Rough Draft Company confidential

FILE MEMORANDUM

SUBJECT: Simplified Timing Rules for SIGMA

I. Introduction:

Quite often as part of the programming of a new problem for SIGMA it is desirable to make rough estimates of the running time of various sections of the code. Sometimes these estimates are for comparing the running time on STRETCH vs some other computer, but usually they are for estimating the amount of machine time needed or for comparing alternate ways of programming the same job to see which runs faster.

Because most applications do not require the precision of a detailed timing chart or Simulator run, I have listed two simplified methods of estimating approximate running times. The methods are in the order of increasing complexity. The user should pick the one which is commesurate with the accuracy of the estimate needed.

The times listed are averages obtained from Simulator runs assuming "normal" sequences of instructions and memory conflicts and have only indirect relationships to the hardware speeds.

II. Method One -- Simple Averaging

1. Form a total counting all the instructions executed in the program as one except the following.

   (a) Omit all simple index type instructions which are followed by Floating Point or VFL instructions.

   (b) Omit all branches ~~xxbxbx~~ which are not taken.

(c) Count TRANSMIT orders 2^n times, where n is the number of words transmitted.

(d) Count SWAP orders 3^n times.

2. Multiply the total thus obtained by 1.4 microseconds. This is the approximate total time for the program not counting start-up or run-down times. An additional 10 microseconds should be added for these if the program is not imbedded in another program.

III Method Two -- Averaging by Classes of Instructions

1. Count the number of instructions to be executed in the program in each of the following types:

(a) Floating Point

(b) VFL

(c) Indexing

(d) Branching

(e) Miscellaneous (TRANSMIT, etc)

2. For Floating Point, form a sum of the following:

(a) Count each add type instruction as 1.0 usec, Multiply instructions 2.0 us, Divide 7.0 usec, Square Root 20 usec, Loads and Stores 0.6 usec.

(b) Add 0.5 usec for each case of memory conflicts (data coming from the same box 2 instructions in a row.) If the exact number isn't known — assume one quarter of consecutive main memory references are conflicts.

3. For VFL Instructions, multiply the number of Add or Connect Type instructions by:

   (a) 1.0 usec for 1-8 bit field data

   (b) 0.5 usec for each additional byte in the data

   (c) Add 1.0 usec for each cross-word-boundary case. (as an estimate assume one tenth of references cross boundaries)

   (d) Add 1.0 usec for for each "To Memory" type.

   (e) Add 0.5 usec for each memory conflict as for floating point

   (f) For binary Multiply add 2.0 usec to the corresponding ADD case

   (g) For binary Divsion add 7.0 usec to the corresponding ADD case

   (h) For Decimal Multiply and Divide, add 3.0 usec to the corresponding binary case.

4. For Indexing Instructions (except branches)

   (a) Count as zero time if followed by a VFL or Floating Point Instruction.

   (b) Count as 0.8 usec if followed by another index operation or a branch order.

   (c) Count each step in LVS, LVE, and RN as 0.8 usec.

   (d) Add 0.5 usec for each memory conflict.

5. For Branching Instructions

   (a) For all branches not taken count as zero time if followed by a VFL or Floating Point Instruction

   (b) Count other branches not taken as 0.8 usec.

   (c) Count unconditioned branches and index branches which are taken as 2.0 usec.

(d) Count arithmetic result branches which are taken as 5.0 usec.

6. For Miscellaneous:

Each special instruction has its own time. Some of the more common cases are:

(a) TRANSMIT: Count 1.6 usec per word transmitted

(b) SWAP: Count 6.0 usec per pair of words swapped

(c) CONVERT: Count 15.0 usec.

Other repetitive type instructions such as EX, EXIC, MPYC should be broken into their component parts and these summed as above.

7. If the program is isolated (not imbeddded in another program) add 10 usec for initial start-up and final run-down times.

Method I - Example: Mesh Problem

No. of Ops	226
Covered	- 2
SWAP	+ 2
	(226) (1.4) = 3.16 us
	Start-up = 10
	326. us

Method II - Example: Mesh Problem

		Number	Time
(a)	Floating Point		
	L & S	112	67.2
	ADD	52	52.0
	Multiply	28	56.0
	Divide	8	56.0
	Memory Conflicts (est)	10	5.0
(b)	VFL	0	0
(c)	Indexing (Except Branching)		
	Followed by FP or VFL	1	0
	Not followed	11	8.8
	LVS	8 x 2	12.8
	Memory Conflicts	0	0
(d)	Branching:		
	Branches not taken	1	0
	Index Branches Taken	4	8.0
	Arithmetic Branches Taken	0	0
(e)	Miscellaneous		
	SWAP	1 x 1	6.0
(f)	Startup Time		10.0
			281.8 usec.
		Actual time from SIMULATOR =	286.2