Apollo Guidance Computer

Justin Ely

605.411 Foundations of Computer Architecture
6 December, 2016

---

1 Contents

2 Introduction and Overview

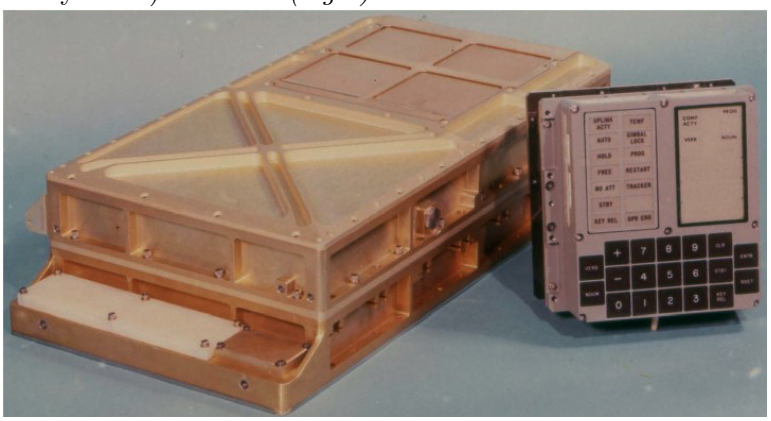
While the Apollo project was beginning and taking shape in the 1960's, digital computing was still in it's infancy. This provided great opportunities for the space program, from sharp decreases in size and sharp increases in power of computing equipement, but also caused a great deal of problems. As history has shown, the pace of advancement in digital computing has been incredibly rapid, which can produce interesting results as both technology and mission requirements evolve over a decade of use.

Both the Apollo Command Module and the Lunar Module employed a computer called the Apollo Guidance Computer (AGC) for control of guidance and navigation. This included activities like changing velocity and orientation. Astronauts would interface with the AGC through the Display and Keyboard interface (DSKY). (MIT PDF)

Created by the Charles Stark Draper Laboratory and fabricated by Raytheon, the AGC was a modest 12.5x13x6 inches but weighed an impressive 70 pounds. It used approx 55 watts of power which is comparable to a modern laptop. (Wikipedia)

The APG utilized a Von Neumann Architecture with a CISC instruction set architecture. The Von Neumann design, where both instructions and data occupied the same memory, served to simplify the hardware design

Figure 1: Apollo Guidance Computer (AGC) *(left)* next to the DSKY (Display and Keyboard) interface *(right)*.



and improve reliability. (MIT PDF). The word size for the APG was 16 bits, with 1-bit dedicated to parity and not accessible to the programmer or user. A total of 38K words of memory was available to the system. The APG is also a One-address machine, which uses only a single register in instructions. The other register is always assumed; typically the general-purpose accumulator the logical and arithmetic operations.

Though the AGC represented some of the most cutting edge technology for the time, current technology highlights how far computing has gone in so little time. By the 1980's, storage technology had advanced enough that the entirety of the command and lunar module software for all 6 lunar landings would easily fit on a single floppy disk. Today, the entire APG memory system would fit into a modern CPU cache.

3 Data Representation and Data Path

The data representation scheme used in the APG was a direct result of balancing the speed and efficiency of small word sizes against the accuracy of calculations. A large word size would give better precision to numerical values, but would require larger registers, bigger buses, and more memory. In the end, engineers concluded that a word size of 16 bits ($15 + 1$ parity) was sufficient for instructions and provided enough precision for the equations used in navigation and control. This would allow a double-precision value of 28 bits to give approximately 9 decimals digits of precision. This translated into distances of about 1 foot and velocities below 1 foot per second.

3.1 Integers

The APG represents integers using a modified version of one's complement. As in the standard implimentation, the first bit is used for the sign with the remaining 14 bits for the value (ignoring the leading parity bit). This allows representation of values from -16383 to 16383 with both positive and negative zeros.

The modification to the standard one's compliment is to handle overflow. scheme combats overflow by adding special detection of sign bit flips. When doing calculations, the 15 bit value is loaded into a 16-bit accumulator, where the 16th bit is a copy of the sign bit from the operand. This last bit is not accessible to the programmer; instead the AGC determines when an overflow occurs and alters programming accordingly.

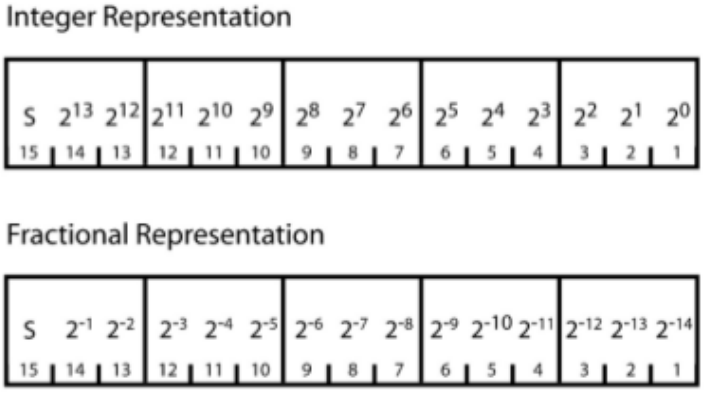
Namely, interrupts are temporarily prevented in the AGC (as the overflow detection bit would not be preserved accross an interrupt) until the overflow condition has been cleared. Such as by loading a new value into the accumulator or clearing the content.

Interestingly, when overflow does occur, the original sign bit (not the computed sign bit) is left in the register. This keeps the value of the same sign, instead of opposite sign, but changes the actual magnitude.

3.2 Real Numbers

Real numbers are incredibly important in the sort of mathematical calculations that the APG needed to perform. From inputs and results of calculations With only integer units, the APG contained no units for representing or performing arithmetic, on floating point numbers. Instead, an alternative system was used to do calculations on real numbers. This was called fractional notation.

Figure 2: Apollo Guidance Computer (AGC) *(left)* next to the DSKY (Display and Keyboard) interface *(right)*.



This system can only represent numbers between 0 and .9999.
$\frac{1}{3}$ = .33331298828125 in this fractional notation.

Put burden on programmer to arrange programs to only need a single order of magnitude at a time so that only a fractional part (and not an exponent) is needed.

3.3 Units

Of interesting note is the mathematical units used in the computations. Internally to the APG, the units were metric. However, the astronauts wanted to read and input values in Imperial, so conversions were employed in the layer between the core and the UI.

4 Memory

The memory system of the APG differs significantly from the systems we're used to today. Physically, three different memory structures were employed; flip-flops for registers, coincident current ferrite cores for writable memory, and fixed core rope for read-only memory (ROM). The use of magnetic memory was very important for space based applications as it was immune to most radiation-induced errors and would persist after the removal of the power source.

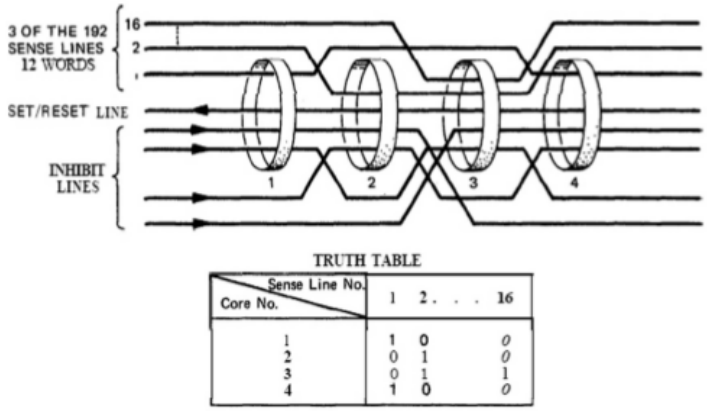
Ferrite-core?

magnetic core rope: The contents of read-only memory were physically written during fabrication. This read-only memory held some programs, but also kept mathematical constants to prevent overwrite. Ex. star locations, etc. This memory was compact, reliable, had random access, and was extremely inexpensive. Each bit cost between $.05 and $.07 to produce when most comparable solutions were a factor of 10 higher. (MIT) The main detractor to rope memory came from it's characteristic of being programmed during production. This meant that any desired change, due to changing requirements or simple mistakes in programming or manufacture, couldn't be implemented without re-wiring the core. Due to their complexity, this usually meant entire modules needed to be replaced for any change. (MIT)

Of the 38K words in total system memory, the first 48 words belonged to the central registers. (FIXME, Explain the various ones?) The next 2K were the writable memory, and the remaining 36K were ROM. Furthering the departments from current trends, the registers were addressable just like the other memory locations. This allows the same instructions to be used for memory and register access.

It may have been apparent that the 12-bit addresses in the instruction set can only reference 4K of storage and are not sufficient to address all locations in the 38K memory space. At this time in history, memory was relatively cheap and allowed systems to have far more memory than was

Figure 3: Apollo Guidance Computer (AGC) *(left)* next to the DSKY (Display and Keyboard) interface *(right)*.



physically addressable with a reasonable word-size. Contrast this today, where modern 64-bit word-sizes can address far more memory than can be included with the system. Virtual memory management wasn't yet in a mature form at this time in computing history.

To combat this, a "memory banking" scheme was used. Bank registers.

5 Instruction Set

The APG is a Complex Instruction Set Computer (CISC), where instructions perform complex tasks through microprogrammed implementations requiring many clock cycles. This is clearly evident in many of the available commands, such as ADS which performs addition into a memory location. This behavior is in stark contrast to a Reduced Instruction Set Computer (RISC), where typically only Load and Store commands may access memory.

The basic format for instructions in the AGC is shown below. The first bit provides parity, the next 3 bits are the op-code, and the remainder is the address.

Parity	Op-code	Address
1	011	000110101110

A 3-bit opcode allows for a total of 8 unique instructions. Clearly, trying to work the entire operation of a complicated spacecraft with only 8 instructions would present significant challenges. More instructions were necessary, but even as the mission requirements progressed it would be too costly to re-design the system around a larger word size. The engineers created some unique solutions to get more instructions within the existing limitations.

The first addition to the instruction set came from exploiting existing commands that could utilize smaller operand fields. In particular, instructions that write to memory don't need to access the higher addresses of the ROM, and will only ever use the lower 10 bits of the operand field. By combining the two left-over bits, called quarter-codes, with the opcodes for this subset of commands yields an entirely new suite of available instructions. This provided additional instructions by re-purposing the opcodes from (TS, ADS, INCR, XCH) (001, 010, 101 and 110).

The quarter codes expanded the instruction set considerably, and added efficiency to the system by giving useful meaning to commands and addresses that wouldn't work with the memory system, but still more were needed. To further add to the available commands, the engineers repurposed nonsensical (though valid) instructions. One example of this would be using the TC command to transfer control to the EBANK register. This register was only 3 bits wide and would never contain valid instructions or data. Instead, the AGC turned TC to EBANK into a brand new instruction: RELINT (Enable Interrupts). Similarly, this was done for TC commands to 3 other registers that would never be valid. This strategy served not only to expand the available commands, but to also serve as protection against programmer error. If TC to EBANK ever was accidently entered, without the protection of the special commanding, the effects could be very difficult to debug.

Lastly, when these commands still proved insufficient, a brute-force software solution was employed. One of the special cases described above, TC to BBANK, was changed to instruct the AGC to interpret the subsequent command using an entirely new set of opcodes. This effectively created a second instruction set, complete with quarter codes and special cases like the original. Though this nearly doubled the possible instruction set, it comes at the cost of increased software overhead to processing the extended set. This was mitigated as much as possible by placing the most often used opcodes in the standard set, and the less-frequently used ones in the extended.

Similarly, a 12-bit address also presents a problem. 12 bits is sufficient only to address 4K of memory; far short of the 38K necessary.

Group	Command	Meaning
Sequence Changing	TC	Transfer Control
	TCF	Transfer Control to Fixed
	CCS	Count Compare and Skip
	BZF	Branch Zero to Fixed
	BZMF	Branch Zero or Minus to Fixed
Reading and Writing	CA	Clear and Add
	CS	Clear and Subtract
	DCA	Double Clear and Add
	DCS	Double Clear and Subtract
	TS	Transfer to Storage
	XCH	Exchange A and K
	LXCH	Exchange L and K
	QXCH	Exchange Q and K
	DXCH	Double Exchange
Instruction Modification	INDEX	Index Next Instruction
	INDEXE	Index Next Instruction Extended
Arithmetic and Logic	AD	Add
	Su	Subtract
	ADS	Add to Storage
	MSU	Modular Subtract
	INCR	Increment
	AUG	Augment
	DIM	Diminish
	DAS	Double Add to Storage
	MASK	Mask A by K
	MP	Multiply
	DV	Divide
I/O Channel	READ	READ KC
	WRITE	Write Channel KC
	RAND	Read and Mask
	WAND	Write and Mask
	ROR	Read and Superimpose
	WOR	Write and Superimpose
	RXOR	Read and Invert
Miscellaneous	XXALQ	Execute Extracode Using A, L, and Q
	XLQ	Execute Using L and Q
	RETURN	Return From Subroutine
	RELINT	Enable Interrupts
	INHINT	Inhibit Interrupts
	EXTEND	Set Extracode Flag
	EDRUPT	Ed Smally's Interrupt
	RESUME	Resume Interrupted Program

6 Summary

7 References

- The Apollo Guidance Computer: architecture and operation, Frank O'Brien., Chichester : Praxis, 2010.
- https://en.wikipedia.org/wiki/Apollo_Guidance_Computer
- Apollo 11 Guidance Computer source code: https://github.com/chrislgarry/Apollo-11/
- Delco Electronics, Apollo 15 - Manual for CSM and LEM AGC software used on the Apollo 15 mission