KEEP

Chapter 7

Some aspects of the logical design of a control computer: a case study[1]

R. L. Alonso / H. Blair-Smith / A. L. Hopkins

Summary Some logical aspects of a digital computer for a space vehicle are described, and the evolution of its logical design is traced. The intended application and the characteristics of the computer's ancestry form a framework for the design, which is filled in by accumulation of the many decisions made by its designers. This paper deals with the choice of word length, number system, instruction set, memory addressing, and problems of multiple precision arithmetic.

The computer is a parallel, single address machine with more than 10,000 words of 16 bits. Such a short word length yields advantages of efficient storage and speed, but at a cost of logical complexity in connection with addressing, instruction selection, and multiple-precision arithmetic.

1. Introduction

In this paper we attempt to record the reasoning that led us to certain choices in the logical design of the Apollo Guidance Computer (AGC). The AGC is an onboard computer for one of the forthcoming manned space projects, a fact which is relevant primarily because it puts a high premium on economy and modularity of equipment, and results in much specialized input and output circuitry. The AGC, however, was designed in the tradition of parallel, single-address general-purpose computers, and thus has many properties familiar to computer designers [Richards, 1955], [Beckman et al., 1961]. We will describe some of the problems of designing a short word length computer, and the way in which the word length influenced some of its characteristics. These characteristics are number system, addressing system, order code, and multiple precision arithmetic.

A secondary purpose for this paper is to indicate the role of evolution in the AGC's design. Several smaller computers with about the same structure had been designed previously. One of these, MOD 3C, was to have been the Apollo Guidance Computer, but a decision to change the means of electrical implementation (from core-transistors to integrated circuits) afforded the logical designers an unusual second chance.

It is our belief, as practitioners of logical design, that designers, computers and their applications evolve in time: that a frequent reason for a given choice is that it is the same as, or the logical nest step to, a choice that was made once before.

A recent conference on airborne computers [*Proc. Conf. Spaceborne Computer* Eng., Anaheim, Calif., Oct. 30–31, 1962] affords a view of how other designers treated two specific problems: word length and number system. All of these computers have word lengths of the order of 22 to 28 bits, and use a two's complement system. The AGC stands in contrast in these two respects, and our reasons for choosing as we did may therefore be of interest as a minority view.

2. Description of the AGC

The AGC has three principal sections. The first is a memory, the fixed (read only) portion of which has 24,376 words, and the erasable portion of which has 1024 words. The next section may be called the ***central section;*** it includes, besides an adder and a parity computing register, an instruction decoder (SQ), a memory address decoder (S), and a number of addressable registers with either special features or special use. The third section is the *sequence generator* which includes a portion for generating various microprograms and a portion for processing various interrupting requests.

The backbone of the AGC is the set of 16 write busses: these are the means for transferring information between the various registers shown in Fig. 1. The arrowheads to and from the various registers show the possible directions of information flow.

In Fig. 1, the data paths are shown as solid lines: the control paths are shown as broken lines.

Memory: fixed and erasable

The Fixed Memory is made of wired-in "ropes" [Alonso and Laning, 1960], which are compact and reliable devices. The number of bits so wired is about 4×10^5. The cycle time is 12 μsec.

The erasable memory is a coincident current system with the same cycle time as the fixed memory. Instructions can address registers in either memory, and can be stored in either memory.

[1] *IEEE Trans.*, EC-12 (6), 687–697 (December, 1963).

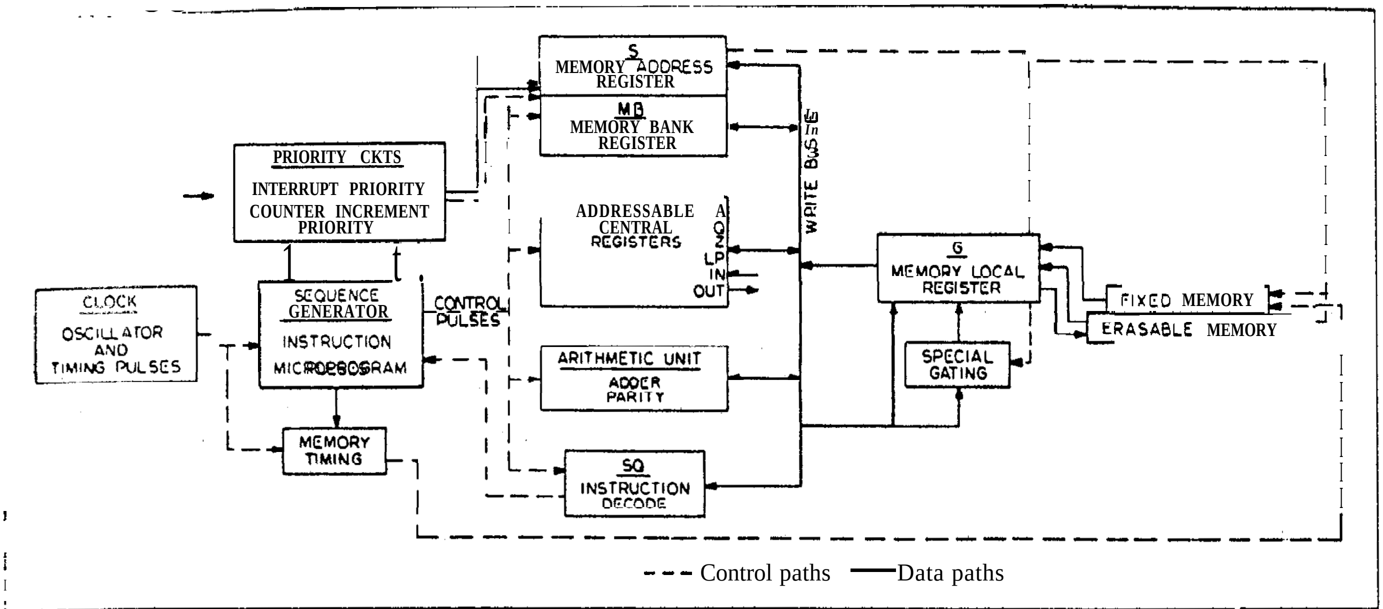

fig. 1. AGC block diagram.

The only logical difference between the two memories is the inability to change the contents of the fixed part by program steps.

Each word in memory is 16 bits long (15 data bits and an odd parity bit). Data words are stored as signed 14 bit words using a one's complement convention. Instruction words consist of 3 order code bits and 12 address code bits.

The contents of the address register S uniquely determine the address of the memory word only if the address lies between octal 0000 and octal 5777, inclusive. If the address lies between octal 6000 and octal 7777, inclusive, the address in S is modified by the contents of the memory bank register *MB*. The modification consists in adding some integral multiplies of octal 2000 to the address in S before it is interpreted by the decoding circuitry. The memory bank register *MB* is itself addressable; its address, however, is *not* modified by its own contents.

Transfers in and out of memory are made by way of a memory local register *G*. For certain specific addresses, the word being transferred into *G* is not sent directly, but is modified by a special gating network. The transformations on the word sent to *G* are right shift, left shift, right cycle, and left cycle.

Central section

The middle part of Fig. 1 shows the central section in block form. It consists of the address register S and the memory bank register *ME* both of which were mentioned above. There is also a block of addressable registers called "central and special registers," which will be discussed later, an arithmetic unit, and an instruction decoder register SQ.

The arithmetic unit has a parity generating register and an adder. These two registers are not explicitly addressable.

The SQ register bears the same relation to instructions as the S register bears to memory locations; neither S nor SQ are explicitly addressable.

The central and special registers are **A**, Q, Z, *LP*, and a set of input and output registers. Their properties are shown in Table 1.

Sequence *generator*

The sequence generator provides the basic memory timing, the sequences of control pulses (microprograms) which constitute an instruction, the priority interrupt circuitry, and **a** number of scaling networks which provide various pulse frequencies used by the computer and the rest of the navigation system.

Instructions are arranged so as to last an integral number of memory cycles. The list of 11 instructions *is* treated in detail in Sec. 6. In addition to these there are a number of "involuntary" sequences, not under normal program control. which may break into the normal sequence of instructions; these are triggered either by external events, or by certain overflows within the AGC, and

Table 1 Special and central registers

Register (s)	Octal address	Purpose and/or properties
A	0000	Central accumulator. Most instructions refer to A.
Q	0001	If a transfer of control (TC) occurred at L, $(Q) = L + 1$.
Z	0002	Program counter. Contains $L + 1$, where L is the address of the instruction presently being executed.
LP	0003	Low product register. This register modifies words written into it by shifting them in a special way.
IN	. . .	Several registers which arc used for sampling either external lines. or internal computer conditions such as time or alarms.
OUT	. . .	Several output registers whose bits control switches, networks, and displays.

may be divided into two categories: counter incrementing and program interruption.

Counter incrementing may take place between any two memory cycles. External requests for incrementing a counter are stored in a counter priority circuit. At the end of every memory cycle a test is made to see if any incrementing requests exist. If not, the nest normal memory cycle is esecuted directly, with no time between cycles. If a request is present, an incrementing memory cycle is executed. Each "counter" is a specific location in erasable memory. The incrementing cycle consists of reading out the word stored in the counter register, incrementing it (positively or negatively), or shifting it, and storing the results back in the register of origin. All outstanding counter incrementing requests are processed before proceeding to the nest normal memory cycle. This type of interrupt provides for asynchronous incremental or serial entry of information into the working erasable memory. The program steps may refer directly to a "counter" to obtain the desired information and do not have to refer to input buffers. Overflows from one counter may be used as the input to another. A further property of this system is that the time available for normal program steps is reduced linearly by the amount of counter activity present at any given time.

Program interruption occurs between normal program steps rather than between memory cycles. An interruption consists o storing the contents of the program counter and transferring con trol to a fixed location. Each interrupt line has a different locatior associated with it. Interrupting programs may not be interrupted but interrupt requests are not lost, and are processed as soon a the earlier interrupted program is resumed. Culling the resum sequence. which restores the program counter. is initiated by referencing a special address.

3. Word length

In an airborne computer, granted the initial choice of parallel transfer of words within it, it is highly desirable to minimize the word length. This is because memory sense amplifiers, being high-gain class A amplifiers, are considerably harder to operate with wide margins (of temperature, voltages, input signal) than, say, the circuits made up of NOR gates. It is best to have as few of these as possible. Furthermore, the number of ferrite-plane inhibit drivers equals the number of bits in a word in this case. Similarly, the time required for a carry to propagate in a parallel adder is proportional to the word length, and in the present case, this factor could be expected to affect the microprogram-ning of instructions. The initial intent, then, was to have as short a word length as possible.

Another initial choice is that the AGC should be a "common storage" machine, which means that instructions may be executed from erasable memory as well as **from fixed memory,** and that **data** (obviously constants, in the case of fixed memory) may be stored in either memory. This in turn means that the word sizes of both types of memory must be compatible in some sense;,for the AGC, the easiest form of compatibility is to have equal word lengths. So-called "separate storage" solutions which allow different word lengths for instructions and data can be made to work [Walendziewicz, 1962] but they have a drawback in that three memories are then required: a data memory (erasable),and two fised memories, one for instructions and one for constants. In addition, we have found that separate storage machines are more awkward to program, and use memory less efficiently. than common storage machines.

There are three principal factors in the choice of word length. These are:

1 Precision desired in the representation of navigational variables.

2 Range of the input variables which are entered serially and counted.

3 Instruction word format. Division of instruction words into two fields, one for operation code and one for address.

As a start, the choice of word length (15 bits) for two previous machines in this series was kept in mind as a satisfactory word length from the point of view of mechanization; *i.e.*, the number of sense amplifiers, inhibit drivers, the carry propagation time, etc., were all considered satisfactory. The act of "choosing" word length really meant whether or not to alter the word length, at the time of change from MOD 3C to the AGC, and in particular whether to increase it. The influence of the three principal factors will be taken up in turn.

Precision of data words

The data words used in the AGC may be divided roughly into two classes: data words used in elaborate navigational computations, and data words used in the control of various appliances in the system. Initial estimates of the precision required by the first class ranged from 27 to 32 bits, $O(10^{8\pm1})$. The second class of variables could almost always be represented with 15 bits. The fact that navigational variables require about twice the desired 15-bit word length means that there is not much advantage to word sizes between 15 and 28 bits, as far as precision of representation of variables is concerned, because double-precision numbers must be used in any event. Because of the doubly signed number representation for double-precision words, the equivalent word length is 29 bits (including sign), rather than 30, for a basic word length of 15 bits.

The initial estimates for the proportion of 15-bit vs 29-bit quantities to be stored in both fixed and erasable memories indicated the overwhelming preponderance of the former. It was also estimated that a significant portion of the computing had to do with control, telemetry and display activities, all of which can be handled more economically with short words. A short word length allows faster and more efficient use of erasable storage because it reduces fractional word operations, such as packing and editing; it also means a more efficient encoding of small integers.

Range of input variables

As a control computer, the AGC must make analog-to-digital conversions, many of which are of shaft angles. Two principal forms of conversion exist: one renders a whole number, the other produces a train of pulses which must be counted to yield the desired number. The latter type of conversion is employed by the AGC, using the counter incrementing feature.

When the number of bits of precision required is greater than the computer's word length, the effective length of the counter must be extended into a second register, either by programmed scanning of the counter register, or by using a second counter register to receive the overflows of the first. Whether programmed scanning is feasible depends largely on how frequently this scanning must be done. The cost of using an extra counter register is directly measured in terms of the priority circuit associated with it.

In the AGC, the equipment saved by reducing the word length below 15 bits would probably not match the additional expense incurred in double-precision extension of many input variables. The question is academic, however, since a lower bound on the word length is effectively placed by the format of the instruction word.

Instruction word format

An initial decision was made that instructions would consist of an operation code and a single address. The straightforward choices of packing one or two such instructions per word were the only ones seriously considered, although other schemes, such as packing one and a half instructions per word, are possible [England, 1962]. The previous computers MOD 3S and MOD 3C had a 3-bit field for operation codes and a 12-bit field for addresses, to accommodate their 8 instruction order codes and 4096 words of memory. In the initial core-transistor version of the AGC (*i.e.*, MOD 3C), the 8 instruction order codes were in reality augmented by the various special registers provided, such as shift right, cycle left, edit, so that a transfer in and out of one of these registers would accomplish actions normally specified by the order code (see Sec. 6). These registers were considered to be more economical than the corresponding instruction decoding and control pulse sequence generation. Hence the 3 bits assigned to the order code were considered adequate, albeit not generous. Furthermore, as will be seen, it is possible to use an indexing instruction so as to increase to eleven the number of explicit order codes provided for.

The address field of 12 bits presented a different problem. At the time of the design of MOD 3C we estimated that 4000 words would satisfy the storage requirements. By the time of redesign it was clear that the requirement was for 10^5 words, or more, and the question then became whether the proposed estension of the address field by a bank register (see Sec. 7) was more economical than the addition of 2 bits to the word length. For reasons of modularity of equipment, adding 2 more bits to the word length would result in adding 2 more bits to all the central and special registers, which amounts to increasing the size of the nonmemory portion of the AGC by 10 per cent.

In summary, the 15-bit word length seemed practical enough so that the additional cost of extra bits in terms of size, weight, and reliability did not seem warranted. A 14-bit word length was thought impractical because of the problems with certain input variables, and it would further restrict the already somewhat cramped instruction word format. Word lengths of 17 or 18 bits would result in certain conceptual simplicities in the decoding of instructions and addresses, but would not help in the representation of navigational variables. These require 28 bits, and so they must be represented to double precision in any event.

4. Number representation

Signed numbers

In the absence of the need to represent numbers of both signs, the discussion of number representation would not extend beyond the fact that numbers in AGC are expressed to base two. But the accommodation of both positive and negative numbers requires that the logical designer choose among at least three possible forms of binary arithmetic. These three principal alternatives are: (1) one's complement, (2) two's complement, and (3) sign and magnitude [Richards, 1955].

In one's complement arithmetic, the sign of a number is reversed by complementing every digit, and "end around carry" is required in addition of two numbers.

In two's complement arithmetic, sign reversal is effected by complementing each bit and adding a low order one, or some equivalent operation.

Sign and magnitude representation is typically used where direct human interrogation of memory is desired, as in "post-mortem" memory dumps, for example. The addition of numbers of opposite sign requires either one's or two's complementation or comparison of magnitude, and sometimes may use both. No advantage is offered in efficiency with the possible exception of sign changing, which only requires changing the sign bit. A disadvantage is engendered in magnetic core logic machines by the extra equipment needed for subtraction or conditional recomplementation.

The one's complement notation has the advantage of having easy sign reversal, which is equivalent to Boolean complementation: hence a single machine instruction performs both functions. Zero is ambiguously represented by all *zero's* and by all one's, so that the number of numerical states in an n-bit word is $2^n - 1$.

Two's complement arithmetic is advantageous where end around carry is difficult to mechanize, as is particularly true in serial computers. An n-bit word has 2^n states, which is desirable for input conversions from such devices as pattern generators, geared encoders, or binary scalers. Sign reversal is awkward, however, since a full addition is required in the process.

The choice in the case of the AGC was to use one's complement arithmetic in general processing, and two's complements for certain input angle conversions. Since the only arithmetic done in the latter case is the addition of plus or minus *one*, the two's complement facility is provided simply by suppressing end around carry and using the proper representation of minus one. The latter is stored as a fixed constant, so that no sign reversal is required.

Modified one's *complement system*

In a standard one's complement adder, overflow is detected by examining carries into and out of the sign position. These overflow indications must be "caught on the fly" and stored separately if they are to be acted upon later. The number system adopted in the ACC has the advantage of being a one's complement system with the additional feature of having a static indication of overflow. The implementation of the method depends on the AGC's not using a parity bit in most central registers. Because of certain modular advantages, 16, rather than 15, columns are available in all of the central registers, including the adder. Where the parity bit is not required, the extra bit position is used as an extra column. The virtue of the 16-bit adder is that the overflow of a 15-bit sum is readily detectable upon examination of the two high order bits of the sum (see Fig. 2). If both of these bits are the same, there is no overflow. If they are different, overflow has occurred with the sign of the highest order bit.

The interface between the 16-bit adder and the 15-bit memory is arranged so that the sign bit of a word coming from memory enters both of the two high order adder columns. These are denoted S_2 and S_1 since they both have the significance of sign bits. When a word is transferred from the accumulator A to memory, only one of these two signs can be stored. Our choice was to store the S_2 bit, which is the standard one's complement sign except in the event of overflow, in which case it is the sign of the two operands. This preservation of sign on overflow is an important asset in dealing with carries between component words of multiple-precision numbers (see Sec. 5).

In a standard one's complement system, a series of additions may result in subtotals which overflow, yet still produce a valid sum so long as the total does not exceed the capacity of one word. In a modified one's complement system, however, where sign is preserved on overflow, this is no longer true; and the total may depend on the order in which the numbers are added; this is not a serious drawback, but it must be accounted for in all phases of logical design and programming.

		STANDARD						MODIFIED						
		S_1	4	3	2	1		S_2	S_1	4	3	2	1	
EXAMPLE 1:	Both operands positive; Sum positive, no overflow. Identical results in both systems.	0	0	0	0	1		0	0	0	0	0	1	
		0	0	0	1	1		0	0	0	0	1	1	
		0	0	1	0	0		0	0	0	1	0	0	
EXAMPLE 2:	Both operands positive; positive overflow. Standard result is negative; Modified result is positive using S_2 as sign of the answer. Positive overflow indicated by $S_1 \cdot \bar{S}_2$.	0	1	0	0	1		0	0	1	0	0	1	
		0	1	0	1	1		0	0	1	0	1	1	
		1	0	1	0	0		0	1	0	1	0	0	
EXAMPLE 3.	Both operands negative; Sum negative, no overflow. End around carry occurs. Identical results in both systems using either S_1 or S_2 as the sign of the answer.	1	1	1	1	0		1	1	1	1	1	0	
		1	1	1	0	0		1	1	1	1	0	0	
		1	1	0	1	0		1	1	1	0	1	0	
						1	carry						1	carry
		1	1	0	1	1		1	1	1	0	1	1	
EXAMPLE 4:	Both operands negative; negative overflow. Standard result is positive: modified result is negative using S_2 as the sign of the answer. Negative overflow indicated by $\bar{S}_1 \cdot S_2$.	1	0	1	1	0		1	1	0	1	1	0	
		1	0	1	0	0		1	1	0	1	0	0	
		0	1	0	1	0		1	0	1	0	1	0	
						1	carry						1	carry
		0	1	0	1	1		1	0	1	0	1	1	
EXAMPLE 5:	Operands have opposite sign; Sum positive. Identical results in both systems.	1	1	1	1	0		1	1	1	1	1	0	
		0	0	0	1	1		0	0	0	0	1	1	
		0	0	0	0	1		0	0	0	0	0	1	
						1	carry						1	carry
		0	0	0	1	0		0	0	0	0	1	0	
EXAMPLE 6:	Operands have opposite sign: sum negative. Identical results in both systems.	1	1	1	0	0		1	1	1	1	0	0	
		0	0	0	0	1		0	0	0	0	0	1	
		1	1	1	0	1		1	1	1	1	0	1	

Fig. 2. Illustrative example of properties of modified one's complement system.

5. Multiple precision arithmetic

A short word computer can be effective only if the multiple-precision routines are efficient corresponding to their share of the computer's word load. In the AGC's application there is enough use for multiple-precision arithmetic to warrant consideration in the choice of number system and in the organization of the instruction set. Although the limited number of order codes prohibits multiple-precision instructions, special features are associated with the conventional instructions to expedite multiple-precision operations.

Independent sign representation

A variety of formats for multiple-precision representation are possible, probably the most common of these is the identical sign representation in which the sign bits of all component words agree. The method used in the AGC allows the signs of the components to be different.

Independent signs arise naturally in multiple-precision addition and subtraction, and the identical sign representation is costly because sign reconciliation is required after every operation. For example, $(+6, +4) + (-4, -6) = (+2, -2)$, a mixed sign representation of $(+1, +8)$. Since addition and subtraction are the most frequent operations, it is economical to store the result as it occurs and reconcile signs only when necessary. When overflow occurs in the addition of two components, a *one* with the sign of the overflow is carried to the addition of the next higher components. The sum that overflowed retains the sign of its operands. This overflow is termed an *interflow* to distinguish it from an overflow that arises when the maximum multiple-precision number is exceeded.

The independent sign method has a pitfall arising from the fact that every number has two representations, either one of which may occur as a sum. There are some numbers for which one of the representations esceeds the capacity of the most significant component. The overflow is false in the sense that the double-precision capacity is not esceeded, only the single word capacity of the upper component. Sign reconciliation can be used in this case to yield an acceptable representation. This problem can be avoided if all numbers are scaled so that none are large enough to produce false overflows. Such a restriction is not necessary, however, since the false overflow condition arises infrequently and can be detected at no expense in time. The net cost of reconciliation is therefore very low.

Multiplication and division

For triple and higher orders of precision, multiplication and division become excessively complex, unlike addition and subtraction where the complexity is only linear with the order of precision.

The algorithm for double-precision multiplication is directly applicable to numbers in the independent sign notation. False overflow does not arise, and the treatment of interflow is simplified by an automatic counter register which is incremented when overflow occurs during an add instruction. The sign of the counter increment is the same as the sign of the overflow; and the increment takes place while one of the product components of next higher order is stored in that counter.

Double-precision division is exceptional in that the independent sign notation may not be used; both operands must be made positive in identical sign form, and the divisor normalized so that the left-most nonsign bit is *one.*

Triple prccision

A few triple-precision quantities are used in the AGC. These are added and subtracted using independent sign notation with interflow and overflow features the same as those used for double-precision arithmetic.

6. Instruction set

Basic design criteria

The implicit requirements for any von Neumann-type machine demand that facilities exist for:

1 Fetching from memory
2 Storing in memory
3 Negating (complementing)
4 Combining two operands (*e.g.*, addition)
5 Address modification (more generally, executing as an instruction the result of arithmetic processing)
6 Normal sequencing (to each location from which an instruction can be executed there corresponds one location whose contents are the nest instruction)
7 Conditional sequence changing, or transfer of control
8 Input
9 Output

An instruction can, of course, provide several of these facilities. For instance, some computers have an instruction that subtracts the contents of a memory location from an accumulator and leaves the result in that memory location and in the accumulator; this instruction fulfills all of requirements 1–4 above. Requirement 5 is met in a somewhat primitive manner if instructions can be executed from erasable memory, and is met elegantly by the use of index registers. Still another scheme, somewhat similar to one used in the Bendix *G-20,* is employed in the AGC. Requirement 6 is usually fulfilled by having an instruction location counter which contains the address of the next instruction to be executed, and is incremented by *one* when an instruction is fetched. Alternatively, each instruction may include the address of the next instruction, as is often done in machines having drum memories. In the AGC, as in most short-word computers, the former method, with one single-address instruction per word, is clearly the simplest and cheapest. Requirement 7 is generally met by examining a condition such as the s i p of an accumulator and, if the condition is satisfied, either incrementing the instruction location counter (skipping). or using an address included in the instruction as that of the next instruction (conditional transfer of control). An unconditional transfer of control is usual but not necessary, since any desired condition can be forced. Most machines have special input-output instructions to satisfy requirements 8 and 9. In the AGC, however, since input and output is through addressable registers. input is subsumed under fetching from memory, and output under storing in memory. Counter incrementing and program interruption aid these functions also.

Further criteria

The major goals in the AGC were efficient use of memory, reasonable speed of computing, potential for elegant programming, efficient multiple precision arithmetic, efficient processing of input and output, and reasonable simplicity of the sequence generator. The constraints affecting the order code as a whole were the word length, one's complement notation, parallel data transfer, and the characteristics of the editing registers. The ground rules governing the choice of instructions arose from these goals and constraints.

a Three bits of an instruction word are devoted to operation code.

b Address modification must be convenient and efficient.

c There should be a multiply instruction yielding a double length product.

d Treatment of overflow on addition must be flexible.

e A Boolean combinatorial operation should be available.

f No instruction need be devoted to input, output, or shifting.

This list is by no means complete, but gives a good indication of what kind of computer the AGC has to be. In the following paragraphs the ways in which the instructions fulfill the above requirements are described.

Details of the instruction set

In the listing that follows, L denotes the location of the instruction; K denotes the data address contained in the instruction. Parentheses mean "content of," and the leftward arrow means that the register named at the arrowhead is set to the quantity named to the right.

L: TC K; Transfer Control

$Q \leftarrow L + 1$; go to R.

This is the primary method of transferring control to any stated location, and thus meets part of requirement 7. The setting of the return address register Q renders complex subroutines feasible. TC Q may be used to return from a subroutine (with no other TC's) because the binary number "$L + 1$" is the same as the binary word "TC $L + 1$," by virtue of the TC code being all zeros. TC A behaves like an "execute" instruction, executing whatever instruction is in A, because Q follows A in the address pattern. see Table 1.

L: CCS K; Count, Compare, and Skip

If $(K) > +0$, $A \leftarrow (K) - 1$. no skip: if $(K) = +0$, $A \leftarrow +0$. skip to $L + 2$; if $(K) < -0$, $A \leftarrow 1 - (K)$, skip to $L + 3$; if $(K) = -0$, $A \leftarrow +0$, skip to $L + 4$.

This instruction fulfills the remainder of requirement 7 and provides several features. It is clear that in a machine with a 3-bit operation code there should be only one code devoted entirely to branching, if at all possible. It is inefficient to program a zero test using only a sign-testing code: it is even more inefficient to program a sign test using only a zero-testing code. This instruction was therefore designed to test both types of conditions simultaneously. It has to be a four-way branch, and since there is only one address per instruction, it follows that CCS must be a skipping-type branch.

The function of (K) delivered to A is the diminished absolute value (DABS). It serves two primary purposes: to do most of the work in generating an absolute value, and to apply a negative increment to the contents of a loop-counting register, so that CCS has some of the properties of TIS in the IBM 704.

L: INDEX K; Index using *K*

Use $(L + 1) + (K)$ as the next instruction.

In a short-word machine where there is no room in the instruction word to specify indexing or indirect addressing, this code meets requirement 5 in a way far superior to forming an instruction and placing it in A or in erasable memory for execution. INDEX operates on whole words, so that the operation code as well as the address may be modified. It may be used recursively (consider the implications of several INDEX's in succession, assuming that no operation codes are modified). Finally, it permits more than 8 operation codes to be specified in 3 bits, since overflow of the indexing addition is detectable.

L: XCH K; Exchange

This instruction meets requirements 1, 2, and 8. When K is in fixed memory, it is simply a data-fetching (clear and add) code. Its use with erasable memory aids efficiency by reducing the need for temporary storage. XCH is also an important input instruction in a machine where addressable counters, incremented in response to external events, are an input medium, because a counter can be read out and reset (to zero or any desired value) by SCH with no chance of missing a count.

L: CS K; Clear and Subtract

$A \leftarrow -(K)$.

CS is the primary means of sip-changing and logical negation, and so fulfills requirements 1 and 3. Since there is no clear and add instruction, it is the usual operation for nondestructive readout of erasable memory in simple data transfers. that is. when no addition or other arithmetic is required. Usually the programming can be arranged so that complementing during transfer is acceptable; otherwise the CS can be followed by CS A before storing.

L: TS K; Transfer to Storage

$K \leftarrow (A)$; if (A) includes $\pm$ overflow, $A \leftarrow \pm 1$, skip to $L + 2$.

This instruction is the primary means of transfers to memory and output, satisfying requirements 2 and 9. It is also the most convenient method of testing for overflow. Since A and the other central registers have two sign positions, overflow indication is retained in a central register. TS always stores (A) and tests whether overflow is present. If K is in erasable memory and is not a central register, the lower-order sign bit S_1 is not transmitted; this is the process of overflow correction. If positive overflow indication is present in A, TS skips over the next instruction and sets $A \leftarrow +1$ ($+1$ denotes octal 000001); if negative overflow is present, TS skips over the next instruction and sets $A \leftarrow -1$ (-1 denotes octal 177776); otherwise (A) are unchanged. The sequence

TS K
XCH ZERO (ZERO in fixed memory)

suffices to store in K an overflow-corrected word of a multiple-precision sum and leave in A the interflow to the next higher-order part. TS A skips if either type of overflow is present, but leaves all 16 bits of (A) unchanged.

Finally, a computed transfer of control may be achieved by TS Z because Z is the program counter; only the low-order 12 bits of (A) are significant, being the address of the instruction to which control is transferred. Overflow in (A) in this case does not affect the transfer but sets $A \leftarrow \pm 1$.

L: AD K; Add

$A \leftarrow (A) + (K)$; if the final (A) includes $\pm$ overflow, OVCTR $\leftarrow$ (OVCTR) ± 1.

Addition is the most frequently used combinatorial operation (requirement 4). The property of OVCTR is used chiefly in developing double-precision products and quotients, partly because the additions in these processes are less susceptible to false overflow than are multiple-precision additions.

L: MASK K; Mask

$A \leftarrow (A) \cap (K)$.

This is the only combinatorial Boolean instruction, and may be used with CS to generate any Boolean function.

Extracodes

The AGC instruction set was carried over in large part from its ancestor, MOD 3C [Alonso et al., 1961]. All instructions of MOD 3C were retained in the AGC, modifications and additions being adopted where a substantial increase in computing power could be obtained at small cost. The MOD 3C instruction set was like the one described above for the AGC with two major exceptions: first, instead of a mask instruction, MOD 3C had a multiply instruction. Second, the transfer to storage instruction did not include the property of skipping on overflow, although it did have properties which aided masking.

After the design of MOD 3C was completed, it was discovered that the INDEX instruction could be used to expand the instruction set beyond eight instructions by producing overflow in the instruction word following the INDEX. For example, the addition of octal 47777 to the instruction word "CS K" in the course of an INDEX instruction will cause negative overflow, producing MP K, a multiply instruction with operand address K.

In order to implement the extracodes in the AGC, it was necessary to provide a path from the high-order 4 bits of the adder to the unaddressable sequence selection register SQ. Part of this path is the unaddressable buffer register B; these requirements helped to suggest the benefits of retaining two sign bit positions in all the central registers.

In principle, eight additional instruction codes can be obtained by causing overflow, but we did not feel obliged to use them all. Because every extracode must be indexed, the instructions chosen for this class had two properties to some degree: they are normally indexed, or they take long enough so that the cost of indexing without address modification is small. All the extracodes are combinatorial, and therefore relate to requirement 4.

L: MP K; Multiply

$A \leftarrow$ upper part, $LP \leftarrow$ lower part, of $(A) \cdot (K)$; the two words of the product agree in sign, which is determined strictly by the sign bits of the operands.

Experience with MOD 3C showed that it was worthwhile making a completely algebraic, self-contained multiply instruction, especially in doing double-precision multiplication whose operands have independent signs. The AGC multiply is much faster than that of MOD 3C, being limited by adder carry propagation time rather than core-switching time.

L: DV K; Divide

$A \leftarrow$ quotient, $Q \leftarrow -|\text{remainder}|$, of $(A)/(K)$; $LP \leftarrow$ nonzero number with the sign of the quotient.

Many facets of AGC design originally adopted for other reasons combined to make a divide instruction inexpensive. The foremost of these is the nature of the editing registers, which are in the standard erasable memory and have no special wiring. The special properties of these registers are supplied by a shift or cycle of the word being written into the memory local register G, when the address of an editing register is selected. The central loop of DV selects such an address and inhibits memory operations, so that all the left shifts required in division are accomplished in the G register while the editing register itself remains unchanged. The microprogrammed nature of order construction makes a restoring algorithm more efficient than a nonrestoring one. The quotient delivered to A has a sign determined according to normal algebraic rules by the signs of (A) and (K); the same sign is available in *LP* to aid in determining the correct sign of the remainder from those of the divisor and quotient in case the quotient has been absorbed by subsequent processing. DV is not usually indexed, but it pays such large benefits in space and time, especially in double-precision division. that the cost of extracode indexing is negligible. If the divisor is less in magnitude than the dividend, or is zero, the quotient has correct sign and, in general, maximum magnitude. No infinite loop results in any case.

L: SU *K; Subtract*

$A \leftarrow (A) - (K)$; if the final (A) includes $\pm$ overflow, $\text{OVCTR} \leftarrow (\text{OVCTR}) \pm 1$.

The primary justification for this instruction is that it allows multiple-precision addition subroutines to be changed into multiple-precision subtract subroutines merely by changing the indexing quantity. There are occasions in the middle of involved calculations where it is clumsy to construct a subtraction out of complementations and additions, especially when the sign of an overflow is of interest. Since SU differs from AD only in that the operand from memory is read out of the complement side of the buffer register B rather than the direct side, its cost is virtually zero. This last is not necessarily true when using core-transistor logic, or *two's* complement notation.

7. Expansion of memory addressing

The AGC's 12-bit address field is insufficient for specifying directly all the registers in its memory. This predicament seems increasingly to afflict most computers, either because indirect addressing is assumed as a necessary evil from the start or, as was our case, because our earliest estimates of memory requirements were wrong by a factor of two or three. The method of indirect addressing we arrived at uses a bank register MB, but with an important modification: the 3-bit number stored in SIB has no effect unless the address is in the range (octal) 6000 to 7777. The MB register contents are not interpreted as higher-order bits of the address; they are interpreted as integers which specify which bank of 1024 words is meant in the event of the address part of the instruction being in the ambiguous range. The over-all map of memory is shown in Table 2. The unambiguous, fixed memory addresses domain has come to be known as "fixed-fixed."

It is interesting that this method of extending the addressing capability was not the result of trying to improve upon more conventional methods, but was almost a consequence of the physical difference between fixed and erasable memory. Since all data other than constants are concentrated in the erasable memory, these had to be exempt from modification by the MB register. An alternative arrangement, whereby only the addresses of instructions (as opposed to the addresses within an instruction word) are modified, would be deficient in that it would allow only instructions to be stored in banks; there would be no way to refer to constants stored in banks, or to use bank addresses to store arguments of arithmetic operations. The possibility of using two bank registers is worthy of serious consideration [Casale, 19621, but it did not occur to us.

Table 2 Address part of an instruction word

(Decimal)	
0–3071	Fixed and erasable memory: unambiguous addresses.
3072–4095	Fixed memory, ambiguous address. Contents of MB used to resolve the ambiguity. Up to 32 such banks are possible.

In addition to the addresses in erasable, it is necessary to exempt the addresses of interrupting programs (*i.e.*, the addresses to which a program interrupt transfers control) from the influence of the MB register. It was clear that it would be valuable to have a large body of unambiguous addresses for use in executive and dispatcher programs.

The most frequent and critical applications of bank changing are in the AGC's interpretive mode. Most of the programs relevant to navigation are written in a parenthesis-free pseudocode notation for economy of storage. An interpretive program executes these pseudocode program by performing the indicated data accesses and subroutine linkages.

The format of the notation permits two macrooperators (*e.g.*, "double-precision vector dot product") or one data address to be stored in one AGC word. Thus data addresses appear as full 15-bit words, potentially capable of addressing up to 32,768 registers. Each such address is examined in the interpreter and the contents of the bank register are changed if necessary; preparation is also made for subsequent return if a subroutine call is being made.

The structure of the interpretive program, and its relationship to the computer characteristics discussed in this paper will not be taken up here except to point out that parenthesis-free notation is particularly valuable in a short-word computer such as the AGC. It permits a very substantial expansion of the address and pseudo-operation fields without sacrificing efficiency in program storage [Muntz, 1962].

The conversion of a 15-bit address into a bank number and an ambiguous 12-bit address is as follows: the top 5 bits correspond directly to the desired bank number. The remaining lower-order 10 bits. logically added to octal 6000, form the proper ambiguous address. If the 15-bit address is less than octal 6000, however. the address is in erasable or fixed-fixed memory. In this case the logical addition of octal 6000 is suppressed.

It is possible to have a program in one bank call a closed subroutine in another bank, and then have control returned to the proper place in the bank of origin. This is done by means of a short bank switching routine which is in fixed-fised memory.

One potential awkwardness about this method of extending memory addresses is the possible requirement for a routine in one bank to have access to large amounts of data stored in another. There are many programming solutions to this problem. obviously at a cost in operating speed; a better solution would be to have two bank registers. No problems of this nature have yet materialized. however.

References

AlonR63; AlonR60; AlonR61; AlonR62; BeckF61; CasaC62; EnglW62; HopkA63; MuntC62; RichR55; WaleW62; Proc. *Conf. Spaceborne Computer Eng.;* Anaheim, Calif., Oct. 30–31, 1962.

APPENDIX 1 BACKGROUND FOR AGC DESIGN

Name, date completed	Memory *size* (F = *fixed* E = erasable)	*Number of bits*	*Number of instructions*	*Purpose of design*	Features *incorporated.* at this stage
MOD 1, 1960	F:448 E: 64	11 and parity	4 plus involuntary	Feasibility Prototype	Counter increments, Interrupts, Core-Transistor Logic, Pulse rate outputs, Editing registers, Wired-in fixed memory, Interpretive programs.
MOD 2, not built	about 4000 total	23 and parity	16 plus indirect	Unmanned Space Probe	"Extended Operation" subroutine linkages (only instance).
MOD 3S, 1962	F 3584 E: 512	15 and parity	8	Earth Satellite	Modified one's complement, Parallel adder, Addressable central registers.
MOD 3C, 1962	F: greater than 10^4 E: greater than 10^3	15 and parity	8 and involuntary	Apollo Guidance	CCS, INDEX. MULTIPLY instructions. Overflow counter, Bank switching.
AGC. 1963	F: greater than 10^4 E: greater than 10^3	15 and parity	11 and involuntary	Apollo Guidance	DV, SU. MSK instructions. Editing memory buffer. All transistor NOR logic instead of core-transistor logic, Extracodes. Parenthesis-free interpreter.