Massachusetts Institute of Technology
Instrumentation Laboratory
7 October 1963

Critique of IBM Apollo Study Report - IBM #63-928-129 - 1 Oct 1963

The following is a collection of comments on the use of the Saturn V guidance computer as a back-up for the MIT AGC. Although it is not intended to be complete, the preliminary evidence presented makes clear that the IBM computer is neither suitable nor capable of performing the Apollo mission as it has been defined. It is indeed unfortunate that the portion of the BELLCOMM staff which has been an enthusiastic supporter of the IBM computer did not discern these obvious deficiencies in the many months of their association with the IBM and MIT staffs.

One of the purposes of this critique is to refute the following summary on programming requirements quoted from the IBM report.

"The storage capacities are roughly equivalent, the AGC-4 being more economical in subroutine linkage and double precision computations, while the Saturn is more efficient for the following reasons or areas:

(1) nonsubroutine computation

(2) less double precision computations are required

(3) a larger basic instruction set"

I. PROGRAMMING CONSIDERATIONS

1. Storage Capacity & Organization

(a) Size

The IBM computer memory consists of two sets of three modules each; the second set is redundant with the first to gain the required reliability missing in the erasable ferrite plane used. This leaves three modules of data, each module containing 4096 words of 28 bits each. Two of these 28 bits are parity bits and will be ignored in the discussion to follow. This yields an information content of 319,488 bits.

The AGC computer memory consists of 1024 words of erasable memory, 2048 words of non-switchable (or fixed-fixed memory), and 21 switchable banks of 1024 words each. Each word contains 16 bits, one of which is a parity bit and will be ignored. This yields a total information content for AGC of ~~390,216~~ 384,000 bits. (About ~~389,000~~ 384,000 of these bits are indestructable in a wired rope configuration.)

Comparison of these numbers indicates that the IBM computer has an information content which is about 20% less than AGC.

(b) Module Organization

As pointed out above, the IBM computer memory is logically divided into three distinct modules of 4096 words each. Program control is transferred from module to module by the HOP order code with an appropriate setting of a HOP constant. Data addresses cannot cross module boundaries unless the reference is to the Accumulator register or location 775, the PQ register (a delay line).

To use subroutines which cross modules, the return address must be passed over in one of these registers; the other register can be used to pass over a

single precision argument. Even with the proposed engineering change to allow automatic generation of HOP constants for return addresses, only a double precision quantity may be passed over conveniently. Vector subroutines, for example, must be duplicated in each module.

No such restrictions apply to the AGC because 3072 words are addressable from any location in the machine while the IBM computer supplies only 2 such words.

(c) Sector Organization

Each module of the IBM computer memory is further divided into sectors of 256 words each. At any particular point in time an instruction can address:

(1) the 256 word residual memory, identical for all addresses in the same module.

(2) the currently selected 256 word data sector.

(3) the currently selected 256 word instruction sector.

To select a location not contained in these 768 words requires a HOP instruction.

In the case of the AGC, at any particular point in time an instruction can address:

(1) the 1024 word erasable memory (including specials, centrals, I/O)

(2) the 2048 word fixed-fixed memory, identical for all addresses.

(3) the currently selected 1024 word switchable bank.

To select a location not contained in these 4096 words requires a TS to the BANK register.

It is not easy to assess the wasted storage required to overcome the fact that only about 20% as many locations are directly addressable in the IBM computer as in the AGC. In fact, the 256 words in the instruction sector can only be addressed by TRA, TMI, and TNZ. Further, the sector registers are not directly addressable and a subroutine can never HOP without destroying the present contents of the data sector register.

It is estimated that the duplication of numerical constants, HOP constants, execution of HOP'S to switch data sector, and inability to save sector registers will consume 10-15% of the total memory.

2. Subroutine Computations

The principal advantage of performing computer calculations by means of subroutines is to minimize the amount of storage capacity required at the expense of some increase in computing time. For example, if it is required to perform a square root computation more than once, it is clearly better, from the point of view of conserving storage capacity, to have only one square root routine in the computer which is called each time a square root is required than to have a separate copy of the square root routine each time it is needed.

(a) Because of the nature of the Apollo guidance problem, subroutine computation is the only practical method of conserving storage capacity. Many computations, such as the square root example, just cited, are required which would consume storage at an exorbitant rate if they were programmed each time they were used.

On the other hand, subroutine computation is, perhaps, not needed for the mechanization of the so-called Saturn "path adaptive" guidance mode for which the IBM computer was designed. The fundamental computation is simply polynomial evaluation. If an explicit guidance technique were employed for Saturn, it would be extremely difficult to implement without the use of subroutines.

(b) Subroutine computation is facilitated in the AGC by means of a programmed interpreter. For example, the following program will perform the vector sum of two double precision vectors A and B to produce a double precision vector C.

```
VAD     0
        A
        B
STORE   C
```

Required Storage 60 bits

The same computation performed in the IBM computer is programmed

```
CLA     A
ADD     B
STO     C
CLA     A + 1
ADD     B + 1
STO     C + 1
CLA     A + 2
ADD     B + 2
STO     C + 2
```

Required Storage 117 bits

(c) The IBM computer is not well adapted to efficient use of subroutines. For example, if a subroutine were provided for vector addition, more storage would be required to call the subroutine then that required to perform the computation directly. A program to call a vector add subroutine, or any vector subroutine for that matter, would be as follows:

```
CLA     ADRESA
STO     VCAADR
CLA     * + 2
HOP     VCALINK
HOPCON  * + 1
CLA     ADRESB
STO     VADADR
CLA     * + 2
HOP     VADLINK
HOPCON  * + 1
CLA     CADRES
STO     VTSADR
```

```
CLA      *+2
HOP      VTSLINK
HOPCON   *+1
```

Required storage 312 bits

(d) IBM has recognized their deficiency in efficient subroutine management and has suggested an engineering change. With this change implemented, the program to call the vector add routine would be

```
CLA      ADRESA
HOP      VCALINK
CLA      ADRESB
HOP      VADLINK
CLA      CADRES
HOP      VTSLINK
```

Required storage 78 bits

It should be noted, however, that because of the modular structure of the computer, vector subroutines are impractical. A copy of each vector subroutine would be required in each module because of the extreme difficulty of exchanging data between modules. For example, to move a vector across module boundaries requires 1230 μ sec and 156 bits of program storage.

(e) Finally, to complete the comparison, we have programmed for both the AGC and IBM computers, the integration routine needed to up-date position and velocity vectors during all phases of powered flight. We have tried to exploit the characteristics of the IBM computer to produce an efficient program. Therefore, it has been assumed that the components of each vector are stored in the same relative position of three different memory sectors. The AGC program requires 375 bits of storage while the IBM computer requires 1001 bits for the same program or 2.7 times the storage required for the AGC. The AGC computer produces a twenty-eight bit result.

The algorithm for a spherical gravitational field may be summarized as follows

NAVIGATION EQUATIONS

Spherical Gravitational Field

Quantities in storage:

$$\underline{R}_k,\ \underline{V}_k,\ \frac{h}{2}\underline{G}_k,\ \underline{W}_{k+1}$$

Equations:

$$\underline{R}_{k+1} = \underline{R}_k + h\underline{V}_k + \frac{h^2}{2}\underline{G}_k + \frac{h}{2}\underline{W}_{k+1}$$

$$RS = \underline{R}_{k+1} \bullet \underline{R}_{k+1}$$

$$R = SQRT\ (RS)$$

$$\frac{h}{2}\underline{G}_{k+1} = \left(-\frac{h}{2}\mu / R\ RS\right)\underline{R}_{k+1}$$

$$\underline{V}_{k+1} = \underline{V}_k + \underline{W}_{k+1} + \frac{h}{2}\underline{G}_k + \frac{h}{2}\underline{G}_{k+1}$$

The programs for the two computers are given below in detail.

```
AVERAGEG    STO     EXITHOP
            HOP     HOPSET1
AVG1        CLA     WK
            SHF     R1
            ADD     HGK/2
            ADD     VK
            MPH     H
            ADD     R
            STO     R
            MPY     R
            HOP     THISSEC1
AVG4        CLA     HOPWD1
            ADD     ONE
            STO     HOPWD1
            CLA     PQ
            ADD     DOTSUM
            STO     DOTSUM
HOPWD1      HOP     HOPSET1
AVG2        CLA     DOTSUM
            STO     SQRTARG
            CLA     *+2
            HOP     SQRTLINK
            HOPCON  *+1
            CLA     SQRTANS
            MPY     DOTSUM
            CLA     -MUH/2
            NOOP
            NOOP
            DIV     PQ
            HOP     THISSEC2
```

```
AVG5        CLA         HOPSET1
            STO         HOPWD1
            CLA         HOPSET2
            STO         HOPWD2
            NOOP
            NOOP
            NOOP
            CLA         PQ
            STO         DOTSUM
            HOP         HOPSET2
AVG3        CLA         R
            MPY         DOTSUM
            CLA         HGK/2
            ADD         W
            ADD         V
            STO         V
            CLA         PQ
            STO         HGK/2
            ADD         V
            STO         V
            HOP         THISSEC3
AVG6        CLA         HOPWD2
            ADD         ONE
            STO         HOPWD2
            HOP         HOPSET2
HOPSET1     HOPCON      AVG1, XCOMP
            HOPCON      AVG1, YCOMP
            HOPCON      AVG1, ZCOMP
            HOPCON      AVG2, XCOMP
HOPSET2     HOPCON      AVG3, YCOMP
            HOPCON      AVG3, ZCOMP
EXITHOP     (     exit hop con          )
```

```
SQRTLINK        HOPCON      SQRT, XCOMP
THISSEC1        HOPCON      AVG4, AVG4
THISSEC2        HOPCON      AVG5, AVG5
THISSEC3        HOPCON      AVG6, AVG6
```

AGC Computer Program for Integration During Acceleration Flight

```
AVERAGEG        VXSC        0
                            W
                            SCFTR
                NOLOD       1
                VAD         VAD
                            HG
                            V
                NOLOD       1
                VXSC        VAD
                            H9
                            R
                STORE       R
                NOLOD       1
                DOT         ROUND
                            R
                NOLOD       2
                SQRT        DMPR
                BDDV        VXSC
                            -
                            HMUE
                            R
                STORE       HG
                NOLOD       1
                VAD         VAD
                STORE       V
```

3. Double Precision Computations

(a) The basic data word length in the IBM computer is 25 bits plus sign. However, with overflow protection this precision is automatically reduced to 24 bits. Also the multiply instruction produces only 23 bit product. In contrast, the AGC in the double precision mode has 28 bits plus sign with automatic overflow protection and produces a full 42 bit product.

To check the adequacy of the IBM word length we have performed the following experiment. A complete simulation of midcourse guidance and navigation procedure was made using our Monte Carlo program.

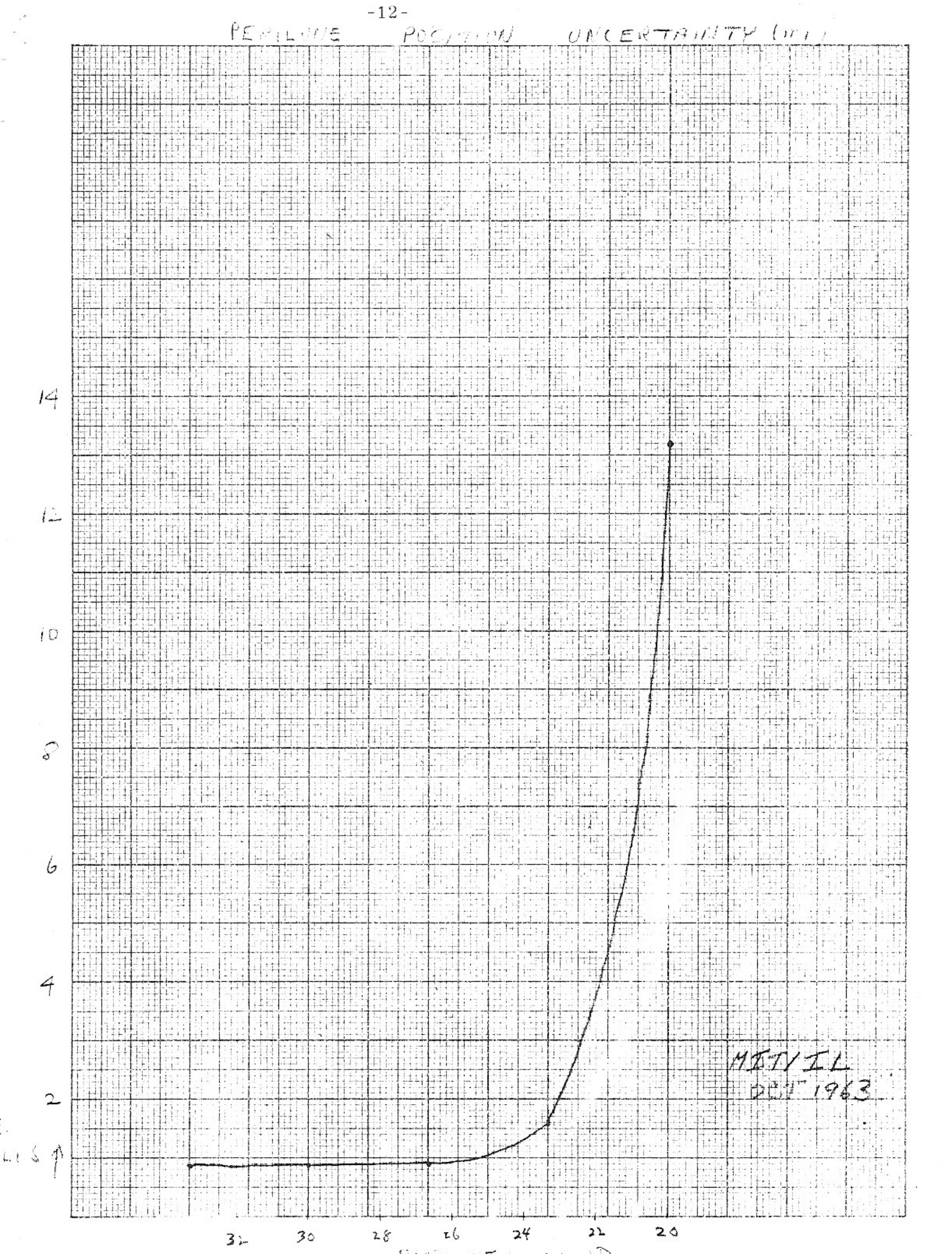
Two trajectories are calculated: the true position and velocity and the estimated value as computed by the guidance computer. A set of five circumlunar runs were made all with the same random numbers. At the end of each integration time step the estimated position and velocity vectors and the error matrix were fixed-point truncated. All intermediate calculations were made with full ten digit floating point numbers. Position and velocity uncertainties at perilune and at entry are shown in the following graphs as functions of the number of bits remaining after truncation.

These results are, of course, optimistic since the guidance computer does all computations with the limited word length. In our simulations it should be emphasized that all computations were performed with the full ten digit floating point and only the results truncated.

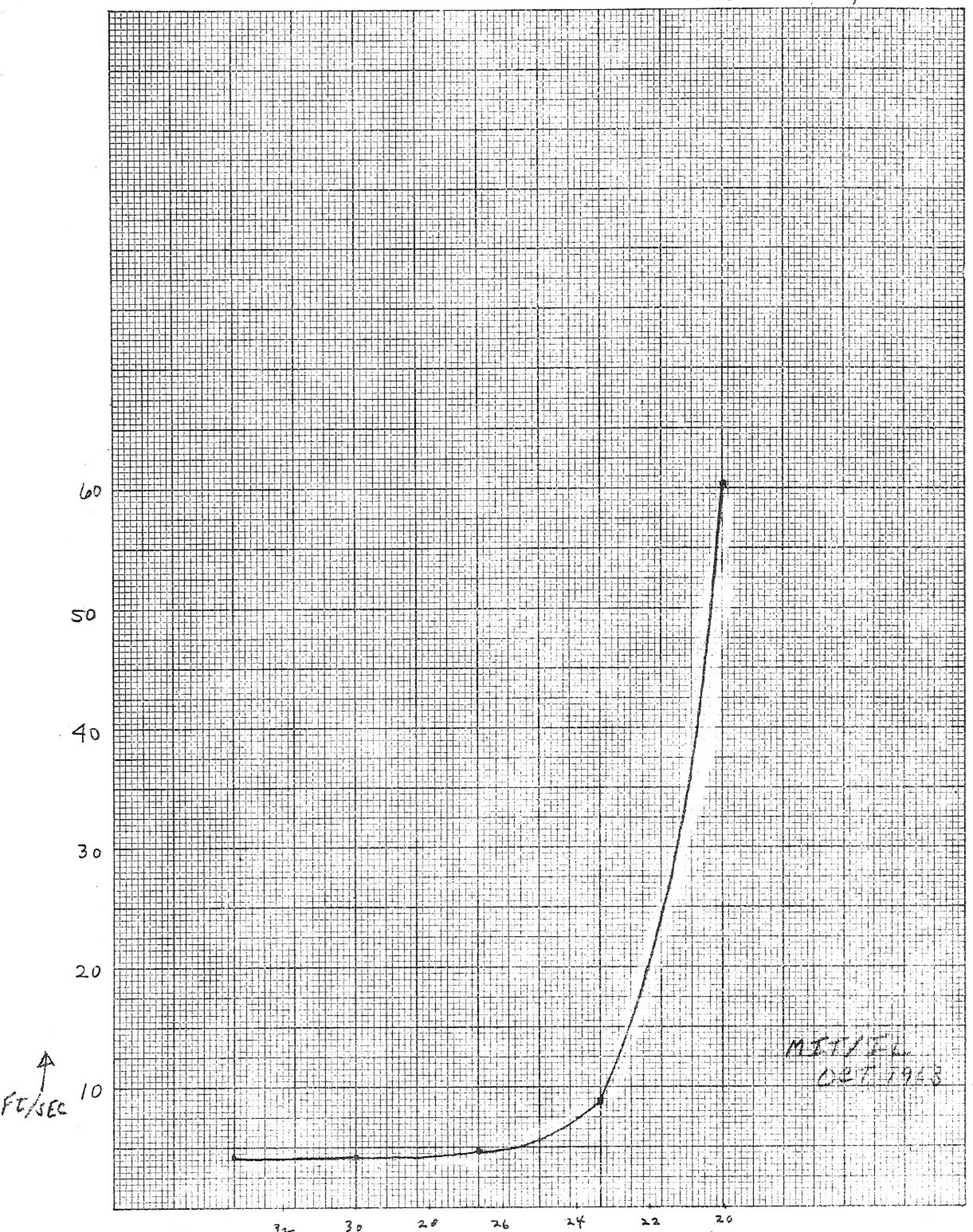
Ames Research Center completed a similar analysis two years ago. Their results are shown also in graphical form. They confirmed the fact that accuracy is degraded by at least a factor of five from that attainable with AGC double precision arithmetic. As a result it seems highly unlikely that entry corridor requirements could be achieved with the IBM computer in single precision.

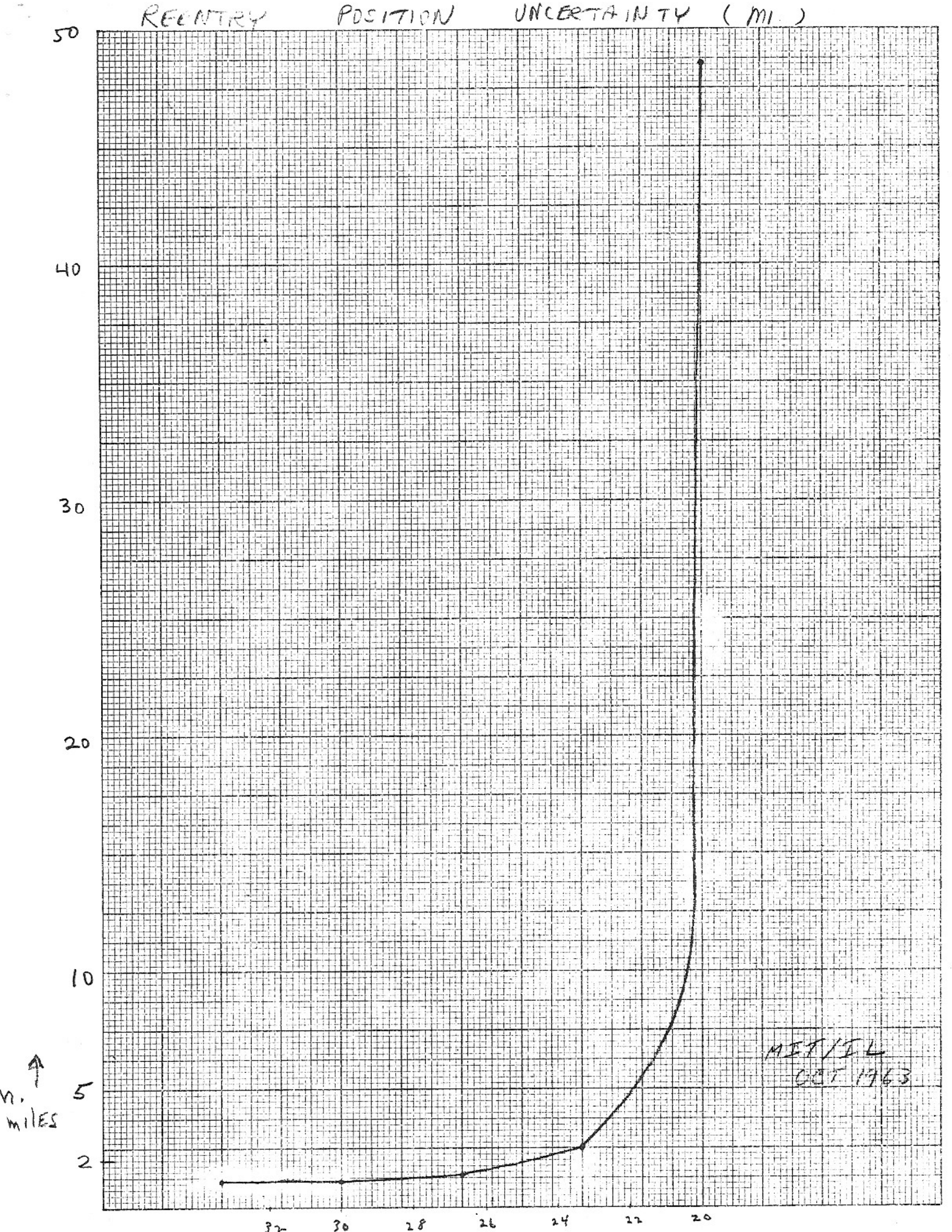
Even if the accuracies read from these graphs were tolerable it is extremely dangerous to be operating on the steep part of the curves. A slight loss of accuracy can result in enormous degradation in performance.

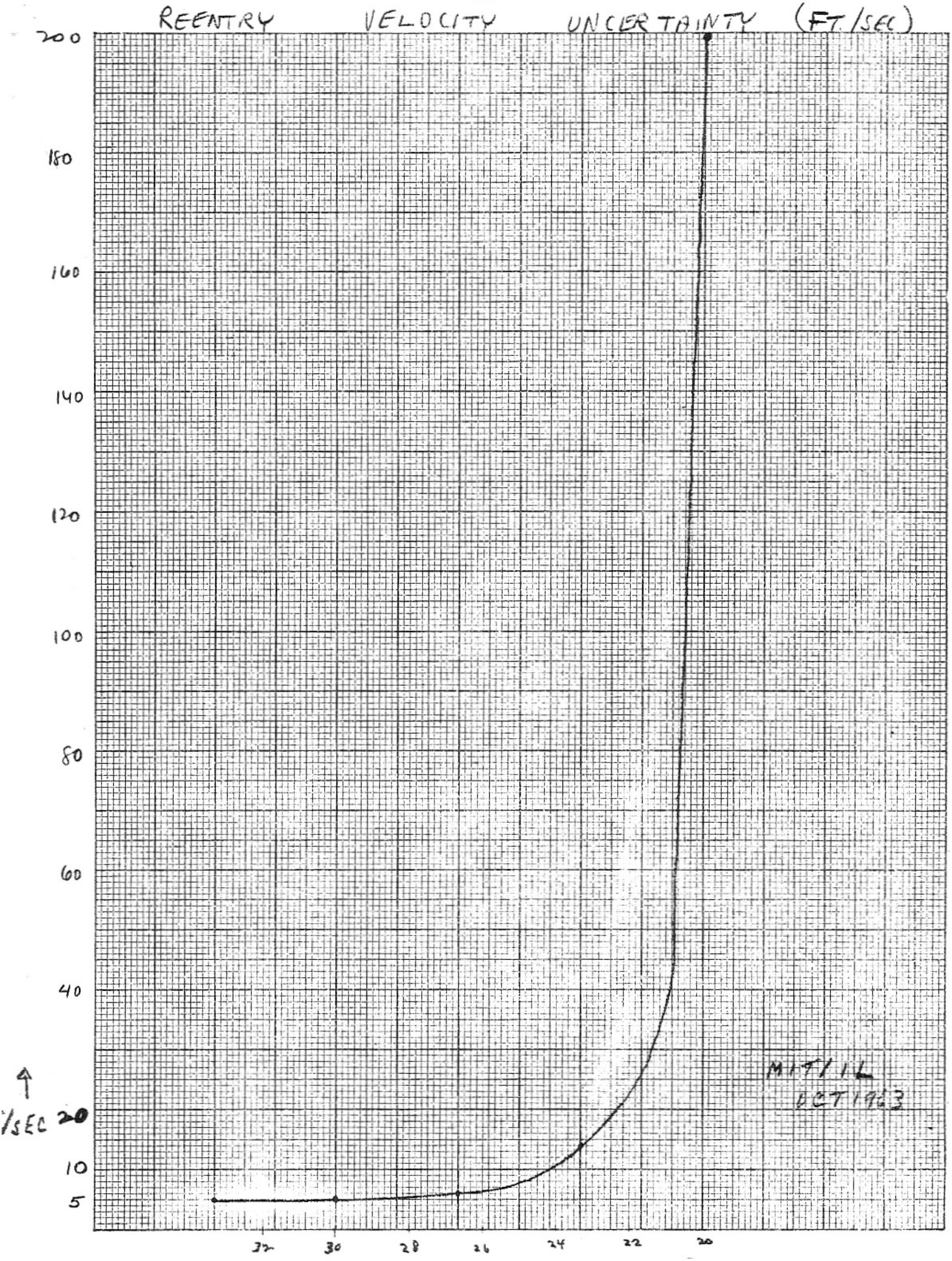
(b) The more frequent need for double precision operations in the AGC is not necessarily a severe penalty. For instance, the equation $A + B = C$ is done faster in <u>double</u> precision by AGC than in <u>single</u> precision by the IBM

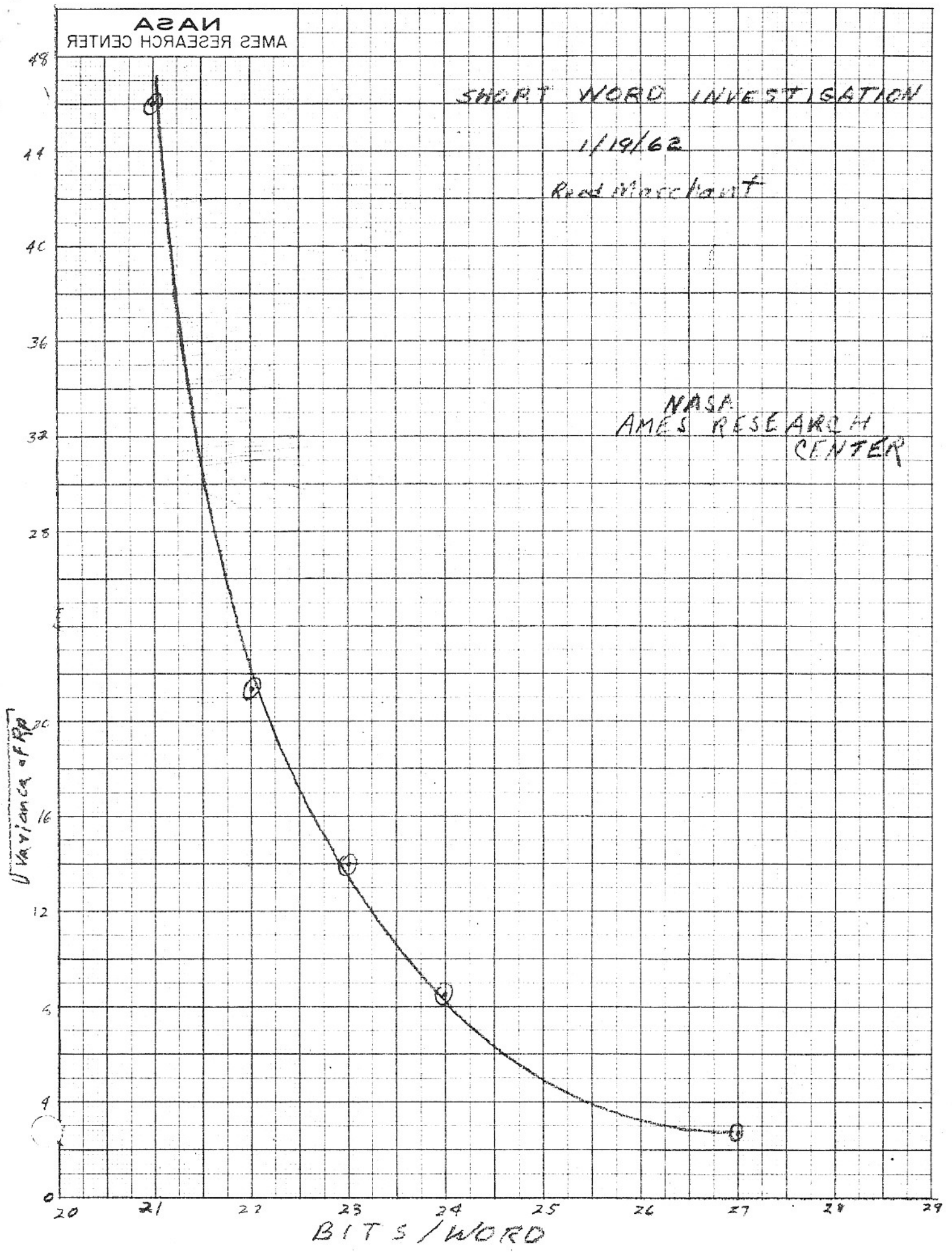

PERIGEE POSITION UNCERTAINTY (KM)
14
12
10
8
6
4
2
1
32
30
28
26
24
22
20
MITRE
DEC 1963

PERILUNE VELOCITY UNCERTAINTY (FT/SEC)




REENTRY POSITION UNCERTAINTY (MI.)
50
40
30
20
10
5
2
n. miles
32
30
28
26
24
22
20
MIT/IL
OCT 1963


REENTRY VELOCITY UNCERTAINTY (FT/SEC)
200
180
160
140
120
100
80
60
40
20
10
5
↑
/SEC
32
30
28
26
24
22
20
MIT/IL
OCT 1963


NASA
AMES RESEARCH CENTER
SHORT WORD INVESTIGATION
1/19/62
Read Miscellaneous
NASA
AMES RESEARCH
CENTER
48
44
40
36
32
28
20
16
12
8
4
0
20
21
22
23
24
25
26
27
28
29
BITS/WORD

computer.

AGC (A in fixed)		IBM	
CAF	A + 1	CLA	A
AD	B + 1	ADD	B
TS	C + 1	STO	C
CAF	ZERO		
AD	A		
AD	B		
TS	C		
28 bits of precision in 168 μ sec		25 bits of precision in 246 μ sec	

Also, the AGC double precision multiply, yielding 42 bits in 1 msec, runs only 20% longer than IBM's double precision addition (840 μ sec average).

As another example, AGC performs square root in double precision (28 bits) in about 2.4m sec minimum, while the IBM computer performs square root in single precision (22 or 23 bits) in about 3.9 msec minimum. Thus AGC gets 125% of IBM's precision in 60% of the time.

The IBM program developed for this comparison follows:

```
SQRT          STO       RETURN
              CLA       ZERO
              STO       NORMCNT
              CLA       ARG
NORMTEST      AND       HIGH3
              TNZ       NORMDUN
              CLA       NORMCNT
              ADD       ONE
              STO       NORMCNT
              CLA       ARG
              SHF       L2
              STO       ARG
              TRA       NORMTEST
```

```
HIGH3       DEC     -.75
1/2         DEC      .5
SLOPELO     DEC      .4162
BIASLO      DEC      .1487
SLOPEHI     DEC      .2942
BIASHI      DEC      .2046
NORMDUN     AND      1/2
            TNZ      ARGHI
            CLA      ARG
            MPY      SLOPELO
            SHF      R1
            STO      ARG
            CLA      BIASLO
            ADD      PQ
            TRA      NEWTON
ARGHI       CLA      ARG
            MPY      SLOPEHI
            SHF      R1
            STO      ARG
            CLA      BIASHI
            ADD      PQ
NEWTON      STO      BUF
            CLA      ARG
            DIV      BUF
            ADD      ZERO
            ADD      ZERO
            ADD      ZERO
            ADD      ZERO
            ADD      ZERO
            ADD      ZERO
            ADD      ZERO
            CLA      PQ
            SHF      R1
            ADD      BUF
            STO      BUF
```

```
                    CLA         ARG
                    DIV         BUF
                    ADD         ZERO
                    ADD         ZERO
                    ADD         ZERO
                    ADD         ZERO
                    ADD         ZERO
                    CLA         BUF
                    SHF         R1
                    ADD         PQ
                    STO         ARG
CLANORC             CLA         NORMCNT
                    TNZ         POSTSQRT
                    CLA         ARG
                    HOP         RETURN

POSTSQRT            SUB         ONE
                    STO         NORMCNT
                    CLA         ARG
                    SHF         R1
                    STO         ARG
                    TRA         CLANORC
```

The calling sequence for this (and indeed, any unary) subroutine is:

```
                    CLA         X
                    STO         ARG
                    CLA         REHOP
                    HOP         SQRTLINK
RETURN              CLA         ARG
```

for which two HOP constants are required:

```
REHOP               HOPCON      RETURN
SQRTLINK            HOPCON      SQRT
```

4. Basic Instruction Set

It is very misleading to judge a computer solely on its ability to perform arithmetic. In fact, numerical computations generally account for only a small fraction of the actual program storage and computation time. When the AGC and IBM instruction set for non-arithmetic operations are compared as to their versatility and speed, the AGC order code is far more powerful and can accomplish anything in less time than IBM computer. In particular, the AGC has indexing capability, easy subroutine linkage and complete freedom to interrogate output registers. None of these features are present in the IBM machine and cannot be provided without a significant redesign.

In the two tables which follows the AGC and IBM instructions are compared. It is seen that on the average the AGC counterpart of the IBM instructions requires 70% of the program executive time. On the other hand, for the IBM computer to duplicate AGC instructions requires over 13 times the computation time. These facts markedly reduce the feasibility of a programmed interpreter for the IBM computer to conserve storage in a manner similar to the AGC. Indeed, equivalent coding of representative sections of a programmed interpreter of the type used in the AGC indicates a five to one speed disadvantage.

IBM Operation Code Equivalents
for Non-Arithmetic Operations

IBM Code		IBM	AGC	Speed Ratio
HOP		82 μ sec	60 μ sec	.73
TRA		82 μ sec	12 μ sec	.15
TMI		82 μ sec	72 μ sec	.88
TNZ		82 μ sec	84 μ sec	1.02
SHF	R1, L1	82 μ sec	48 μ sec	.59
	R2, L 2	82 μ sec	72 μ sec	.88
AND		82 μ sec	24 μ sec	.30
CLA		82 μ sec	48 μ sec	.59
STO		82 μ sec	24 μ sec	.30
PIO		82 μ sec	24 μ sec	.30
XOR		82 μ sec	168 μ sec	2.05

Table 2

AGC Operation Code Equivalents
for Non-Arithmetic Operations

AGC Code	AGC	IBM	Speed Ratio
TC	12 μ sec	82 μ sec	6.8
CCS	24 μ sec	246 μ sec	10.3
INDEX	24 μ sec	573 μ sec	23.9
XCH	24 μ sec	492 μ sec	20.5
CS	24 μ sec	164 μ sec	6.8
TS (no overflow protection)	24 μ sec	82 μ sec	3.4
TS (with overflow protection)	24 μ sec	738 μ sec	30.8
MASK	24 μ sec	82 μ sec	3.4

Table 1

II. DATA ADAPTER

The AGC counter registers contain a full 15 bits and the scale factors for G&N devices have been chosen accordingly. 2^{-15} of a circle is 40 secs of arc; 2^{-13} of a circle is 2.6 min of arc which is inadequate for guidance accuracy requirements. The IBM data adapter allows only 13 bits. Therefore, in most cases, the true counters must be kept in memory outside the DA with the 13 bit DA quantities serving as incremental quantities. For example, the CDU shaft encoder has 2^{15} states and the shaft position can only be stored as a 15 bit quantity in the central processor. This represents a duplication in storage as well as a programming inconvenience.

To change the DA counters from 13 to 15 bits would involve changing the word length in the IBM computer or almost doubling the number of delay lines in the DA.

The AGC counters may be read by any instruction which, along with INHINT and RELINT, eliminates the need for a PIO type instruction. This counter reading is achieved in 24μsec. On the other hand, the IBM computer's DA takes a number of instructions to do the same task. Thus, to complete our example, equivalent programs for reading the CDUX counter into the accumulator is given below:

AGC		IBM	
CS	CDUX	PIO	READCDUX
		PIO	READFAZE
		AND	FAZEMASK
		TNZ	* - 2
		PIO	READCOD
		ADD	CDUX
		AND	LOW15
		STO	CDUX

Storage = 15 bits	Storage = 104 bits
Time = 24 μ sec.	Time = 656 μ sec.

This procedure is required for CDUX, CDUY, CDUZ, OPTX, OPTY, TRKRX, and TRKRY. This procedure, 27 times slower than that required in the AGC, will exert the biggest load during LEM operations when computation time is most needed.

We concur with IBM that Servicing of Telemetry, Displays, Keyboard, etc. requires about 5% of AGC time vs 25% of IBM time.

III. INTERRUPT

The AGC has 5 distinct interrupt sequences. This allows rapid isolation of the cause of interrupt. The IBM computer has only one distinct interrupt. This means a lengthy scan must be performed to determine what has initiated the interrupt.

The time spent to initiate an interrupt on the IBM computer is about 25% longer than necessary (cf. Vol. 1, pp. 4-15) because of poor design of the instruction repertoire. In any case, it is at least 4 times longer than the corresponding AGC sequence.

The address used to initiate the interrupt program is taken from register 776 of the module being used when the interrupt occurs. Thus, if the interrupt address is to be changed, it must be changed in all three modules. This further duplication of program and increase in execution of interrupt task is debilitating.