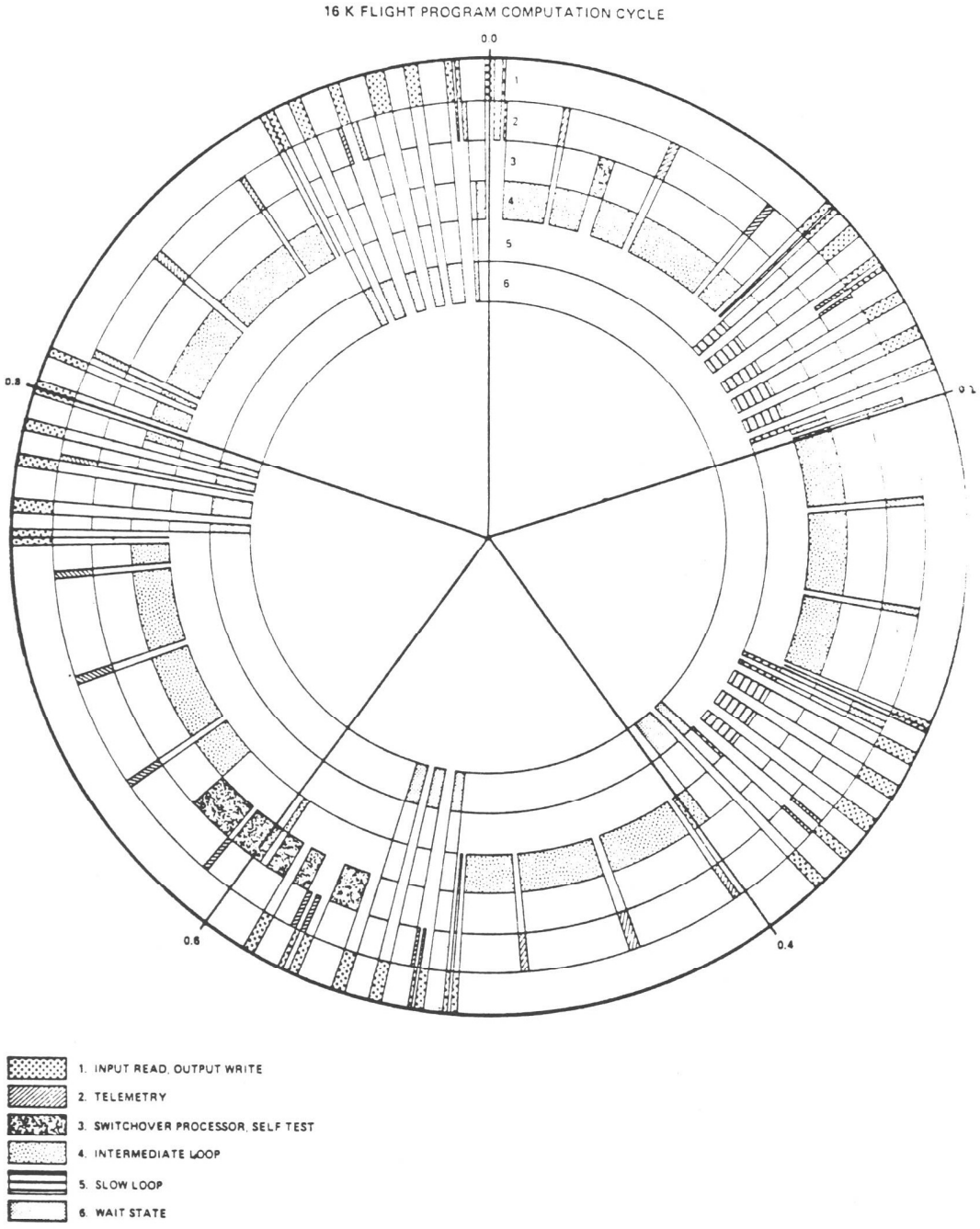

Figure 3–4. The real-time cycle of the Skylab 16K flight program. (From IBM, *Skylab Operation Assessment, ATMDC*, 1974)

functioning as a Digital Command System (DCS) word consisted of 35 bits. Buried in it were an enable bit, an execute bit, and 12 information bits. The enable and execute bits caused an interrupt, making it possible for the data to be stored[45].

The 16K program had a computation cycle consisting of six levels: experiment input, Control Moment Gyro gimbal rates, Workshop Computer Interface Unit tests, and the command system processor; telemetry output; the switch-over timer (reset each second) and 64-bit transfer register (refreshed about once every 17 seconds); the intermediate loop (made up of Control Moment Gyro control); the slow loop (containing timing, navigation, maneuver, momentum management, display, redundancy, self test, and experiment support functions); and the "wait" state (when all functions in a particular cycle finished, about 15% of cycle time in the flight release of the program, depending on the number and nature of interrupts[46].

The 8K Program

The 8K program was strongly related to the 16K program in that the larger version served as the model for the smaller. Its design, released April 3, 1972, developed from the Phase IA version of the software. IBM delivered the 8K program on November 14, 1972 after 10 weeks of verification activity. The functions of the short program were largely limited to attitude control and solar experiment activity and data handling[47]. It was 8,001 words in length. IBM reduced the number of levels in the computation cycle of the 8K program to four: Level I handled command processing and I/O to the Gyros, Level II did telemetry, Level III consisted of the time-dependent functions from both the original intermediate loop and slow loop, and Level IV was the wait state[48].

Redundancy Management

All mission-critical systems in Skylab were redundant. The computer program contained 1,366 words of redundancy management software[49]. At less than 10% of the total memory, it was a bargain. Managing redundancy with stand-alone hardware and solely mechanical switching would have added much more cost, weight, and complexity to the workshop design, with the loss of a certain amount of reliability.

The redundancy management software consisted of two parts: self tests of the computer system and an error detection program for

mission-critical hardware not in the computer system. Self tests of the computer were quite extensive: Logic tests might involve doing a Boolean OR operation on the contents of a register to see if a carry occurred; operation tests required executing EXCHANGE and LOAD instructions; and arithmetic tests meant executing an ADD and checking for planned answer[50]. IBM also designed tests for memory addressing and I/O[51].

The error detection program examined critical signs in several systems. If a failure was detected in attitude control hardware such as the Control Moment Gyros, rate gyros or acquisition sun sensors, then backups or reconfigurations were activated[52]. During the mission, one Control Gyro and several of the rate gyros failed. In fact, a "six-pack" of replacement rate gyros had to be brought up by the second crew.

Switch-over between the two computers was handled by the error detection program or automatically activated by the TMR timer circuits. If self tests indicated a computer hardware failure or that the software was not properly maintaining the workshop's attitude, switch-over would then be initiated. The timers were supposed to be reset about once each second during the computation cycle, after which they then counted down until reset. If two of the three reached zero, then switch-over occurred[53]. Besides automatic switch-over, the crew or the ground could initiate it, as actually happened in mid-mission. So that the secondary computer would be properly activated, a 64-bit transfer register was kept loaded with relevant data. This register, like the timers, consisted of TMR circuits. Great care was taken to ensure that data loaded into the transfer register were uncontaminated. A write operation to the register was restricted in length to a period of 672 microseconds plus or minus 20%, which was just about how long it took to write 64 bits into a redundant circuit. This operation could only take place after 1.5 to 2.75 seconds had elapsed since the last write, so the computer would not accept transient signals as correct data and a new write could not interfere with an earlier write[54]. Besides this "time-out feature," the transfer register could only be refreshed *after* a successful execution of the error detection program[55]. This way, data could not be written to the register from a failed computer.

The redundancy management software was a step toward the eventual Shuttle redundancy management scheme. Previously, IBM had used TMR hardware to ensure reliability. This system, with its watchdog timer, was software based and, in effect, saved space and weight. Two ATMDCs were smaller and required less power than a single TMR computer of equal reliability.

The Development Environment and Integration

The Skylab software development was done in a programming environment that took advantage of useful software tools and proper integration techniques. Binary code for the computer was in hexadecimal (base 16) format, and loaded in that format[56]. Hand coding in hex is rather tedious, so IBM prepared an assembler to translate mnemonics into it. They also provided a relocatable loader for placing separately coded modules in contiguous memory locations. Macros, blocks of frequently used code, were kept in common libraries. Listings of programs and the original source resided in an IBM System 360/75[57]. This environment was small compared with the later Software Production Facility for the Shuttle, but the concept of a good tool set, promoted by IBM's Mills and Brooks, was well realized.

Integration of the Skylab software followed a top-down approach: The program was highly modular so as to keep individual functions separate for easy modification and also simple enough for a single programmer to handle. The executive and major subprocesses were coded and integrated first; then the remaining modules were added. The modules were grouped into three batches, so all the modules in a batch were added and tested, then the next batch would be added, and so on[58]. This helped in the integration process.

Verification

The software for Skylab was one of the most extensively verified systems of its era. Since it was a real-time program, verification was more difficult than a corresponding batch program because it is hard to replicate test inputs when interrupts can occur at any time; thus, a combination of simulators is needed to properly verify a real-time program.

IBM used a number of different simulation configurations in the verification process. The AS-II simulator consisted of a System 360/75 used for analysis of the Skylab while it was in orbit. It could evaluate the effects of changes to the flight program. The Skylab Workshop Simulator (SWS) was an all-digital simulation used in developing the initial software, as well as verification. It ran at a 3.5/1 ratio of execution time to real time. The SWS was so effective that it once correctly identified a deficiency in the requirements relating to the Control Moment Gyro system. The Skylab Hybrid Simulator (SHS) included some analog circuits for greater fidelity. One of the most effective simulators was a System 360/44 connected to an actual ATMDC; the program in the 44 could simulate six degrees of freedom[59].

The verification process was scheduled for the final 10 weeks prior to the delivery of any software phase. The process included validation of the baseline program to the requirements, coding analysis, logic analysis, equation implementation tests, performance evaluations, and mission procedure validation. The AS-II did the logic analysis and was designed to trace all logic paths through the software. The 360/44 and ATMDC system did performance tests since it was near real time in operation[60]. The digital simulators could be stopped in order to insert program changes. Tracing was also possible[61]. Combining simulators and software verification tools contributed to a high level of confidence that was confirmed in actual performance.

USER INTERFACES

NASA and IBM designed the computer system to operate autonomously. One crewman reported "not much interaction" with the system at all[62], but the capability was present for significant activity if needed[63]. The crew could enter data and actually make changes in the software through a keyboard located in the DAS on the ATM Control and Display Console.

The DAS had only 10 keys and a three-position switch. The keys were the digits 0–7 (all entries were in octal), a clear key, and an enter key. The switch could select either power bus one or two, or be off. Above the DAS was an "Orbit Phase" panel containing a digital readout of minutes and seconds to the next orbital benchmark. When the first keystroke of a five-digit command was made, the uplink DCS commands were inhibited, and the time remaining clock inputs were inhibited, so that the clock digits could be used for displaying the keystrokes. In that mode, five digits would be lit instead of four. The remaining four keystrokes were the data/command inputs[64]. The display of the keystrokes represented an echo. If the sequence was correct, the astronaut pressed the enter key, or else he would restart the input process. Pressing the clear key brought back the digital clock. The rather limited nature of this command system indicates that it was intended for sparing use.

Besides the DAS, one other switch on the control panel related to the computer system. In the "Attitude Control" area of the panel was a three-position switch that controlled which computer was in actual use. It could be set for automatic (and usually was), in which case the redundancy management software would take care of matters. Alternately, the crew could purposely select either the primary or secondary computer. If either of these was selected, then automatic changeover was inhibited[65]. The switch gave the crew protection from

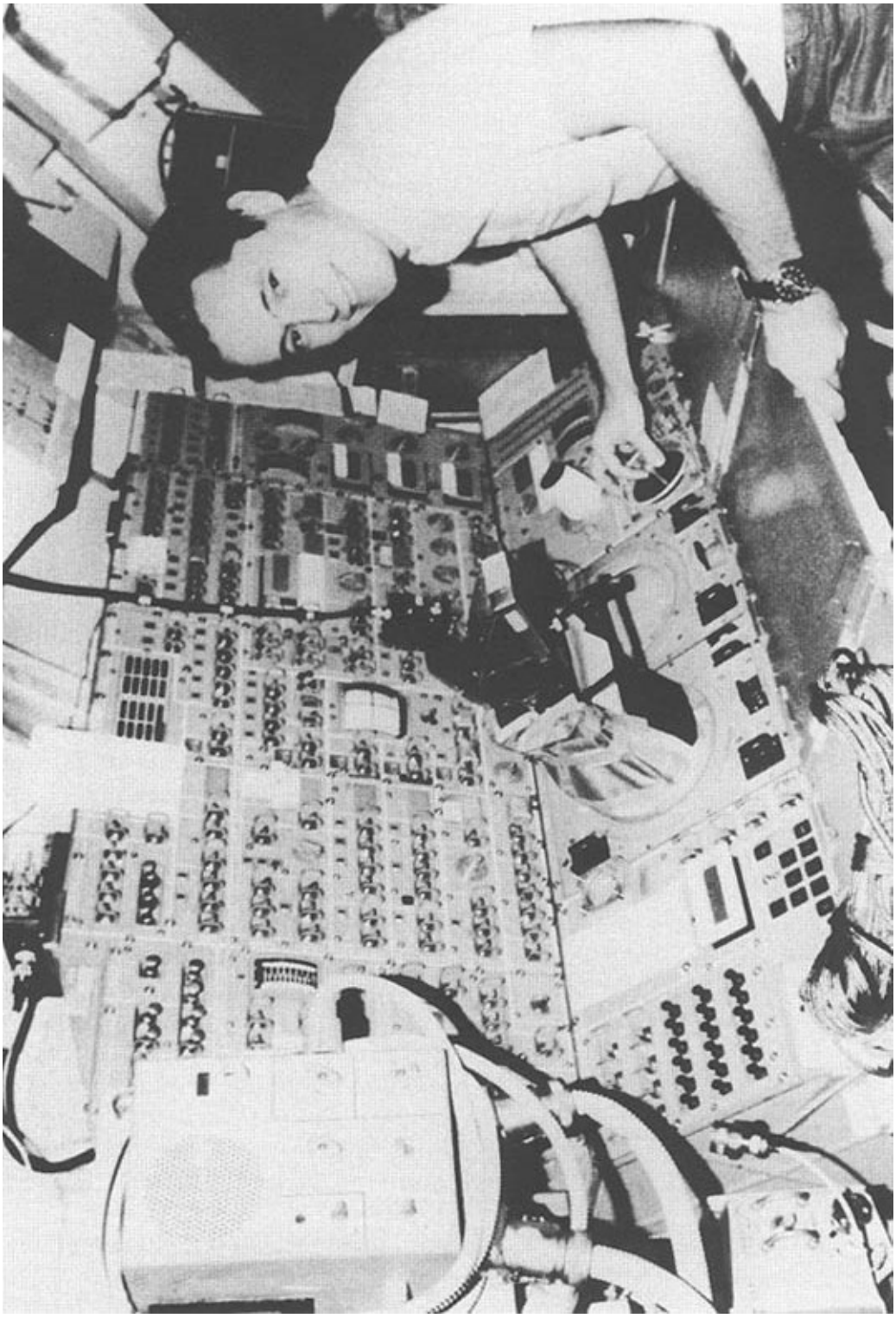

Figure 3–5. Dr. Edward Gibson at the Apollo Telescope Mount Control console. The interface to the digital computer is at lower left, on the panel immediately above the coil of cable. (NASA photo 4-60352)

failure of the redundancy management software. Incidentally, the

switch was not a common three-position toggle switch but, instead, required the crew to pull out and rotate the post. This protected the crew from accidental switching.

THE REACTIVATION MISSION

The Skylab Reactivation Mission represents one of the most interesting examples of the autonomy and reliability of manned spacecraft computers. The original Skylab mission lasted 272 days with long unmanned periods. The reactivation mission, flown entirely under computer control, lasted 393 days. Therefore, the bulk of the activated life of the space laboratory fully depended on the ATMDCs.

When it was obvious that the Workshop was going to fall to the earth long before a rescue mission could be launched, NASA began studying methods of prolonging the orbital life of the spacecraft. Even though the atmosphere is very thin at the altitude Skylab was flying, the drag produced on the spacecraft was highly related to its attitude with respect to its direction of flight (velocity vector). During most of the manned mission periods Skylab flew in solar inertial (SI) mode, in which the lab was kept perpendicular to the sun to provide maximum exposure for the solar collectors. Momentum desaturation maneuvers were done on the dark side of the earth to compensate for bias momentum buildup resulting from noncyclic torques acting on the spacecraft. The SI mode was high drag, so engineers devised two new modes, end-on-velocity-vector (EOVV) and torque equilibrium attitude (TEA). EOVV pointed the narrow end of the lab in the direction of flight, minimizing the aerodynamic drag on the vehicle. TEA could control the re-entry, using the gravity gradient and gyroscopic torques to counterbalance the aerodynamic torque. Only in this way could the Workshop be controlled below 140 nautical miles altitude[66].

Use of the new modes required that they be coded and transmitted to the computers in orbit. First it was necessary to discover whether or not the computers still functioned. Since the ATMDC used destructive readout core memories, there was some concern that the software might have been destroyed during restart tests if the refreshment hardware had failed. On March 6, 1978, NASA engineers at the Bermuda tracking station ordered portions of Skylab to activate. On March 11, the ATMDC powered up for 5 minutes to obtain telemetry confirmation that it was still functioning. The software resumed the program cycle where it had left off 4 years and 30 days earlier. As far as the computer was concerned, it had suffered a temporary power transient[67]!

When IBM began to make preparations to modify the software, it discovered that there was almost nothing with which to work. The carefully constructed tools used in the original software effort were

dispersed beyond recall, and, worse yet, the last of the source code for the flight programs had been deleted just weeks beforehand. This meant that changes to the software would have to be *hand coded* in hexadecimal, as the assembler could not be used—a risky venture in terms of ensuring accuracy. Eventually it became necessary to repunch the 2,516 cards of a listing of the most recent flight program, and IBM hired a subcontractor for the purpose[68].

Engineers could not test this software with the same high fidelity as during the original development. They abandoned plans for real time simulations because they could not find enough parts of any of the original simulators. Interpretive simulation could be performed because the tapes for that form of testing had been saved. However, the interpretive simulator ran 20 times slower than real time, so less testing was possible[69].

IBM approached the modification using the same principles as in the original production. The baseline software for the reactivation was Flight Program 80, including change request 3091, which was already in the second computer. Software changes for reactivation were simply handled as routine change requests. They placed the EOVV software in memory previously occupied by experiment calibration and other functions useless in the new mission. TEA replaced the command and display software[70].

When the software was ready for flight, NASA uplinked it to a reserve area of memory and then downlinked and manually verified it. If it passed the verification, engineers gave a command to activate it. The reprogramming was generally successful. The four people assigned to the software revision maintained IBM's record of quality throughout the reactivation mission[71].

CONCLUSIONS

The Skylab program demonstrated that careful management of software development, including strict control of changes, extensive and preplanned verification, and the use of adequate development tools, results in quality software with high reliability. Attention to piece part quality in hardware development and the use of redundancy resulted in reliable computers. However, it must be stressed that part of the success of the software management and the hardware development was due to the small size of both. Few programmers were involved in initial program design and writing. This meant that communications between programmers and teams were relatively minimal. The fact that IBM produced just 10 computers and really needed to ensure the success of just 2 of those helped in focusing the quality assurance effort expended on the hardware.

What happened after the manned Skylab program demonstrated the need for foresight and proper attention to storage of mission-critical materials until any possibility of their use had gone away. The dispersal of the verification hardware is understandable, as it is expensive to maintain. However, some provision should have been made for retaining mission-unique capabilities such as actual flight hardware. The destruction of the flight tapes and source code for the software by unknown parties was inexcusable. A single high-density disk pack could have held all relevant material.

Skylab marked the beginning of redundant computer hardware on manned spacecraft. It was also the first project that developed software with awareness of proper engineering principles. The Shuttle continued both these concepts but on a much larger and more complex scale.