WIKIPEDIA
The Free Encyclopedia

Apollo Guidance Computer

The **Apollo Guidance Computer** (**AGC**) was a digital computer produced for the Apollo program that was installed on board each Apollo command module (CM) and Apollo Lunar Module (LM). The AGC provided computation and electronic interfaces for guidance, navigation, and control of the spacecraft.[3] The AGC was the first computer based on silicon integrated circuits. The computer's performance was comparable to the first generation of home computers from the late 1970s, such as the Apple II, TRS-80, and Commodore PET.[4]

The AGC has a 16-bit word length, with 15 data bits and one parity bit. Most of the software on the AGC is stored in a special read-only memory known as core rope memory, fashioned by weaving wires through and around magnetic cores, though a small amount of read/write core memory is available.

Astronauts communicated with the AGC using a numeric display and keyboard called the DSKY (for "display and keyboard", pronounced "DIS-kee"). The AGC and its DSKY user interface were developed in the early 1960s for the Apollo program by the MIT Instrumentation Laboratory and first flew in 1966.[5]

Operation

Astronauts manually flew Project Gemini with control sticks, but computers flew most of Project Apollo except briefly during lunar landings.[6] Each Moon flight carried two AGCs, one each in the command module and the Apollo Lunar Module, with the exception of Apollo 8 which did not need a lunar module for its lunar orbit mission. The AGC in the command module was the center of its guidance, navigation and control (GNC) system. The AGC in the lunar module ran its Apollo PGNCS (primary guidance, navigation and control system), with the acronym pronounced as *pings*.

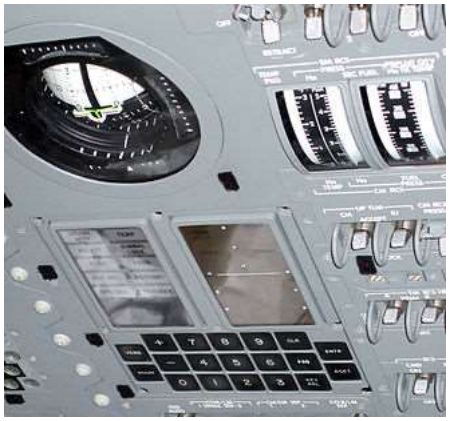
Each lunar mission had two additional computers:

- The Launch Vehicle Digital Computer (LVDC) on the Saturn V booster instrumentation ring
- the Abort Guidance System (AGS, pronounced *ags*) of the lunar module, to be used in the event of failure of the LM

Apollo Guidance Computer

Apollo Guidance Computer and DSKY

Invented by	Charles Stark Draper Laboratory
Manufacturer	Raytheon
Introduced	August 1966
Discontinued	July 1975
Type	Avionics Guidance computer
Processor	Discrete silicon integrated circuit (IC) chips (RTL based)
Frequency	2.048 MHz
Memory	15-bit wordlength + 1-bit parity 2048 words RAM (magnetic-core memory) 36,864 words ROM (core rope memory)[1]
Ports	DSKY, IMU, Hand Controller, Rendezvous Radar (CM), Landing Radar (LM), Telemetry Receiver, Engine Command,



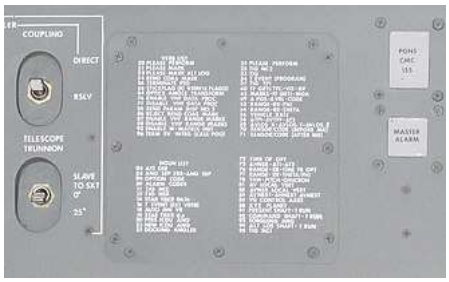
The display and keyboard (DSKY) interface of the Apollo Guidance Computer mounted on the control panel of the command module, with the flight director attitude indicator (FDAI) above

PGNCS. The AGS could be used to take off from the Moon, and to rendezvous with the command module, but not to land.

	Reaction Control System
Power consumption	55 W[2]:120
Language	AGC Assembly Language
Weight	70 lb (32 kg)
Dimensions	24 in × 12.5 in × 6.5 in (61 cm × 32 cm × 17 cm)

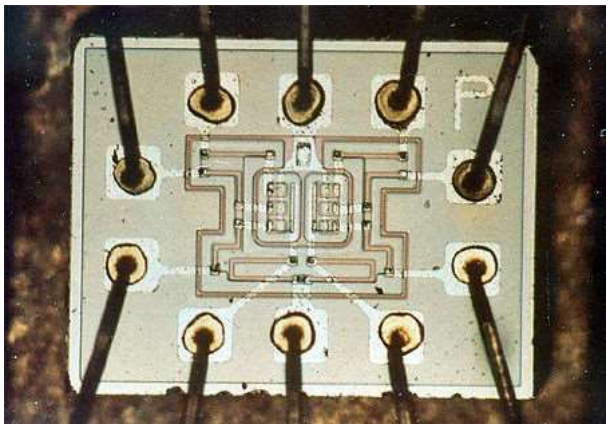
Design

The AGC was designed at the MIT Instrumentation Laboratory under Charles Stark Draper, with hardware design led by Eldon C. Hall.[2] Early architectural work came from J. H. Laning Jr., Albert Hopkins, Richard Battin, Ramon Alonso,[7] [8] and Hugh Blair-Smith.[9] The flight hardware was fabricated by Raytheon, whose Herb Thaler[10] was also on the architectural team.



Partial list of numeric codes for verbs and nouns in the Apollo Guidance Computer, printed for quick reference on a side panel

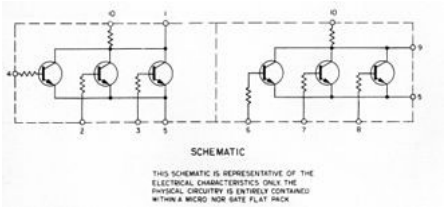
Logic hardware



Photograph of the dual NOR gate chip used to build the Block II Apollo Guidance Computer. Connections (clockwise from top center) ground, inputs (3), output, power, output, inputs (3).

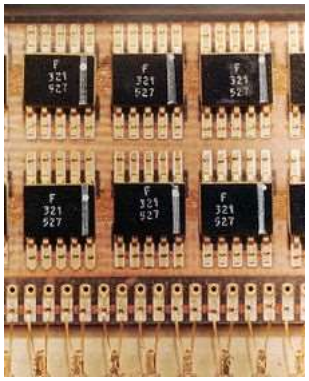
Following the use of integrated circuit (IC) chips in the Interplanetary Monitoring Platform (IMP) in 1963, IC technology was later adopted for the AGC.[11] The Apollo flight computer was the first computer to use silicon IC chips.[12]

While the Block I version used 4,100 ICs, each containing a single three-input NOR gate, the later Block II version (used in the crewed flights) used about 2,800 ICs, mostly dual three-input NOR gates and smaller numbers of expanders and sense amplifiers.[13]:27,266 The ICs, from Fairchild Semiconductor, were implemented using resistor–transistor logic (RTL) in a flat-pack. They were connected via wire wrap, and the wiring was then embedded in cast epoxy plastic.[13]:129



AGC dual 3-input NOR gate schematic

The use of a single type of IC (the dual NOR3) throughout the AGC avoided problems that plagued another early IC computer design, the Minuteman II guidance computer, which used a mix of diode–transistor logic and diode logic gates. NOR gates are universal logic gates from which any other gate can be made, though at the cost of using more gates.[14]



Flatpack silicon integrated circuits in the Apollo guidance computer

Memory

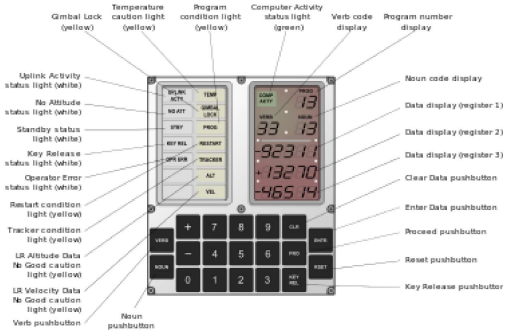
The computer had 2048 words of erasable magnetic-core memory and 36,864 words of read-only core rope memory.[13]:27,90–93 Both had cycle times of 11.72 microseconds.[13]:27 The memory word length was 16 bits: 15 bits of data and one odd-parity bit. The CPU-internal 16-bit word format was 14 bits of data, one overflow bit, and one sign bit (ones' complement representation).[13]:35–37

DSKY interface

The user interface to the AGC was the *DSKY*, standing for *display and keyboard* and usually pronounced *"DIS-kee"*. It has an array of indicator lights, numeric displays, and a calculator-style keyboard. Commands were entered numerically, as two-digit numbers: Verb, and Noun. *Verb* described the type of action to be performed and *Noun* specified which data were affected by the action specified by the Verb command.

Apollo computer DSKY user interface unit

Each digit was displayed via a green (specified as 530 nm[15]) high-voltage electroluminescent seven-segment display; these were driven by electromechanical relays, limiting the update rate. Three five-digit signed numbers could also be displayed in octal or decimal, and were typically used to display vectors such as space craft attitude or a required velocity change (delta-V). Although data was stored internally in metric units, they were displayed as United States customary units. This calculator-style interface was the first of its kind.



LM DSKY interface diagram

The command module has two DSKYs connected to its AGC: one located on the main instrument panel and a second located in the lower equipment bay near a sextant used for aligning the inertial guidance platform. The lunar module had a single DSKY for its AGC. A flight director attitude indicator (FDAI), controlled by the AGC, was located above the DSKY on the commander's console and on the LM.

Timing

The AGC timing reference came from a 2.048 MHz crystal clock. The clock was divided by two to produce a four-phase 1.024 MHz clock which the AGC used to perform internal operations. The 1.024 MHz clock was also divided by two to produce a 512 kHz signal called the *master frequency*; this signal was used to synchronize external Apollo spacecraft systems.

The master frequency was further divided through a *scaler*, first by five using a ring counter to produce a 102.4 kHz signal. This was then divided by two through 17 successive stages called F1 (51.2 kHz) through F17 (0.78125 Hz). The F10 stage (100 Hz) was fed back into the AGC to increment the real-time clock and other involuntary counters using Pinc (discussed below). The F17 stage was used to intermittently run the AGC when it was operating in the *standby* mode.

Central registers

The AGC had four 16-bit registers for general computational use, called the *central registers*:

- **A**: The accumulator, for general computation
- **Z**: The program counter – the address of the next instruction to be executed
- **Q**: The remainder from the `DV` instruction, and the return address after `TC` instructions
- **LP**: The lower product after `MP` instructions

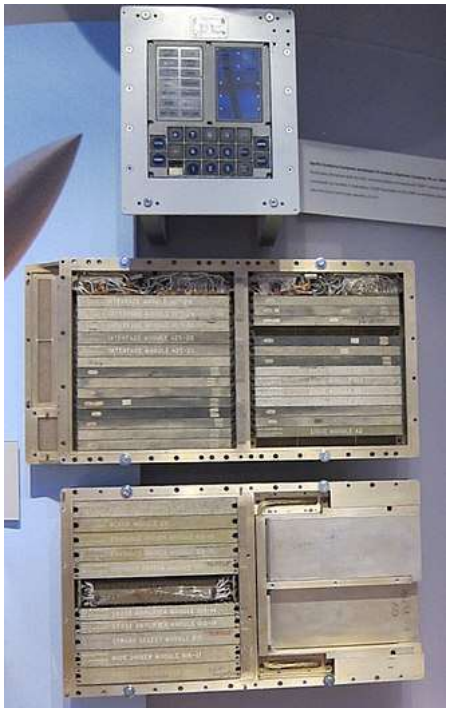
There were also four locations in core memory, at addresses 20–23, dubbed *editing locations* because whatever was stored there would emerge shifted or rotated by one bit position, except for one that shifted right seven bit positions, to extract one of the seven-bit interpretive op. codes that were packed two to a word. This was common to Block I and Block II AGCs.

Other registers

The AGC had additional registers that were used internally in the course of operation:

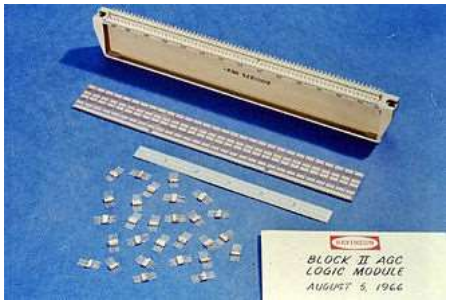
- **S**: 12-bit memory address register, the lower portion of the memory address
- **Bank/Fbank**: 4-bit ROM bank register, to select the 1 kiloword ROM bank when addressing in the fixed-switchable mode
- **Ebank**: 3-bit RAM bank register, to select the 256-word RAM bank when addressing in the erasable-switchable mode
- **Sbank** (super-bank): 1-bit extension to Fbank, required because the last 4 kilowords of the 36-kiloword ROM was not reachable using Fbank alone
- **SQ**: 4-bit sequence register; the current instruction
- **G**: 16-bit memory buffer register, to hold data words moving to and from memory
- **X**: The 'x' input to the *adder* (the adder was used to perform all 1's complement arithmetic) or the increment to the program counter (**Z** register)
- **Y**: The other ('y') input to the adder
- **U**: Not really a register, but the output of the adder (the one's complement sum of the contents of registers **X** and **Y**)
- **B**: General-purpose buffer register, also used to pre-fetch the next instruction. At the start of the next instruction, the upper bits of **B** (containing the next op. code) were copied to **SQ**, and the lower bits (the address) were copied to **S**.
- **C**: Not a separate register, but the one's complement of the **B** register
- **IN**: Four 16-bit input registers

- **OUT**: Five 16-bit output registers



DSKY and AGC prototypes on display at the Computer History Museum. The AGC is opened up, showing its logic modules.

Prototype logic module from Block I AGC



Block II logic module, with flat-pack ICs

Instruction set

The instruction format used 3 bits for opcode, and 12 bits for address. Block I had 11 instructions: `TC`, `CCS`, `INDEX`, `XCH`, `CS`, `TS`, `AD`, and `MASK` (basic), and `SU`, `MP`, and `DV` (extra). The first eight, called *basic instructions*, were directly accessed by the 3-bit op. code. The final three were denoted as *extracode instructions* because they were accessed by performing a special type of `TC` instruction (called `EXTEND`) immediately before the instruction.

The Block I AGC instructions consisted of the following:

`TC` (transfer control)
: An unconditional branch to the address specified by the instruction. The return address was automatically stored in the Q register, so the `TC` instruction could be used for subroutine calls.

`CCS` (count, compare, and skip)
: A complex conditional branch instruction. The A register was loaded with data retrieved from the address specified by the instruction. (Because the AGC uses ones' complement notation, there are two representations of zero. When all bits are set to zero, this is called *plus zero*. If all bits are set to one, this is called *minus zero*.) The *diminished absolute value* (DABS) of the data was then computed and stored in the A register. If the number was greater than zero, the DABS decrements the value by 1; if the number was negative, it is complemented before the decrement is applied—this is the absolute value. *Diminished* means "decremented but not below zero". Therefore, when the AGC performs the DABS function, positive numbers will head toward plus zero, and so will negative numbers but first revealing their negativity via the four-way skip below. The final step in `CCS` is a four-way skip, depending upon the data in register A before the DABS. If register A was greater than 0, `CCS` skips to the first instruction immediately after `CCS`. If register A contained plus zero, `CCS` skips to the second instruction after `CCS`. Less than zero causes a skip to the third instruction after `CCS`, and minus zero skips to the fourth instruction after `CCS`. The primary purpose of the count was to allow an ordinary loop, controlled by a positive counter, to end in a `CCS` and a `TC` to the beginning of the loop, equivalent to an IBM 360's `BCT`. The absolute value function was deemed important enough to be built into this instruction; when used for only this purpose, the sequence after the `CCS` was `TC` *+2, `TC` *+2, `AD` ONE. A curious side effect was the creation and use of *CCS-holes* when the value being tested was known to be never positive, which occurred more often than one might suppose. That left two whole words unoccupied, and a special committee was responsible for assigning data constants to these holes.

INDEX
: Add the data retrieved at the address specified by the instruction to the next instruction. INDEX can be used to add or subtract an index value to the base address specified by the operand of the instruction that follows INDEX. This method is used to implement arrays and table look-ups; since the addition was done on both whole words, it was also used to modify the op. code in a following (extracode) instruction, and on rare occasions both functions at once.

RESUME
: A special instance of INDEX (INDEX 25). This is the instruction used to return from interrupts. It causes execution to resume at the interrupted location.

XCH (exchange)
: Exchange the contents of memory with the contents of the A register. If the specified memory address is in fixed (read-only) memory, the memory contents are not affected, and this instruction simply loads register A. If it is in erasable memory, overflow "correction" is achieved by storing the leftmost of the 16 bits in A as the sign bit in memory, but there is no exceptional behavior like that of TS.

CS (clear and subtract)
: Load register A with the one's complement of the data referenced by the specified memory address.

TS (transfer to storage)
: Store register A at the specified memory address. TS also detects, and corrects for, overflows in such a way as to propagate a carry for multi-precision add/subtract. If the result has no overflow (leftmost 2 bits of A the same), nothing special happens; if there is overflow (those 2 bits differ), the leftmost one goes the memory as the sign bit, register A is changed to +1 or −1 accordingly, and control skips to the second instruction following the TS. Whenever overflow is a possible but abnormal event, the TS was followed by a TC to the no-overflow logic; when it is a normal possibility (as in multi-precision add/subtract), the TS is followed by CAF ZERO (CAF = XCH to fixed memory) to complete the formation of the carry (+1, 0, or −1) into the next higher-precision word. Angles were kept in single precision, distances and velocities in double precision, and elapsed time in triple precision.

AD (add)
: Add the contents of memory to register A and store the result in A. The 2 leftmost bits of A may be different (overflow state) before and/or after the AD. The fact that overflow is a state rather than an event forgives limited extents of overflow when adding more than two numbers, as long as none of the intermediate totals exceed twice the capacity of a word.

MASK
: Perform a bit-wise (boolean) *and* of memory with register A and store the result in register A.

MP (multiply)
: Multiply the contents of register A by the data at the referenced memory address and store the high-order product in register A and the low-order product in register LP. The parts of the product agree in sign.

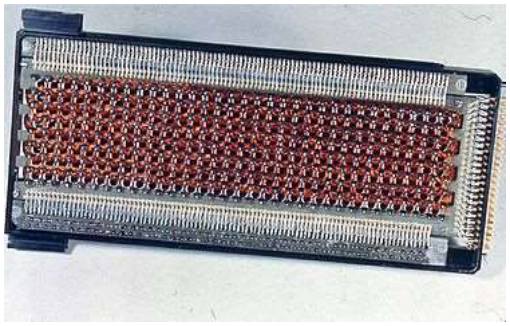
DV (divide)
: Divide the contents of register A by the data at the referenced memory address. Store the quotient in register A and the absolute value of the remainder in register Q. Unlike modern machines, fixed-point numbers were treated as fractions (notional decimal point just to right of the sign bit), so you could produce garbage if the divisor was not larger than the dividend; there was no protection against that situation. In the Block II AGC, a double-precision dividend started in A and L (the Block II LP), and the correctly signed remainder was delivered in L. That considerably simplified the subroutine for double precision division.

SU (subtract)
: Subtract (one's complement) the data at the referenced memory address from the contents of register A and store the result in A.

Instructions were implemented in groups of 12 steps, called *timing pulses*. The timing pulses were named TP1 through TP12. Each set of 12 timing pulses was called an instruction *subsequence*. Simple instructions, such as TC, executed in a single subsequence of 12 pulses. More complex instructions required several subsequences. The multiply instruction (`MP`) used 8 subsequences: an initial one called `MP0`, followed by an `MP1` subsequence which was repeated 6 times, and then terminated by an `MP3` subsequence. This was reduced to 3 subsequences in Block II.

Each timing pulse in a subsequence could trigger up to 5 *control pulses*. The control pulses were the signals which did the actual work of the instruction, such as reading the contents of a register onto the bus, or writing data from the bus into a register.

Memory



AGC core rope memory (ROM)

Block I AGC memory was organized into 1 kiloword banks. The lowest bank (bank 0) was erasable memory (RAM). All banks above bank 0 were fixed memory (ROM). Each AGC instruction had a 12-bit address field. The lower bits (1-10) addressed the memory inside each bank. Bits 11 and 12 selected the bank: 00 selected the erasable memory bank; 01 selected the lowest bank (bank 1) of fixed memory; 10 selected the next one (bank 2); and 11 selected the *Bank* register that could be used to select any bank above 2. Banks 1 and 2 were called *fixed-fixed* memory, because they were always available, regardless of the contents of the Bank register. Banks 3 and above were called *fixed-switchable* because the selected bank was determined by the bank register.

The Block I AGC initially had 12 kilowords of fixed memory, but this was later increased to 24 kilowords. Block II had 36 kilowords of fixed memory and 2 kilowords of erasable memory.

The AGC transferred data to and from memory through the G register in a process called the *memory cycle*. The memory cycle took 12 timing pulses (11.72 μs). The cycle began at timing pulse 1 (TP1) when the AGC loaded the memory address to be fetched into the S register. The memory hardware retrieved the data word from memory at the address specified by the S register. Words from erasable memory were deposited into the G register by timing pulse 6 (TP6); words from fixed memory were available by timing pulse 7. The retrieved memory word was then available in the G register for AGC access during timing pulses 7 through 10. After timing pulse 10, the data in the G register was written back to memory.

The AGC memory cycle occurred continuously during AGC operation. Instructions needing memory data had to access it during timing pulses 7–10. If the AGC changed the memory word in the G register, the changed word was written back to memory after timing pulse 10. In this way, data words cycled continuously from memory to the G register and then back again to memory.

The lower 15 bits of each memory word held AGC instructions or data, with each word being protected by a 16th odd parity bit. This bit was set to 1 or 0 by a parity generator circuit so a count of the 1s in each memory word would always produce an odd number. A parity checking circuit tested the parity bit during each memory cycle; if the bit didn't match the expected value, the memory word was assumed to be corrupted and a *parity alarm* panel light was illuminated.

Interrupts and involuntary counters

The AGC had five vectored interrupts:

- *Dsrupt* was triggered at regular intervals to update the user display (DSKY).
- *Erupt* was generated by various hardware failures or alarms.
- *Keyrupt* signaled a key press from the user's keyboard.
- *T3Rrupt* was generated at regular intervals from a hardware timer to update the AGC's real-time clock.
- *Uprupt* was generated each time a 16-bit word of uplink data was loaded into the AGC.

The AGC responded to each interrupt by temporarily suspending the current program, executing a short interrupt service routine, and then resuming the interrupted program.

The AGC also had 20 involuntary counters. These were memory locations which functioned as up/down counters, or shift registers. The counters would increment, decrement, or shift in response to internal inputs. The increment (*Pinc*), decrement (*Minc*), or shift (*Shinc*) was handled by one subsequence of microinstructions inserted between any two regular instructions.

Interrupts could be triggered when the counters overflowed. The T3rupt and Dsrupt interrupts were produced when their counters, driven by a 100 Hz hardware clock, overflowed after executing many Pinc subsequences. The Uprupt interrupt was triggered after its counter, executing the Shinc subsequence, had shifted 16 bits of uplink data into the AGC.

Standby mode

The AGC had a power-saving mode controlled by a *standby allowed* switch. This mode turned off the AGC power, except for the 2.048 MHz clock and the scaler. The F17 signal from the scaler turned the AGC power and the AGC back on at 1.28 second intervals. In this mode, the AGC performed essential functions, checked the standby allowed switch, and, if still enabled, turned off the power and went back to sleep until the next F17 signal.

In the standby mode, the AGC slept most of the time; therefore it was not awake to perform the Pinc instruction needed to update the AGC's real time clock at 10 ms intervals. To compensate, one of the functions performed by the AGC each time it awoke in the standby mode was to update the real time clock by 1.28 seconds.

The standby mode was designed to reduce power by 5 to 10 W (from 70 W) during midcourse flight when the AGC was not needed. However, in practice, the AGC was left on during all phases of the mission and this feature was never used.

Data buses

The AGC had a 16-bit read bus and a 16-bit write bus. Data from central registers (A, Q, Z, or LP), or other internal registers could be gated onto the read bus with a control signal. The read bus connected to the write bus through a non-inverting buffer, so any data appearing on the read bus also appeared on the write bus. Other control signals could copy write bus data back into the registers.

Data transfers worked like this: To move the address of the next instruction from the B register to the S register, an RB (read B) control signal was issued; this caused the address to move from register B to the read bus, and then to the write bus. A WS (write S) control signal moved the address from the write bus into the S register.

Several registers could be read onto the read bus simultaneously. When this occurred, data from each register was inclusive-*OR*ed onto the bus. This inclusive-*OR* feature was used to implement the Mask instruction, which was a logical *AND* operation. Because the AGC had no native ability to do a logical *AND*, but could do a logical *OR* through the bus and could complement (invert) data through the C register, De Morgan's theorem was used to implement the equivalent of a logical *AND*. This was accomplished by inverting both operands, performing a logical *OR* through the bus, and then inverting the result.

Software

AGC software was written in AGC assembly language and stored on rope memory. The bulk of the software was on read-only rope memory and thus could not be changed in operation,[17] but some key parts of the software were stored in standard read-write magnetic-core memory and could be overwritten by the astronauts using the DSKY interface, as was done on Apollo 14.

Margaret Hamilton standing next to listings of the software she and her MIT team produced for the Apollo Project.[16]

A simple real-time operating system designed by J. Halcombe Laning[18] consisting of the 'Exec', a batch job-scheduling using cooperative multi-tasking,[19] and an interrupt-driven pre-emptive scheduler called the 'Waitlist' which scheduled timer-driven 'tasks', controlled the computer. Tasks were short threads of execution which could reschedule themselves for re-execution on the Waitlist, or could kick off a longer operation by starting a 'job' with the Exec. Calculations were carried out using the metric system, but display readouts were in units of feet, feet per second, and nautical miles – units that the Apollo astronauts were accustomed to.[20]

The AGC had a sophisticated software interpreter, developed by the MIT Instrumentation Laboratory, that implemented a virtual machine with more complex and capable pseudo-instructions than the native AGC. These instructions simplified the navigational programs. Interpreted code, which featured double precision trigonometric, scalar and vector arithmetic (16 and 24-bit), even an `MXV` (matrix × vector) instruction, could be mixed with native AGC code. While the execution time of the pseudo-instructions was increased (due to the need to interpret these instructions at runtime) the interpreter provided many more instructions than AGC natively supported and the memory requirements were much lower than in the case of adding these instructions to the AGC native language which would require additional memory built into the computer (at that time the memory capacity was very expensive). The average pseudo-instruction required about 24 ms to execute. The assembler, named *YUL* for an early prototype *Christmas Computer*,[21] enforced proper transitions between native and interpreted code.

A set of interrupt-driven user interface routines called 'Pinball' provided keyboard and display services for the jobs and tasks running on the AGC. A set of user-accessible routines were provided to let the astronauts display the contents of various memory locations in octal or decimal in groups of 1, 2, or 3 registers at a time. 'Monitor' routines were provided so the operator could initiate a task to periodically redisplay the contents of certain memory locations. Jobs could be initiated.

The design principles developed for the AGC by MIT Instrumentation Laboratory, directed in late 1960s by Charles Draper, became foundational to software engineering—particularly for the design of more reliable systems that relied on asynchronous software, priority scheduling, testing, and human-in-the-loop decision capability.[22] When the design requirements for the AGC were defined, necessary software and programming techniques did not exist so they had to be designed from scratch. Many of the trajectory and guidance algorithms used were based on earlier work by Richard Battin.[18] The first command module flight was controlled by a software package called CORONA whose development was led by Alex Kosmala. Software for lunar missions consisted of COLOSSUS for the command module, whose development was led by Frederic Martin, and LUMINARY on the lunar module led by George Cherry. Details of these programs were implemented by a team under the direction of Margaret Hamilton.[23] Hamilton was very interested in how the astronauts would interact with the software and predicted the types of errors that could occur due to human error.[19][23] In total, software development on the project comprised 1400 person-years of effort, with a peak workforce of 350 people.[18] In 2016, Hamilton received the Presidential Medal of Freedom for her role in creating the flight software.

The Apollo Guidance Computer software influenced the design of Skylab, Space Shuttle and early fly-by-wire fighter aircraft systems.[24][25]

The Apollo Guidance computer has been called "The fourth astronaut" for its role in helping the three astronauts who relied on it Neil Armstrong, Buzz Aldrin and Michael Collins.[26]

Block II

A Block II version of the AGC was designed in 1966. It retained the basic Block I architecture, but increased erasable memory from 1 to 2 kilowords. Fixed memory was expanded from 24 to 36 kilowords. Instructions were expanded from 11 to 34 and I/O channels were implemented to replace the I/O registers on Block I. The Block II version is the one that actually flew to the moon. Block I was used during the uncrewed Apollo 4 and 6 flights, and was on board the ill-fated Apollo 1.

The decision to expand the memory and instruction set for Block II, but to retain the Block I's restrictive three-bit op. code and 12-bit address had interesting design consequences. Various tricks were employed to squeeze in additional instructions, such as having special memory addresses which, when referenced, would implement a certain function. For instance, an `INDEX` to address 25 triggered the `RESUME` instruction to return from an interrupt. Likewise, `INDEX` 17 performed an `INHINT` instruction (inhibit interrupts), while `INDEX` 16 reenabled them (`RELINT`). Other instructions were implemented by preceding them with a special version of `TC` called `EXTEND`. The address spaces were extended by employing the Bank (fixed) and Ebank (erasable) registers, so the only memory of either type that could be addressed at any given time was the current bank, plus the small amount of fixed-fixed memory and the erasable memory. In addition, the bank register could address a maximum of 32 kilowords, so an Sbank (super-bank) register was required to access the last 4 kilowords. All across-bank subroutine calls had to be initiated from fixed-fixed memory through special functions to restore the original bank during the return: essentially a system of far pointers.

The Block II AGC also has the EDRUPT instruction (the name is a contraction of *Ed's Interrupt*, after Ed Smally, the programmer who requested it). This instruction does not generate an interrupt, rather it performs two actions that are common to interrupt processing. The first action, inhibits further interrupts (and requires a RESUME instruction to enable them again). In the second action, the ZRUPT register is loaded with the current value of the program counter (Z). It was only used once in the Apollo software, for setting up the DAP cycle termination sequence in the Digital Autopilot of the lunar module.[27] It is believed to be responsible for problems emulating the LEM AGC Luminary software.

1201 and 1202 program alarms

PGNCS generated unanticipated warnings during Apollo 11's lunar descent, with the AGC showing a *1202 alarm* ("Executive overflow - NO CORE SETS"),[28] and then a *1201 alarm* ("Executive overflow - NO VAC AREAS").[29] The response of the AGC to either alarm was a soft restart. The cause was a rapid, steady stream of spurious cycle steals from the rendezvous radar (tracking the orbiting command module), intentionally left on standby during the descent in case it was needed for an abort.[30][31]

DSKY and Buzz Aldrin on the Apollo 11 Lunar Module *Eagle* en route to the Moon

During this part of the approach, the processor would normally be almost 85% loaded. The extra 6,400 cycle steals per second added the equivalent of 13% load, leaving just enough time for all scheduled tasks to run to completion. Five minutes into the descent, Buzz Aldrin gave the computer the command *1668*, which instructed it to periodically calculate and display DELTAH (the difference between altitude sensed by the radar and the computed altitude).[nb 1] The *1668* added another 10% to the processor workload, causing executive overflow and a *1202* alarm. After being given the "GO" from Houston, Aldrin entered *1668* again and another *1202* alarm occurred. When reporting the second alarm, Aldrin added the comment "It appears to come up when we have a *1668* up". The AGC software had been designed with priority scheduling, and automatically recovered, deleting lower priority tasks including the *1668* display task, to complete its critical guidance and control tasks. Guidance controller Steve Bales and his support team that included Jack Garman issued several "GO" calls and the landing was successful. For his role, Bales received the US Presidential Medal of Freedom on behalf of the entire control center team and the three Apollo astronauts.[32]

The problem was not a programming error in the AGC, nor was it pilot error. It was a peripheral hardware design bug that had already been known and documented by Apollo 5 engineers.[33] However, because the problem had only occurred once during testing, they concluded that it was safer to fly with the existing hardware that they had already tested, than to fly with a newer but largely untested radar system. In the actual hardware, the position of the rendezvous radar was encoded with synchros excited by a different source of 800 Hz AC than the one used by the computer as a timing reference. The two 800 Hz sources were frequency locked but not phase locked, and the small random phase variations made it appear as though the antenna was rapidly "dithering" in position, even though it was completely stationary. These phantom movements generated the rapid series of cycle steals.

J. Halcombe Laning's software and computer design saved the Apollo 11 landing mission. Had it not been for Laning's design, the landing would have been aborted for lack of a stable guidance computer.[34][35]

Applications outside Apollo

The AGC formed the basis of an experimental fly-by-wire (FBW) system installed into an F-8 Crusader to demonstrate the practicality of computer driven FBW. The AGC used in the first phase of the program was replaced with another machine in the second phase, and research done on the program led to the development of fly-by-wire systems for the Space Shuttle. The AGC also led, albeit indirectly, to the development of fly-by-wire systems for the generation of fighters that were being developed at the time.[36]

Fly By Wire testbed aircraft. The AGC DSKY is visible in the avionics bay

Source code release

In 2003, an effort was started by Ron Burkey to recover the source code that powered the AGC and build an emulator able to run it, the VirtualAGC.[37][38] Part of the large amount of source code rescued as a result of this effort was uploaded by a former NASA intern to GitHub on July 7, 2016, attracting significant media attention.[39][40] The original Apollo 11 Guidance Computer source code was originally made accessible in 2003[41] by the Virtual AGC Project (http://ibiblio.org/apollo/index.html) and MIT Museum.[42] It was transcribed and digitalized from the original hard-copy source code listings that were made in the 60s. In mid 2016, former NASA intern, Chris Garry, uploaded the AGC Source code onto GitHub.[43][44]

See also

- Apollo PGNCS - the Apollo Primary Guidance and Navigation System
- AP-101 (IBM S/360-derived) computers used in the Space Shuttle
- Gemini Guidance Computer
- History of computer hardware

Notes

1. More specifically, verb 16 instructs the AGC to print the *noun* (in this case, 68, DELTAH) approximately twice per second. Had Aldrin known this, a simple *0668* (calculate and display DELTAH, once) would have only added approximately 5% load to the system, and would have only done so once, when ENTER was pressed.

References

1. *Programmer's Manual, Block 2 AGC Assembly Language* (https://www.ibiblio.org/apollo/assembly_language_manual.html), retrieved 2018-08-27
2. Hall, Eldon C. (1996), *Journey to the Moon: The History of the Apollo Guidance Computer*, Reston, Virginia, USA: AIAA, p. 196, ISBN 1-56347-185-X

3. Interbartolo, Michael (January 2009). "Apollo Guidance, Navigation and Control Hardware Overview" (https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20090016290.pdf) (PDF).
4. "How did the Apollo flight computers get men to the moon and back ?" (http://curious-droid.com/214/apollo-flight-computers-get-men-moon-back/). 11 March 2017.
5. NASA.gov (https://history.nasa.gov/computers/Ch2-5.html)
6. Agle, D.C. (September 1998). "Flying the Gusmobile" (https://www.airspacemag.com/flight-today/flying-the-gusmobile-218187/). *Air & Space*. Retrieved 2018-12-15.
7. "Ramon Alonso's introduction" (http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/conference1/alonso-intro.htm), *AGC History Project (Caltech archive, original site closed)*, MIT, July 27, 2001, retrieved 2009-08-30
8. "Ramon Alonso's interview (Spanish)" (http://www.lanacion.com.ar/nota.asp?nota_id=1240769), *Ramón Alonso, el argentino que llevó a la Apollo 11 a la Luna*, Diario La Nacion, March 7, 2010
9. "Hugh Blair-Smith biography" (http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/people/hblairsmith.htm), *AGC History Project (Caltech archive, original site closed)*, MIT, January 2002, retrieved 2009-08-30
10. "Herb Thaler introduction" (http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/conference2/thaler-intro.htm), *AGC History Project (Caltech archive, original site closed)*, MIT, 14 September 2001, retrieved 2009-08-30
11. Butrica, Andrew J. (2015). "Chapter 3: NASA's Role in the Manufacture of Integrated Circuits". In Dick, Steven J. (ed.). *Historical Studies in the Societal Impact of Spaceflight* (https://www.nasa.gov/sites/default/files/atoms/files/historical-studies-societal-impact-spaceflight-ebook_tagged.pdf) (PDF). NASA. pp. 149–250. ISBN 978-1-62683-027-1.
12. "Apollo Guidance Computer and the First Silicon Chips" (https://airandspace.si.edu/stories/editorial/apollo-guidance-computer-and-first-silicon-chips). *National Air and Space Museum*. Smithsonian Institution. 14 October 2015. Retrieved 1 September 2019.
13. Hall, Eldon C. (1972). *MIT's Role in Project Apollo: Final report on contracts NAS 9-163 and NAS 94065* (http://ibiblio.org/apollo/hrst/archive/1029.pdf) (PDF). Cambridge, MA: MIT. Retrieved 15 June 2021.
14. Peirce, C. S. (manuscript winter of 1880–81), "A Boolian Algebra with One Constant", published 1933 in *Collected Papers* v. 4, paragraphs 12–20. Reprinted 1989 in *Writings of Charles S. Peirce* v. 4, pp. 218–21, Google [1] (https://archive.org/details/writingsofcharle0002peir/page/218). See Roberts, Don D. (2009), *The Existential Graphs of Charles S. Peirce*, p. 131.
15. "Apollo DSKY panel relight: The full story" (https://www.youtube.com/watch?v=feRCZyLzAwA?t=1405). *YouTube*.
16. Weinstock, Maia (2016-08-17). "Scene at MIT: Margaret Hamilton's Apollo code" (https://news.mit.edu/2016/scene-at-mit-margaret-hamilton-apollo-code-0817). *MIT News*. Retrieved 2016-08-17.
17. Mindell 2008, pp. 154, 157.
18. Hoag, David (September 1976). "The History of Apollo On-board Guidance, Navigation, and Control" (http://klabs.org/history/history_docs/mit_docs/1711.pdf) (PDF). Charles Stark Draper Laboratory.
19. Mindell 2008, p. 149.
20. "The Moon landings" (https://ukma.org.uk/why-metric/myths/metric-internationally/the-moon-landings/). UK Metric Association. 18 October 2018.
21. "Hugh Blair-Smith's Introduction" (http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/conference3/blairsmith.htm), *AGC History Project (Caltech archive, original site closed)*, MIT, 30 November 2001, retrieved 2010-03-21
22. NASA Press Release "NASA Honors Apollo Engineer" (https://history.nasa.gov/alsj/a11/a11Hamilton.html) (September 03, 2003)

23. Harvey IV, Harry Gould (13 October 2015). "Her Code Got Humans on the Moon—And Invented Software Itself" (https://www.wired.com/2015/10/margaret-hamilton-nasa-apollo/). *WIRED*. Retrieved 2018-11-25.
24. NASA Office of Logic Design "About Margaret Hamilton" (http://klabs.org/home_page/hamilton.htm) (Last Revised: February 03, 2010)
25. By *A.J.S. Rayl* "NASA Engineers and Scientists-Transforming Dreams Into Reality" (http://www.nasa.gov/50th/50th_magazine/scientists.html)
26. Fong, Kevin (2019). "13 minutes to the moon: Episode 5 The fourth astronaut" (https://www.bbc.co.uk/programmes/w3csz4dn). *bbc.co.uk*. BBC World Service.
27. O'Brien, Frank (2010-06-25). *The Apollo Guidance Computer: Architecture and Operation* (https://books.google.com/books?id=3fKzL0HfJp4C&q=edrupt+instruction&pg=PA95). Springer Science & Business Media. ISBN 978-1-4419-0877-3.
28. Collins, Michael; Aldrin, Edwin (1975), Cortright, Edgar M. (ed.), "A Yellow Caution Light" (https://history.nasa.gov/SP-350/ch-11-4.html), *NASA SP-350, Apollo Expeditions to the Moon*, Washington, DC: NASA, pp. Chapter 11.4, ISBN 978-0486471754, retrieved 2009-08-30
29. "chrislgarry/Apollo-11" (https://github.com/chrislgarry/Apollo-11/blob/master/Comanche055/EXECUTIVE.agc#L146). *GitHub*. Retrieved 2016-07-17.
30. Adler, Peter (1998), Jones, Eric M. (ed.), "Apollo 11 Program Alarms" (http://www.hq.nasa.gov/office/pao/History/alsj/a11/a11.1201-pa.html), *Apollo 11 Lunar Surface Journal*, NASA, retrieved 2009-09-01
31. Martin, Fred H. (July 1994), Jones, Eric M. (ed.), "Apollo 11 : 25 Years Later" (http://www.hq.nasa.gov/alsj/a11/a11.1201-fm.html), *Apollo 11 Lunar Surface Journal*, NASA, retrieved 2009-09-01
32. Cortright, Edgar M., ed. (1975), "The Lunar Module Computer" (http://www.abc.net.au/science/moon/computer.htm), *Apollo 11 Lunar Surface Journal*, NASA, retrieved 2010-02-04
33. Eyles, Don (February 6, 2004), "Tales From The Lunar Module Guidance Computer" (http://klabs.org/history/apollo_11_alarms/eyles_2004/eyles_2004.htm), *27th annual Guidance and Control Conference*, Breckenridge, Colorado: American Astronautical Society
34. Tales From The Lunar Module Guidance Computer (http://klabs.org/history/apollo_11_alarms/eyles_2004/eyles_2004.htm)
35. Witt, Stephen (June 24, 2019). "Apollo 11: Mission Out of Control" (https://www.wired.com/story/apollo-11-mission-out-of-control/). *Wired*. San Francisco: Condé Nast Publications. Retrieved September 18, 2019.
36. Tomayko, James E. (2000), "NASA SP-2000-4224 — Computers Take Flight: A History of NASA's Pioneering Digital Fly-By-Wire Project" (http://www.klabs.org/history/history_docs/reports/dfbw_tomayko.pdf) (PDF), *The NASA History Series*, Washington, D.C.: NASA, retrieved 2009-09-01
37. Burkey, Ron. "VirtualAGC" (https://www.ibiblio.org/apollo/). *iBiblio*. Retrieved 10 April 2021.
38. "AGC source code collection on Github, maintained by iBiblio" (https://web.archive.org/web/20210507091417/https://github.com/virtualagc/virtualagc). *GitHub*. Archived from the original (https://github.com/virtualagc/virtualagc) on 7 May 2021. Alt URL (https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/virtualagc/virtualagc)
39. Collins, Keith (9 July 2016). "The code that took America to the moon was just published to GitHub, and it's like a 1960s time capsule" (http://qz.com/726338/the-code-that-took-america-to-the-moon-was-just-published-to-github-and-its-like-a-1960s-time-capsule/). *Quartz*. Retrieved 19 August 2016.

40. Garry, Chris. "Original Apollo 11 Guidance Computer (AGC) source code for the command and lunar modules" (https://web.archive.org/web/20210412083345/https://github.com/chrislgarry/Apollo-11). *GitHub*. Archived from the original (https://github.com/chrislgarry/Apollo-11/) on 12 April 2021. Alt URL (https://archive.softwareheritage.org/badge/origin/https://github.com/chrislgarry/Apollo-11/)
41. "Archiving and referencing the Apollo source code" (https://www.softwareheritage.org/2019/07/20/archiving-and-referencing-the-apollo-source-code/). *www.softwareheritage.org*. Retrieved 2021-09-09.
42. "Virtual AGC Home Page" (http://ibiblio.org/apollo/index.html). *ibiblio.org*. Retrieved 2021-09-09.
43. "GitHub - chrislgarry/Apollo-11: Original Apollo 11 Guidance Computer (AGC) source code for the command and lunar modules" (https://github.com/chrislgarry/Apollo-11). *GitHub*. Retrieved 2021-09-09.
44. "Apollo 11's source code is now on GitHub" (https://www.engadget.com/2016-07-10-apollo-11-source-code-on-github.html). *Engadget*. Retrieved 2021-09-09.

Sources

- Mindell, David A. (2008). *Digital Apollo: Human and Machine in Spaceflight* (https://books.google.com/books?id=gXYItzQARVoC&pg=PP1). Cambridge, Massachusetts: The MIT Press. ISBN 978-0-262-26668-0.

External links

Documentation on the AGC and its development

- *AGC4 Memo #9, Block II Instructions* (http://authors.library.caltech.edu/5456/01/hrst.mit.edu/hrs/apollo/public/archive/1689.pdf) – The infamous memo that served as de facto official documentation of the instruction set
- *Computers in Spaceflight: The NASA Experience* (http://www.hq.nasa.gov/office/pao/History/computers/Ch2-5.html) – By James Tomayko (Chapter 2, Part 5, *The Apollo guidance computer: Hardware*)
- *Computers Take Flight* (https://web.archive.org/web/20040719084435/http://www.dfrc.nasa.gov/History/Publications/PDF/DFBW.pdf) – By James Tomayko
- *The Apollo Guidance Computer - A Users View* (http://klabs.org/history/history_docs/ech/agc_scott.pdf) (PDF) – By David Scott, Apollo mission astronaut
- *Lunar Module Attitude Controller Assembly Input Processing* (http://www.klabs.org/mapld04/papers/g/g202_portillo_p.pdf) (PDF) – By José Portillo Lugo, History of Technology
- The MIT AGC Project (https://web.archive.org/web/20070612155102/http://authors.library.caltech.edu/5456/01/hrst.mit.edu/hrs/apollo/public/) – With comprehensive document archive
  - Luminary software source code listing (http://authors.library.caltech.edu/5456/01/hrst.mit.edu/groups/apollo/bibliography/q-and-a-fetch-comments.tcl_lastquestion_id=00016K&topic_id=11&topic=Document%20Library&entryformat=S&sortorder=author.html), for Lunar Module guidance computer. (nb. 622 Mb)
  - Colossus software source code listing (http://authors.library.caltech.edu/5456/01/hrst.mit.edu/groups/apollo/bibliography/q-and-a-fetch-comments.tcl_lastquestion_id=000156&topic_id=11&topic=Document%20Library&entryformat=S&sortorder=author.html), for Command Module guidance computer. (nb. 83 Mb)

- National Air and Space Museum's AGC Block I (https://archive.today/20110807043850/http://collections.nasm.si.edu/code/emuseum.asp?style=browse&currentrecord=1&page=search&profile=objects&searchdesc=A19720340000&quicksearch=A19720340000&newvalues=1&newstyle=single&newcurrentrecord=1) and Dsky (https://archive.today/20110807043856/http://collections.nasm.si.edu/code/emuseum.asp?style=browse&currentrecord=1&page=search&profile=objects&searchdesc=A19760811000&quicksearch=A19760811000&newvalues=1&newstyle=single&newcurrentrecord=1)
- Annotations to Eldon Hall's Journey to the Moon (http://authors.library.caltech.edu/5456/01/hrst.mit.edu/hrs/apollo/public/blairsmith3.htm) – An AGC system programmer discusses some obscure details of the development of AGC, including specifics of Ed's Interrupt

Documentation of AGC hardware design, and particularly the use of the new integrated circuits in place of transistors

- AGC Integrated Circuit Packages (http://klabs.org/history/ech/ic_packages/index.htm)
- Integrated Circuits in the Apollo Guidance Computer (http://klabs.org/history/history_docs/integrated_circuits/ic4-po.pdf)

Documentation of AGC software operation

- Delco Electronics, Apollo 15 (https://history.nasa.gov/alsj/a15/A15Delco.pdf) - Manual for CSM and LEM AGC software used on the Apollo 15 mission, including detailed user interface procedures, explanation of many underlying algorithms and limited hardware information. Note that this document has over 500 pages and is over 150 megabytes in size.
- Source code (http://googlecode.blogspot.com/2009/07/apollo-11-missions-40th-anniversary-one.html) for Command Module code (Comanche054) and Lunar Module code (Luminary099) as text.
- GitHub Complete Source Code (https://github.com/chrislgarry/Apollo-11/) Original Apollo 11 Guidance Computer (AGC) source code for the command and lunar modules.

Some AGC-based projects and simulators

- AGC Replica (http://klabs.org/history/build_agc/) – John Pultorak's successful project to build a hardware replica of the Block I AGC in his basement. Mirror site: AGC Replica (https://web.archive.org/web/20110222113340/http://echoesofapollo.com/resources/apollo-guidance-computer/).
- Virtual AGC Home Page (http://www.ibiblio.org/apollo/index.html) – Ronald Burkey's AGC simulator, plus source and binary code recovery for the Colossus (CSM) and Luminary (LEM) SW.
- Moonjs (http://www.svtsim.com/moonjs/agc.html) – A web-based AGC simulator based on Virtual AGC.
- Eagle Lander 3D (http://www.eaglelander3d.com/) Shareware Lunar Lander Simulator with a working AGC and DSKY (Windows only).
- AGC restarted 45 years later (https://www.youtube.com/watch?v=Bh_gP5aF3ys)

Feature Stories

- Weaving the way to the Moon (http://news.bbc.co.uk/1/hi/8148730.stm) (BBC News)
- Restorers try to get lunar module guidance computer up and running (https://www.wsj.com/video/restorers-try-to-get-lunar-module-guidance-computer-up-and-running/55526F08-0CC6-4CF7-9BCC-6BE6A20FE31C.html) (Wall Street Journal)
- *Computer for Apollo* video (https://www.youtube.com/watch?v=YIBhPsyYCiM)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Apollo_Guidance_Computer&oldid=1142501196"