Kolsky
Memo 4

May 19, 1958

PROJECT 7000

FILE MEMORANDUM

SUBJECT: Branching on Arithmetic Results in SIGMA

1. **Introduction:**

The asynchronous organization of SIGMA allows many of the components of the Computer System to be operating at the same time on different jobs and thus by overlapping greatly increases the overall efficiency of the system.

Unfortunately this organization also has its drawbacks. In particular, one of the curses of the non-sequential preparation and execution of instructions is that if there is a Branch in the problem code it spoils the smooth flow of instructions to the Indexing Arithmetic Unit. Any branch in a program will cause some delay, but the ones which hurt the most are the branches on Arithmetic results which cannot be detected by the Indexing Arithmetic Unit in advance.

This paper reports on an attempt to study Arithmetic Result Branches in the SIGMA system using the SIGMA Timing Simulator (Ref: Project 7000 File Memos by Cocke and Kolsky, Dated February 6 and March 12, 1958.) The time losses to the system are evaluated in several typical cases and a recommendation concerning how the branches should be handled by the Indexing Arithmetic Unit is made.

2. **Ways in which Arithmetic Result Branches can be Handled:**

There are two fundamental ways in which branches on Arithmetic Unit results can be handled by the computer:

1) The computer can stop the flow of instructions until the Arithmetic Unit has completed the preceeding operation so that the result is known,then fetch the next correct instruction. This places a delay on every AU result Branch whether taken or not.

2) The computer can "guess" which way the branch is going to go before it is taken and proceed with fetching and preparing the instructions along one path with the understanding that if the guess was wrong these instructions must be discarded and the correct path taken instead.

Under the second alternative there are four possible ways in which the guessing can be made. The branches in question are in-d~~uction~~icator branches on the Arithmetic Unit result indicators. These operations have a modifier which allows the branch to be taken either if the specified indicator is on or off. Since one can guess that the indicator is on or off for each, the four combinations are:

Case	Name	Operation	Guess	Assumed Result of Operation
I	NN-FF	Ind Branch on	Ind on	branch
		off	off	branch
II	NF-FN	Ind Branch on	Ind off	no branch
		off	on	no branch
III	NN-FN	Ind Branch on	Ind on	branch
		off	on	no branch
IV	NF-FF	Ind Branch on	Ind off	no branch
		off	off	branch

3. **Simulation Results:**

To study the effects of wrong-way branches on the SIGMA Timing Simulator, The Monte Carlo Branching Code was chosen as the guinea pig. (For a description, see the earlier memos). The code was re-written so that every Arithmetic result branch was a wrong guess and again so that every one was guessed correctly. (Note that neither of these extremes is actually possible in a program with branches unless they are essentially unconditional.)

Several runs were made varying the instruction memory speed and the AU and IAU times. The results for the all-wrong, the all-right, and the regular (NF-FN) Cases are shown in graphs one and two. The regular case had two wrong branches out of thirteen encountered in one loop of the program which consists of fifty-nine operations executed per loop.

By examining the timing charts drawn by the Simulator for many of the individual branches, the average time delays listed in Table 1 were derived.

Table 1: Average Time Delay per Individual Branch.

assumed	guessed	For 0.6 us Instr. Mem.	For 2.0 us Instr. Mem.
no branch	right	0 us	0 us
no branch	wrong	2.5 us	3.2 us
branch	right	1.5 us	3.2 us
branch	wrong	3.7 us	4.8 us

For "Standard" Times (AU = 0.64 us, IAU = 0.6 us)

If one takes the actual times to complete the problem in each case and divides the total delay by the number of wrong-way branches, one obtains the times listed in Table 2. The approximate delay due to the memory interferences, etc. caused by starting the processing of the wrong instructions, can be estimated by comparing the times in Table 1 with those in Table 2. These interference times are listed in Table 2.

Table 2: Average Time Delay in Total Problem per Wrong-way Branch.

	0.6 us Instr. Mem	2.0 instr. Mem.
For "Standard Times" (AU = 0.64 us, IAU = 0.6 us	2.9 us	3.5 us
For "Recommended Times"(AU= 1.09 us, IAU= 0.9 us	3.6 us	4.3 us
Extra Delay due to memory Interferences	~ 0.5 us	~1.0 us

Presumably if one holds up on every branch (Case O) the time loss will be about that of assuming no Branch and guessing wrong. (line 2 in Table 1.) If one guesses according to one of the four other cases, the time loss will depend on (1) the percentage of branches which are Br-ons, (2) the percentage of Br-ons which are actually taken, and (3) the percentage of Br-offs which are actually taken.

The calculation will be delayed by each branch taken even when they are guessed correctly, however, since we are interested in examining the additional time lost due to guessing wrong or holding up, the delays due to correct branching should be removed. The following times in Table 3 may be used to compute actual combinations of branches.

Table 3: Average Time Delay per Branch.

Computer Guessed	Should Have Guessed	0.6 us Instr. Mem.	2.0 us Instr. Mem.
Hold up	no branch	2.2 us	1.6 us
Hold up	Branch	2.5 us	3.2 us
no branch	no branch	0 us	0us
no branch	Branch	3.0 us	4.2 us
Branch	Branch	0 us	0 us
Branch	No branch	2.7 us	2.6 us

The temptation in evaluating the Individual cases is to assume 50% for all the combinations and essentially average the time losses. Actually, by examining a few problems superficially, I have found that considerably fewer than half the arithmetic result branches encountered in a code are actually taken. About 20% seem to be more typical. This seems to be due to the tendency of coders to think of the branches as being exceptional cases. They normally write the main flow of the code continuously and the exceptions elsewhere.

There seems to be a tendency to link indicators turning on with exception cases. In time this would result in fewer Br-ons being taken and more Br-offs being taken. These generalizations are admittedly uncertain mainly because very few relevant statistics are available.

There is also a "feedback" in such statistics because the way in which the machine guesses the branches will influence future programmers to write their codes to take advantage of the speed gain, so that the statistics of the future will be biased in favor of the system chosen now!

Table 4 compares the five cases for several assumed values of percentages. The last two lines are my guesses as to the averages to be expected.

Table 4: Average Time Delays per Branch for the Different Cases.

% Br-ons	% Br-ons taken	% Br-offs taken	Case 0 Hold-up	Case I NN-FF	Case II NF-FN	Case III NN-FN	Case IV NF-FF
	For 0.6 us. Instruction Memory						
50%	50%	50%	2.35 us	1.30 us	1.45 us	1.38 us	1.38 us
50%	20%	20%	2.26	2.14	0.54	1.33	1.42
80%	20%	80%	2.30	1.69	0.89	1.19	0.47
	For 2.0 us. Instruction Memory						
50%	50%	50%	2.40 us	1.00 us	1.80 us	1.40 us	1.40 us
50%	20%	20%	1.92	1.96	0.36	1.16	1.64
80%	20%	80%	2.11	1.45	0.94	2.22	0.10

4. Conclusions:

1) The performance variation in a problem with a lot of arithmetic data branching can vary by approximately ≠ 15% depending on the way in which the branches are handled.

2) Holding-up on every branch seems to be less desirable than any of the guessing proceedures.

3) It is very unlikely that one ever get fewer than 15% or more than 85% wrong-way branches regardless of his proceedure.

4) It seems possible to get a fairly low loss by picking Case IV, provided the percentages on the last line of Table 2 really are correct. However, if the percentages should be different, Case IV is much more sensitive to them than Case II.

5) To be really effective Case IV needs the existance of the indicators ≥ 0, ≤ 0 to make the distinction between off and on precise. At present one must code "Br-on$\leq$0", as "Br-off $>$0," so that the equating of "on" to "exceptional case" is spoiled somewhat.

6) The highest performance would be obtained if each branch had an extra "guess bit" which would permit the programmer to specify which way he estimates each branch will most likely go. This seems to be impossible in the present format schemes. It also would place a considerable extra burden on the programmer for the gains promised.

5. Recommendation:

Case II (NF-FN) should be adopted as the guessing scheme. This means that for any branch for which the IAU cannot compute the correct outcome, it should guess that the branch is not taken and proceed with the processing of the next instruction.

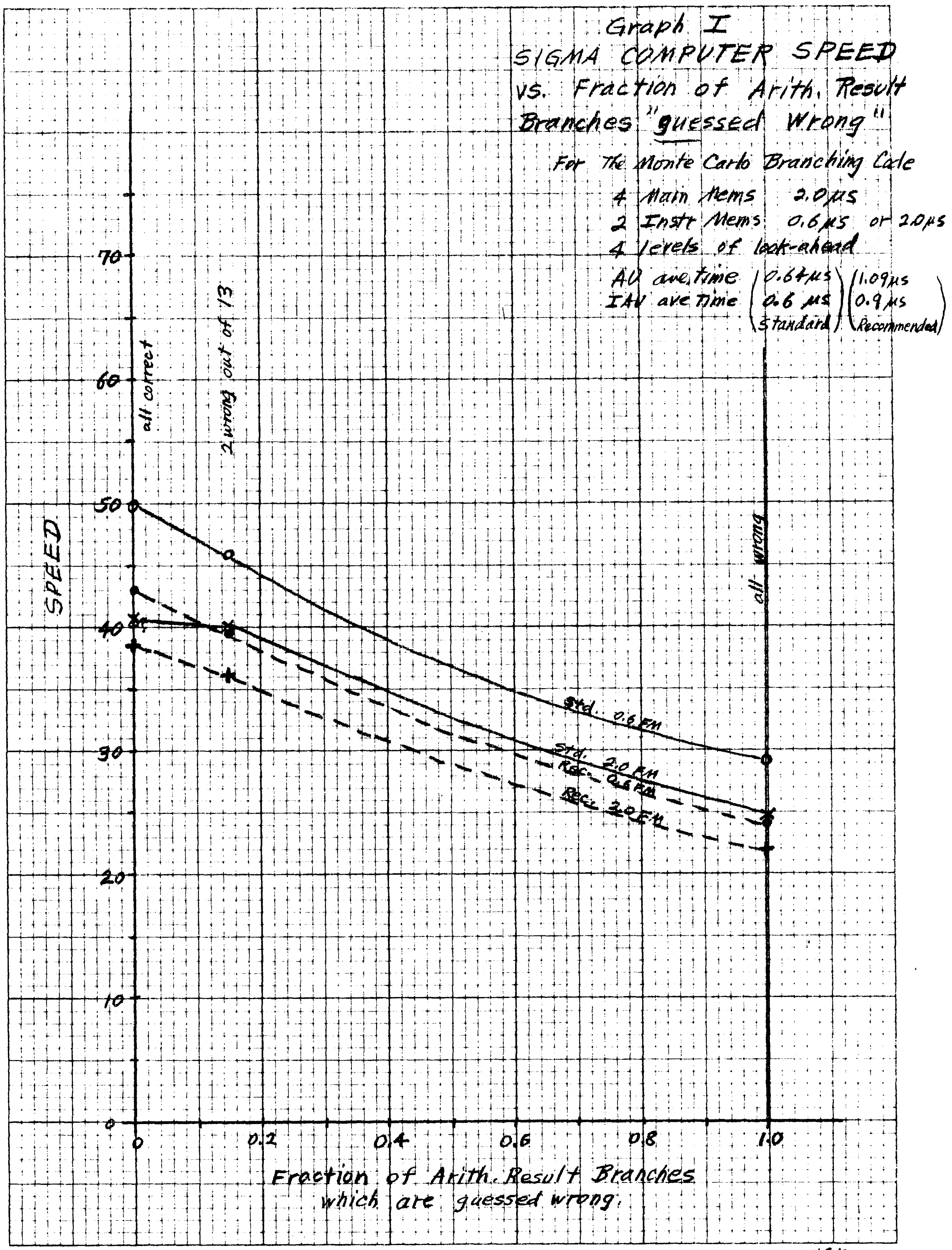
Case II was chosen over case IV because:

1) Its time loss is low (at least second best)
2) It does not require special controls for deciding whether to assume a branch is taken or not.
3) It does not require that new indicators be defined.
4) It should not confuse the programmer with complicated rules of coding the way Case IV might.

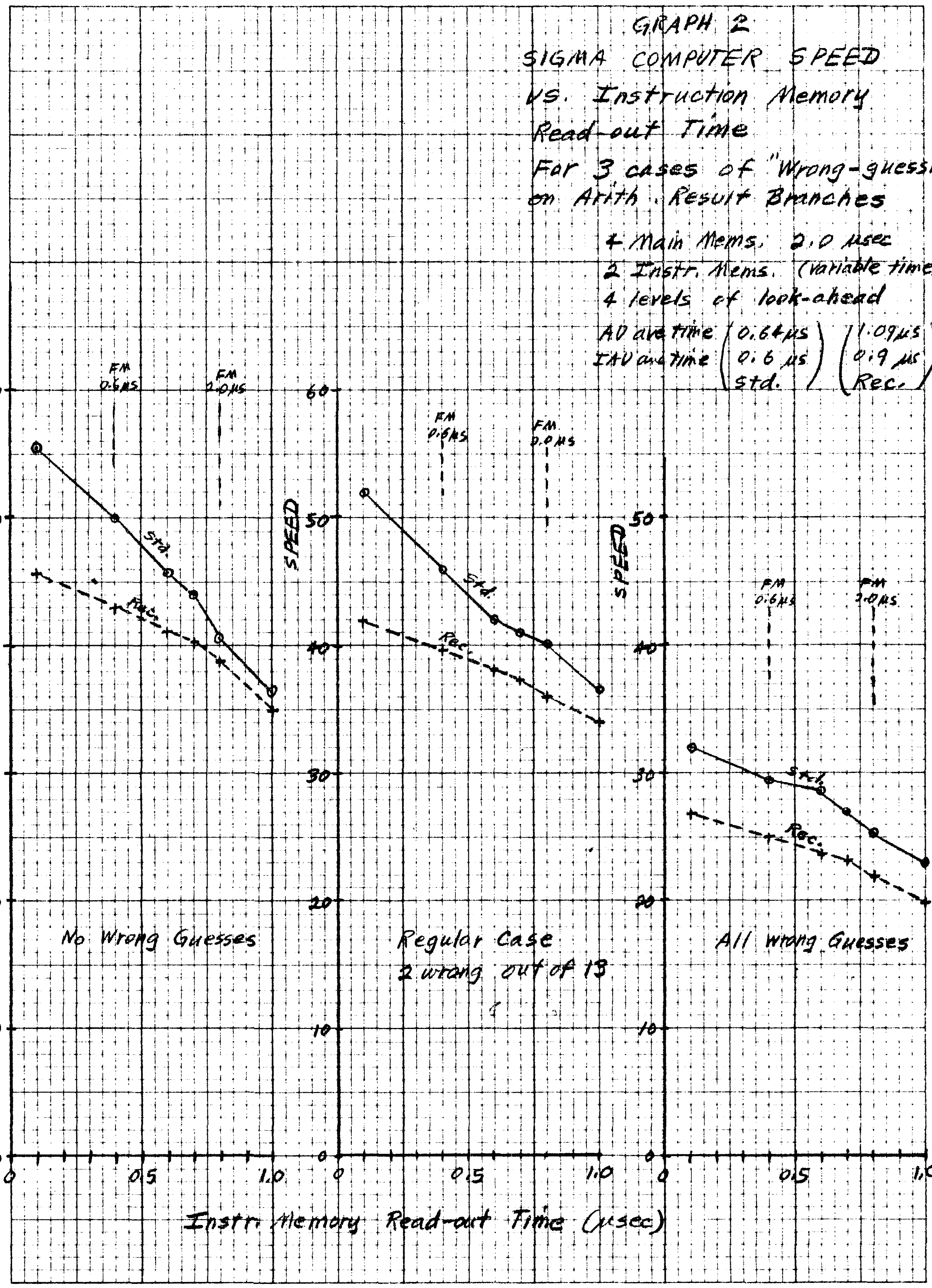
Harwood G. Kolsky

Harwood G. Kolsky
Product Planning Representative
Project 7000

HGK/jcv


Graph I
SIGMA COMPUTER SPEED
vs. Fraction of Arith. Result
Branches "guessed Wrong"
For The Monte Carlo Branching Code
4 Main Mems 2.0 µs
2 Instr Mems 0.6 µs or 2.0 µs
4 levels of look-ahead
AU ave. time (0.64 µs / 0.6 µs / Standard) (1.09 µs / 0.9 µs / Recommended)
IAU ave time
SPEED
0
10
20
30
40
50
60
70
0
0.2
0.4
0.6
0.8
1.0
all correct
2 wrong out of 13
all wrong
Std. 0.6 EM
Std. 2.0 EM
Rec. 0.6 EM
Rec. 2.0 EM
Fraction of Arith. Result Branches
which are guessed wrong.

HGK



GRAPH 2
SIGMA COMPUTER SPEED
VS. Instruction Memory
Read-out Time
For 3 cases of "Wrong-guessing"
on Arith. Result Branches

HGK

FILE MEMO Branching on Arithmetic Results May 19, 1958

DISTRIBUTION:

Mr. D. W. Pendery
Dr. J. Cocke
Dr. F. P. Brooks
Dr. G. A. Blaauw
Dr. W. Buchholz
Mr. E. Bloch
Mr. E. D. Foss
Mr. C. R. Holloran
Mr. S. W. Dunwell
Mr. R. T. Blosk
Mr. R. E. Merwin
Mr. J. E. Pomerene
7000 Product Planners