The DINKIAC I—A pseudo-virtual-memoried mini—For stand-alone interactive use

by RICHARD W. CONN

University of California
Berkeley, California

INTRODUCTION

The past three years have witnessed the development and sale of a large and unanticipated number of small general purpose digital computers. These machines—the mini-computers—originally intended for real-time use in applications such as production control, now serve many diverse functions, ranging all the way from data buffers to the central processing units of small time-sharing systems. One trade journal even reports a sale to a home hobbyist claiming that initial costs are comparable, and upkeep less, than for other "recreational" equipment such as boats or sports-cars.

Several manufacturers have offered a basic machine with four thousand eight or twelve bit words, and with teletype I/O, for under ten thousand dollars.[1,2] Because of keen marketing competition and recent developments in integrated circuit technology these prices are continuously dropping. Memory costs, however, have not kept pace with the decreased logic costs brought about by the new IC's. Before truly spectacular price drops can be made the cost of memory must be reduced.

Memory in the above context evokes images of undelayed random addressability by word, or, more specifically, of magnetic cores. Yet if we consider computing systems generally, core memory represents but a small percentage of a typical installation's total storage. High core fabrication costs have led—in all but the tiniest systems—to the utilization of memory hierarchies. Devices most commonly comprising these hierarchies are, of course, the familiar magnetic cores, drums, disks, and tapes.

The questions to be examined in this study are: How cheaply can a machine adhering to storage hierarchy principles be built? What will it look like? and What good is it? To be in any position for viewing either of the others we must first address ourselves to the question, "What will it look like?" To do this the design of the Dinkiac, a machine meeting the implied constraints, will be summarily described. Explicitly stated these constraints include cheapness, component availability, and completeness in the sense that the user will not be required to purchase additional hardware. Once the Dinkiac design has been outlined, its usefulness can be assessed, its performance and architecture confirmed by simulation; construction details and alternate features may be presented, and its cost ascertained.

THE DINKIAC

Physically, the Dinkiac will appear as a typical keyboard—cathode-ray-tube display terminal. It will consist of a typewriter-like, 64 key, keyboard; a small CRT with a display capability of up to 84 characters presented in seven rows of twelve characters each; a row of lamps and switches; a single track low quality tape cassette recorder; four magneto strictive delay lines—all packaged together with the necessary register and logic components.

With its 16 bit word size the Dinkiac will appear to a machine language programmer as one of the larger minis. A word will represent data as either a single fixed point binary fraction in two's complement form, or as two eight bit character bytes, the last 6 bits of each conforming to USASCII standards.

Each instruction will comprise one full word in a fixed format with the first four bits (0-3) for the operation code; bit 4 a possible index register designator; bit 5, an indirect bit; bits 6 and 7, a page (delay line) address; and the last eight bits (8-15), the address within a page of one of 256 sixteen bit words.

Main memory will be made up of four magnetostrictive delay lines each storing 4096 bits. These lines will have a bit rate of two megahertz for a maximum access of a little over two milliseconds or an average access of approximately a millisecond. Each of these lines with a capacity of 256 words will be said to store a page of information. Processing may take place in any one of these lines concurrent with an exchange of information between secondary storage and some other line, not including the first, or page zero line. (Many readers will challenge the wisdom of choosing delay lines over shift registers. The latter has a speed advantage as well as the greater potential for cost reduction, matching decreases in the other IC's. There are, however, no large cheap shift registers currently available, and since it is our intention to show that a cheap instrument can be immediately constructed from off-the-shelf components, we are forced to choose the moderately priced and readily available delay line.[3])

The previously noted cassette recorder will provide secondary storage; a single tape retaining information in one of 128 blocks of 256 words each. Bit storage and retrieval rates will be around three kilohertz fixing page transfers at around one and a half seconds. The source and adequacy of these speeds will be discussed in the simulation section.

As originally conceived, the Dinkiac included hardware for automated page swapping, thus inspiring the notion—echoed by the paper's title—of a virtual memory machine; the virtual space being the size of the tape or more accurately the number of tape blocks times the number of words in a block, i.e., 32*K* Dinkiac words. Memory addressing was to have employed a page register-associative search scheme which operated in the following manner: Three (because page zero is not swappable) seven bit page address registers were loaded under program control. An instruction pointing to one of these registers (with the delay line address bits) referred to the tape block indicated by that register's contents. The instruction's address field indicated one of 256 words within the page. The requested page may or may not have been physically present in some delay line. Three seven bit registers were to compare their contents with that of the indicated page register and, if found, switch in the associated line. Because of the great disparity between word access and logic switching time the hardware for this associative search need not have been fast. If a specified page was not in any of the delay lines it was to have been retrieved from the cassette and stored in some line according to an algorithm which first checked sequential delay lines to find one in which the dirty bit had not been set. (The dirty bit was set—by the memory store signal—for any line which had been written into.) If all lines were dirty one line was selected and written out before the requested page was fetched. If program execution was delayed awaiting the fetched page, the program counter was stored and control transferred to a preset interrupt location. The described addressing scheme is shown in Figure 1.

Unfortunately this automation accounted for more than 20 percent of the total logic costs. In addition the primitive page swap algorithm may have proven unsatisfactory and required additional commands or even a complex sequence initiated from a read-only memory. In any event, the logic has been reduced to near minimum and any automated page swapping will now be under software control.

It is assumed that this operating system software will minimally include a keyboard input and display program as well as a cassette directory and search routine. Transfer instructions and busy flags will facilitate its operation and attempts to execute instructions from pages in the process of being swapped will still effect a transfer of control to the interrupt location. The inclusion of a fixed memory interrupt location is the primary reason for not swapping delay line zero.

While the cassette's primary function is to provide intermediate storage it also doubles as a cheap and convenient source of input/output. Initial input, however, is entered by way of the alphanumeric keyboard. Depressing a key will enter an encoded character into an eight bit keyboard buffer, turn off a console lamp, and set a one bit flag register. This flag may be interrogated by a running program and is reset—along with the lamp—by transferring the contents of the keyboard buffer to the accumulator. Striking a key will not enter a new character into the buffer while the flag is set.

Visual output is direct to a CRT from the first 42 word locations of the zero or non-transferable delay line. These words are gated sequentially in pairs (modulo 21) into a 32 bit output buffer on each cycle through memory. The low order six bits in each of the four bytes are, in turn, used as an input to a small read-only memory. This memory in conjunction with an appropriate counter and shift register provides serial output for modulating the CRT's "*Z*" or intensity input. These components together with a character, line, and row counter, and two deflection amplifiers and digital to analog converters, constitute the output device. It should be noted that the memory itself pro-

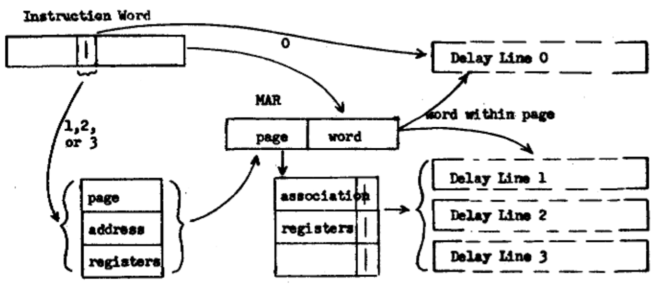

Figure 1

vides for display buffering and that all information is retained in character format. Scan conversion from the 32 bit buffer is performed as needed. Since it is possible to change characters in the output portion of memory before they have actually been displayed, it is anticipated that display programming will be handled as a function of the machine's interactive use. The most obvious example is provided by the displaying of a keyboard input message.

An operation panel located between the keyboard and display tube includes 'power on' and 'interrupt' toggles, 'start' and 'go' buttons, a five position rotary switch, and eighteen display lamps. The start button clears all registers except the program counter—into which the start address $(64)_{10}$ is forced—and loads tape block zero into delay line zero. Once block zero has been read the machine will begin instruction readout from the start location. Depending upon the state of the machine, depressing 'go' will either initiate a read of the next instruction—from the location currently specified by the program counter—or begin the instruction execution. The interrupt toggle will set or reset an interrupt step mode flip-flop. When set, this flag will force a machine halt after each instruction read and after each instruction execute. If 'go' is depressed while halted following a read, the machine will proceed to the execute. If it is depressed after the execute—and without a reset from the toggle—the program counter will be stored and the next instruction will be taken from the interrupt location.

The interrupt arrangement permits program stepping in one of two ways. For possible machine malfunction or difficult logical sequences the display lamps may be used in conjunction with the rotary switch to inspect the contents of the major processor registers. For more routine debugging, the user may choose to enter a subroutine which will convert and store relevant registers for subsequent display on the CRT. This mode will allow him to view, for example, the contents of the accumulator and the program counter—in any format he has chosen—at every other push of the 'go' button.

Given the above design it should be helpful to briefly consider a couple of the Dinkiac's unique operational and programming aspects. First and most obvious is the procedure imposed by keyboard limited input. Since all programs must be typed-in, it is probable that the typical user will be concerned only with conversational routines such as JOSS, FOCAL, or conversational BASIC. These processors should be structured in such a way that an anticipated routine will be scheduled into a delay line and ready for use. For example, an interactive algebraic processor could be segmented such that routines for matching, scheduling, and arithmetic operations are seldom or never swapped, while more complex numerical subroutines are arranged in a hierarchy of priorities with the most common (square root, sine, . . .) at the top and those seldom used (matrix operations, error exceptions and comments, . . .) at the bottom. While the software designer must try to segment these programs for the minimum swapping delay, it should be borne in mind that in conversational systems an occasional delay of several seconds is no cause for concern.[4] Balance between computation and user interaction is the significant factor.

It is hoped that by now the reader—having considered the design overview together with the cursory remarks relating the machine with certain time-sharing concepts—will have acquired sufficient intuition to answer, for himself, the third of our questions, "What good is it?" or more graciously put "What market does the Dinkiac serve?" For our part we will start with the statement that anyone now using a desk calculator can—for the same price and without sacrifice of calculator speeds or functions—enjoy the additional benefits of a completely general purpose digital computer. Additionally, the machine will provide a single user with a computing experience not unlike one he would receive at a time-sharing terminal. That is, for highly interactive work he can expect extremely fast replies with respect to his own response time. For compute bound requests, such as compilations or iterative numeric calculations, he should suffer no greater frustration than that engendered by a small well used time-sharing system. It is accurate to add that for the same jobs these periods of delay would compare favorably with a mini time-sharing system.

Because the tape cassette secondary storage will double as a fast I/O device a library of special purpose application cassettes can also be marketed. Examples are: BASIC for the schools; 'desk calculator' for small businesses; and, 'preparing your federal tax return' for the 'home hobbyist.'

SIMULATION

Our concern with a computer simulation is twofold, aiming first at determining the Dinkiac's gross architectural configuration, that is, the number and length of its delay lines, and second, at obtaining some sense of its overall performance. GPSS/360 (IBM's General Purpose System Simulator for the 360 series) was chosen for this task—both for its ease of use and its ready availability.[5]

For a simulation to serve its intended purpose the assumptions upon which it rests must be both valid and appropriate. The assumptions underlying this simulation are of two kinds, the first has to do with hardware component speeds and may be based on the price quotes of a number of manufacturers, the second requires a knowledge of program behavior and is far more tenuous. An early discussion of equipment characteristics will provide a foundation for the subsequent consideration of these less structured issues.

Magnetostrictive delay lines are offered in models with delays of up to 10 milliseconds at the maximum or 2MHz bit rate. Prices vary only slightly over the range with the longest lines (in quantity lots) costing less than 10 dollars more than the shortest. Since prices are typically constant up to a delay of around 2.5ms, a $4K$ bit line costs no more than one with half that capacity. Restricting the choice to sizes which facilitate binary addressing, these delays and bit rates imply that lines of up to $16K$ bits are feasible.

Because the Dinkiac is a single address machine all non-jump instructions must be taken sequentially, and if operands are positioned properly those with fewer than 128 memory fetches will be executed at delay line speed. (Switching time, even for slow transistor logic, can always be accomplished during the delay line to register transfers and may therefore be completely ignored.) A straight line program, then, will be executed at about the product of the line speed times the number of instructions. For the Dinkiac we have described—with its $4K$ bit line—this would amount to approximately 500 instructions/second while a $2K$ line would double the rate and one with $16K$ bits would cut it to a low of 125 instructions/second.

The tape cassette market is less stable than the market for delay lines and one may find prices ranging all the way from under thirty dollars to 100 times that price. The machines on the low end are intended for audio use while those at the other are designed for the reliable high-speed transfer of digital data. Advertised speeds for the expensive instruments give writing rates at under 10,000 bits/second with reading rates to 20,000. Experiments indicate that digital (square wave) recording on cheap audio equipment can be successful at speeds of two to two and one-half thousand bits per second. Specifications from a number of manufacturers marketing inexpensive recorders indicate that for under 100 dollars one can conservatively assume the following characteristics: (1) Read/write speed of 3.75 ips with a recording density of 800 bpi (bit serial recording) for a transfer rate of 3000 bps; (2) Search speed (fast forward and rewind) of 75 ips; (3) Start/stop time of 60 ms; and (4) Inter-record gap of ½ inch.

The properties given above will be used in the simulation, and to reinforce their conservative character, cassette page transfer times will always include time for the transfer of a full half inch inter-record gap as well as the times for both starting and stopping the tape. This caution also allows for any timing oversight arising from the recording technique, which we have assumed will follow teletype signal transmission methods, i.e., asynchronously, with a start pulse followed by data followed by completion pulses. To time a $16K$ block transfer, then, we will assume that 16,384 data bits plus a 400 bit equivalent inter-record gap are transferred at a rate of 3000 bps to which 120 ms, start and stop time, are added. That is, block transfer time $=(((\text{line size}+400)/3000)+.120)$ seconds.

Tape search time will be based upon a full tape capacity of half a million (2^{19}) information bits. (Later we will include some results gathered when providing for 256 blocks of the larger page sizes, i.e., for tapes of 2^{20} and 2^{21} bits.) Tape length, not including inter-record gaps, is approximately 655 inches—2^{19} bits at 800 bpi. Total search time will be determined by adding—to this length—a half inch for each record and dividing by the 75 inches/second rate, or, total search time $=((655+(\text{no. of blocks on tape}/2)/75)$ seconds.

We may now specifically formulate three questions we wish our simulation to answer: (1) What is the best page size? (2) How many lines are necessary for satisfactory performance? and (3) How will the Dinkiac compare with other machines? Given some assumption regarding the number of jumps expected during the execution of a program plus the anticipated distance of the jumps—i.e., what percentage of jumps will remain within 10 words of the current address, 20 words, etc.—it is possible to run simulations based upon the given transfer rates to obtain meaningful results for the first two of these questions. If, however, we wish to relate the Dinkiac's performance to that of other machines we will need some standard.

Fortunately, such a standard exists in terms of average instruction time. Given anticipated percentages for each instruction type and applying these percentages to the machine's actual instruction execution times, we can determine the time required for an 'average' instruction. Gibson has provided us with a set of such percentages by tracing 55 IBM—7090 programs involving 250 million instructions.[6] The traced programs were comprised of 30 FORTRAN source programs, 5 machine-language programs, 10 assemblies, and 10 compilations. Gibson's set of percentages, called the Gibson mix, has been used in many machine comparison studies. Because the Dinkiac has no floating point hardware, approximate averages for subroutine execution times will be given for the floating point instructions. The same will be done for multiplies and divides. The Gibson mix programs were scientific and give a conservative average with respect to a similar mix projected from the data processing field. Figure 2 is a table of Dinkiac instructions, 'worst' case times, and the loosely corresponding Gibson percentage. Execution times are given as delay-line revolutions.

Because subroutines are included, a single Gibson mix instruction must represent more than one of the Dinkiac's. Specifically, 87 percent are one to one, 7.7 percent are ten to one, and 5.3 percent are twenty to one. There are therefore 2.7 Dinkiac instructions to each of Gibson's and the average execution time for these 2.7 instructions is 8.6 revolutions. At 2.7 words for a Gibson instruction, each line of the 256 words/line machine we presented is capable of 'storing' 94.8 Gibson instructions. Similarly a 128 word line will contain 47.4 instructions, and so on. We have greatly simplified the remaining calculations by assuming a Gibson instruction size of 2.5 words and line lengths which are integral multiples of that number—forcing the use of 125 for the 128 word line, 250 for the 256 word line, etc.

Returning now to the still unspecified assumptions regarding program behavior, we find the question of jumps partially resolved by the Gibson mix. The mix assigns a 16.6 percent likelihood to the 'Test and Jump' instruction. We will assume that the jump is taken half this number, or 8.3 percent. To this we must assign some number of jumps to compensate for those subroutine loops incurred by our superimposition of the Gibson instructions over the Dinkiac's. Suppose 100 Gibson (270 Dinkiac) instructions are executed. Of the 270 Dinkiac instructions, 8 will be for multiply and divide, 69 for floating add and subtract, and 106 for floating multiply and divide. Assuming a five instruction loop for the first two instruction types and a ten word loop for the last, we will arrive at 26 jump instructions or slightly less than 10 percent of the instructions executed. We may further assume that these subroutines will be retained in the zero delay line and that return jumps will be back to the lines from which the subroutines are called. The model reflects this analysis.

The question of how far each jump goes with respect to the current program address counter is not easily answered and is closely allied to the question of how often must a new page be fetched. Until some study is made—similar to Gibson's but with just this aim—or, until studies of time-sharing systems provide further insight into page swapping behavior, no well-grounded assumption can be made. We will postulate that of the jumps taken—not including the 10 percent headed for delay line zero—50 percent will remain in the line they are at while the remaining half will go to the lines following with percentages of 50 percent, 37.5 percent, and 12.5 percent, respectively. Here the delay line sequencing is considered circular. This jump distance assumption is, of course, inconsistent with the varying line size and favors short lines. We will compensate for this advantage by making a near worst case assumption regarding page swapping, namely, that a new page be fetched once for every straight line pass through the memory.

We are now in a position to present details of the model. Each GPSS 'transaction' will represent either ten Gibson instructions or a signal to initiate the operation of some given line or tape. Each delay line consists of a holding 'queue' for the transactions, a memory 'facility' and a 'storage' capable of accommodating the appropriate number of instructions for a specified line size. To avoid simulating the simultaneous execution of instructions in more than one line, only sufficient transactions to queue up for a single line are generated at any one time. A transaction entering a facility (one of the delay lines) from a queue 'seizes' that facility precluding its use by any other transaction. An appropriate number (25 for 10 Gibson instructions) of instructions is 'entered' into the line storage and the total storage entries compared with the line capacity. If the storage is full, it is reset to zero; the facility is released; a transaction is removed from the queue; and new transactions are created for the next memory line. If the storage is not full, 18.3 percent of the transactions go to a jump instruction sequence where the clock is advanced 10 'jump' times and the transaction is entered into holding buffers according to the previously discussed jump distribution. In the 81.7 percent non-jump cases, the clock is advanced by the time required for ten line revolutions times a GPSS 'function' which randomly chooses (on the basis of a given bias—in this case the Gibson percentages) the number of revolutions. The facility is then released to allow for another entry from the queue; a transaction is removed from the queue; and ten transactions (instructions) are terminated.

Except in the case of the zero line, the completion of each line triggers a set of transactions for the next in a round-robin fashion with the last line triggering the first. Thirty percent of the completions from the zero line may additionally store a transaction in one of the holding buffers to simulate the subroutine return jumps. A counter at the end of the last line starts an end-of-job sequence which continues the program for only those lines which have items in their holding buffers. Completion of the last line also sends a transaction into the tape queue. Transactions in the tape queue seize a tape facility and then randomly 'pre-empt' one of the swappable delay lines. A pre-empted line is held until 'returned' and is precluded from seizure or use by any other transaction. The tape and pre-empted line times

INSTRUCTION	TIME IN REVOLUTIONS (Worst case for nonsubroutines)	GIBSON PER-CENTAGE
Load and Store Add and Subtract Logical	1.5	38.9
Multiply and Divide (10 word subroutine)	50.	.8
Floating Point Mult. and Div. (single precision) (20 word subroutine)	100.	5.3
Floating Add and Sub. (single precision) (10 word subroutine)	25.	6.9
Shifts and Register	1.	9.7
Test and Jump	.5	16.6
Index Search or Compare	2.	21.8

Figure 2

are advanced by one block transfer time and also, when appropriate, by tape search time. Simulations may be either "non-predictable"—in which case time to search half of the tape plus or minus any random interval up to that same amount is always applied—or, they may be "predictable." In the predictable or "75 percent predictable" runs it is assumed that the tape will have been correctly prepositioned in all but 25 percent of the transfers. At the completion of these tape advance times the pre-empted line is returned and the tape released. A general program flow is given in Figure 3.

Two results quickly emerged from the simulations, most apparent is the ruling out of either very short, or very long lines. The second, while less glaring, verifies the adequacy of a four line machine. It is tempting to continue the simulations with a greater number of storage lines—and when shift register prices fall this may prove feasible. Meanwhile, price considerations for this study dictate that the number be kept as small as possible. Upper and lower performance bounds were found by running the simulation with either no, or with complete, tape buffering.

The number of instructions executed during any one simulation varies slightly due to the randomness of the jumps. All runs, however, simulate the execution of close to 12,300 instructions. Execution time varies from a lower bound of two plus minutes (120,391 ms) to an upper bound of almost 12 minutes (707,961 ms). A table showing the total execution time in milliseconds for thirty-one simulations is given in Figure 4. Figures 5 through 8 are graphs of the four general cases: four lines both predictable and non-predictable and the same for three lines. The dotted lines in Figures 5 and 6 are the results of allowing the number of tape information bits to double once for the 256 word block and twice for the 512 word block. That is, to maintain the tape block count at 256. Each graph includes upper and lower bounds in addition to the simulation's finding for the particular case. The graphs argue convincingly for the 256 word page size, and yield insight into the nature of the balance between instruction execution and page transfer times.

DESIGN SPECIFICS AND OPTIONS

Sufficient detail to familiarize the reader with the Dinkiac's peculiarities was given in an earlier section. Here we will add a few design particulars, as an aid to cost estimation, and present some significant options.

The Dinkiac is designed around a five register bus in a manner typical of the minis. Signals from decoded instructions, together with outputs from a sequencer

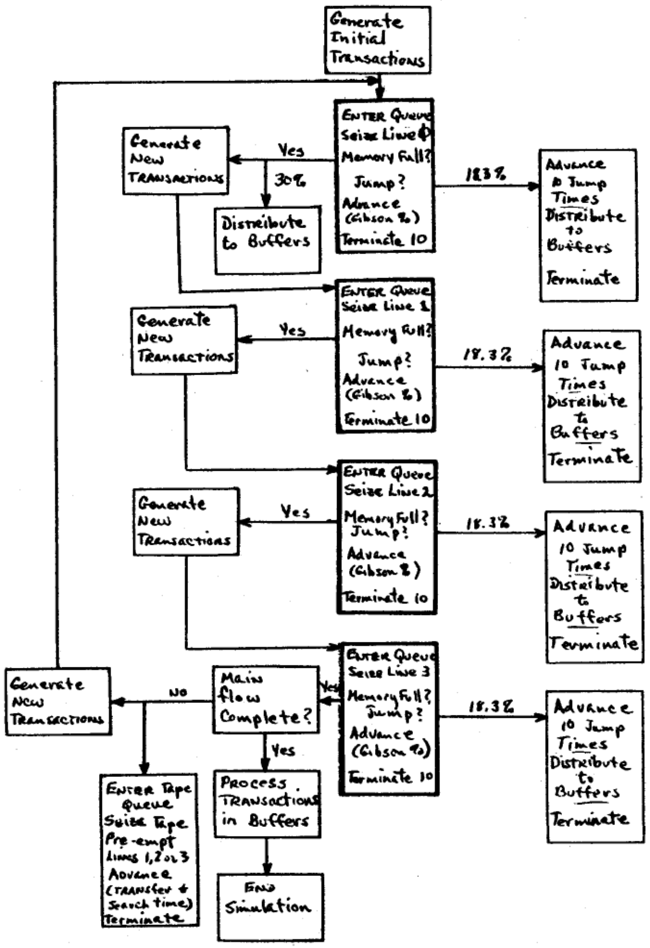
Generate Initial Transactions
ENTER Queue Seize Line 0 Memory Full? Jump? Advance (Gibson %) Terminate 10
Generate New TRANSACTIONS
Yes
30%
Distribute to Buffers
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
ENTER Queue Seize Line 1 Memory Full? Jump? Advance (Gibson %) Terminate 10
Generate New Transactions
Yes
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
ENTER Queue Seize Line 2 Memory Full? Jump? Advance (Gibson %) Terminate 10
Generate New Transactions
Yes
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
Enter Queue Seize Line 3 Memory Full? Jump? Advance (Gibson %) Terminate 10
Main flow Complete?
Yes
No
Generate New Transactions
18.3%
Advance 10 Jump Times Distribute to Buffers Terminate
Enter Tape Queue Seize Tape Pre-empt Lines 1,2 or 3 Advance (transfer + Search time) Terminate
Yes
Process Transactions in Buffers
End Simulation

Figure 3

CASE	128 Word Page	256 Word Page	512 Word Page
Lower Bound	120,391 ms	234,921 ms	479,801 ms
4 lines, 2^{19} bits 75 percent predictable	217,246	245,853	505,688
4 lines, 2^{19} bits non-predictable	440,482	324,034	506,057
4 lines, 256 pages 75 percent predictable	—	278,479	552,542
4 lines, 256 pages non-predictable	—	454,316	695,778
4 lines, Upper Bound 75 percent predictable	285,037	331,847	549,388
4 lines, Upper Bound non-predictable	602,615	472,056	586,818
3 lines, 2^{19} bits 75 percent predictable	289,290	296,161	498,685
3 lines, 2^{19} bits non-predictable	547,898	402,066	532,240
3 lines, Upper Bound 75 percent predictable	372,937	364,044	583,955
3 lines, Upper Bound non-predictable	707,961	559,366	681,738

Figure 4

and the storage completion lines, determine register gating and the consequent bus information. The machine's instruction set should prove helpful in conveying an intuitive notion of its logical complexity, and is given in Figure 9. Codes in that figure are in hexadecimal unless otherwise shown. '*A*' designates the accumulator; '*CB*' the carry bit; '*M*' the memory; '*MBR*' the memory buffer register; '*P*' the program counter;

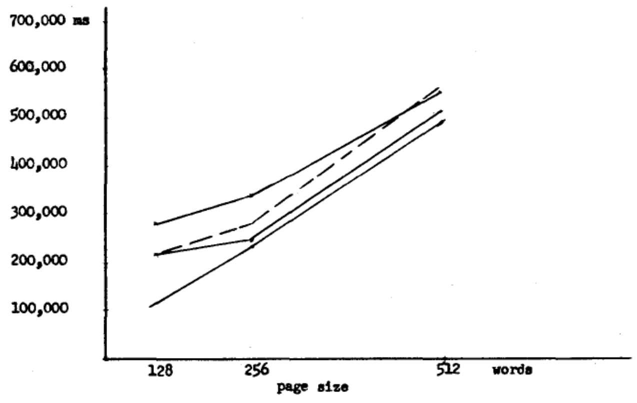

Figure 5—Four line—75 percent predictable

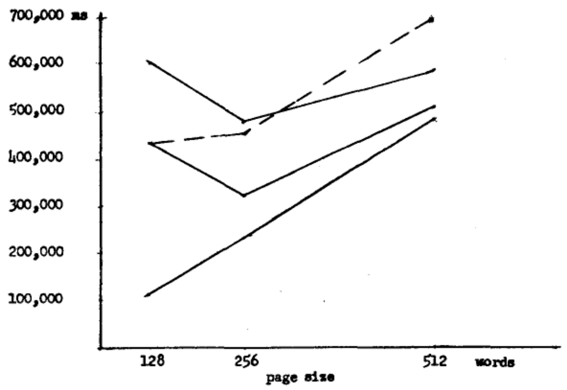

Figure 6—Four line—non-predictable

and, '*Y*' a memory address. Dinkiac word size and instruction format allow for the expansion and modification of this basic instruction set in many ways. For example, an index register may be added, or shifts modified to shift by some specified amount. Multiply and divide logic, too, could be included, and while these instructions might violate the spirit of the machine, they could easily be executed within a single memory cycle.

As implied in the simulation section, the number of delay lines can be increased with only a minor modification to the memory addressing scheme. In this case, the storage lines would continue—as they are now—to be synchronized with a single counter. Such a change could be expected to improve performance by increasing the data transfer-program execution overlap, but it would not alter the sequential instruction time which you may recall is roughly 500 instructions/second for the 256 word/line machine. Recall also that the two MHz bit rate allows for an information exchange between the memory buffer register and the chosen delay line in eight microseconds. This speed would allow the execution of non-memory referencing instructions from contiguous memory locations to proceed at the rate of 125,000 per second for a phenomenal increase of 250 times. A major factor contributing to the Dinkiac's easy circuit realization, however, lies in the difference between memory and switching speeds, and it is this great disparity that allows us to almost disregard the latter. If we wish the increased speed without altering this principle—which also enables us to purchase the cheapest logic components—we must provide both double memory buffer registers and the logic for their utilization. This type of speed-up must be carefully priced and reviewed in the light of the simulation results.

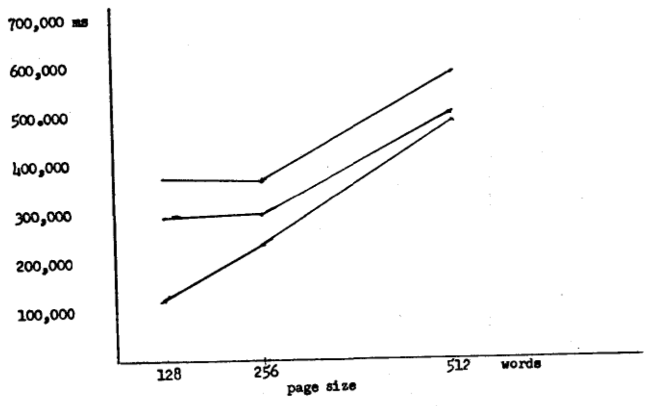

Figure 7—Three line—75 percent predictable

COSTS AND CONCLUSIONS

While the probability is high that any manufacturer seriously considering marketing such a device is already in either the small machine, display terminal or some related business—an instructive way to garner a sense of cost is to consider a prototype builder with no such association but who can avail himself of quantity prices for off-the-shelf items. If we assume a two to one gate to flip-flop ratio—not unrealistic for the proposed serial operation—meaningful logic costs can be ascertained by a simple count of single bit storage registers. Itemizing all registers—not integral with some other priced item (as, for example, the delay line input gates, . . .)—we arrive at a count of less than 200. This count is conservative, allowing bits for miscellaneous control and making no attempt to share or minimize the number or size of the registers.

A notion of dollar value can be ascribed to the count by using quantity prices for standard off-the-shelf TTL gates from leading suppliers. Such an assignment comes to $2.20 per bit where the flip-flop price is $1.60 and the gate cost $.30. This approach is again conservative taking into account no non-standard gates and using no very slow, but adequate, logic. Computed in this way the logic price to a backyard builder with connections is $440.

Similarly pricing the other components puts the delay line memory (four, 4096 bit lines) at $400 (in large quantity); the cassette recorder at $100; the keyboard at $75; the CRT and related components at $175(the display generator including read-only memory is available in lots of over 25 for less than $100); a power supply at $120; and a crystal clock at $100. The total, then, including logic is $1410. It is reasonable to expect that quantity costs to a manufacturer—including labor—would be a good deal less than this amount. We may note in this respect that currently

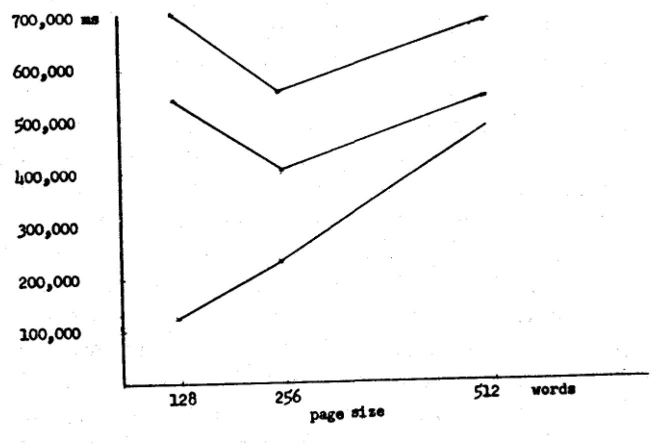

Figure 8—Three line—non-predictable

DINKIAC INSTRUCTIONS

Memory Reference

STO	Y	1xxx	$A \rightarrow M_Y$
ADD	Y	2xxx	$A + M_Y \rightarrow A$
SUB	Y	3xxx	$A - M_Y \rightarrow A$
JMP	Y	4xxx	$Y \rightarrow P$
JAM	Y	5xxx	$Y \rightarrow P$, if $A < 0$
JAZ	Y	6xxx	$Y \rightarrow P$, if $A = 0$
JSP	Y	7xxx	$P+1 \rightarrow M_Y$, $Y+1 \rightarrow P$
LDA	Y	8xxx	$M_Y \rightarrow A$
AND	Y	9xxx	$A \wedge M_Y \rightarrow A$
ISP	Y	Axxx	$M_Y+1 \rightarrow M_Y$, if $M_Y = 0$ then $P+1 \rightarrow P$
JCB	Y	Cxxx	$Y \rightarrow P$, if $CB = 1$

Non-Memory Reference

NOP		0000	No Operation
HLT		0001	Halt
SNI		0002	$P+1 \rightarrow P$, if Interrupt Flag $\neq 1$
SNK		0003	$P+1 \rightarrow P$, if Keyboard Flag $\neq 1$
CLA		002-	$0 \rightarrow A$
CMA		003-	$2^C(A) \rightarrow A$
CLC		004-	$0 \rightarrow CB$
CMC		005-	$2^C(CB) \rightarrow CB$
LAK		006-	Keyboard Buffer $\rightarrow A_{8-15}$
LAB		007-	$MBR \rightarrow A$
SHR		008-	Shift CB and A right 1
SHL		009-	Shift CB and A left 1
RTR		00A-	Rotate CB and A right 1
RTL		00C-	Rotate CB and A left 1
RLR		001-	Rotate A, 8
SCO	Y	04xx	If Cassette not Busy, $P+1 \rightarrow P$ and Search Cassette 0 for tape page xx.
RCO		$0(10xx)_2$--	If Cassette not Busy, $P+1 \rightarrow P$ and Read tape 0 into memory page xx (where xx = 1, 2, or 3).
WCO		$0(11xx)_2$--	If Cassette not Busy, $P+1 \rightarrow P$ and Write tape 0 from memory page xx (where xx = 1, 2, or 3).

Figure 9

advertised prices for display terminals are as low as $1500, and include all Dinkiac components excepting three delay lines (the displays have one), a cassette recorder, and computer logic. This price, incidentally, includes beautiful packaging. Suppose we add to the $1500 the excluded items, priced as above, for a grand total of $2340. There is nothing to indicate that a Dinkiac cannot be profitably marketed for under $3000.

This report has attempted to show that a general purpose digital computer—suitable for a large class of users, including those in small businesses and engineering firms, schools, and even private homes—can be built to market for a price near the low end of the desk calculator range. A GPSS simulation has shown the optimum memory length to be the one in which time for the execution of a page of instructions is closely matched with tape block transfer time, and has confirmed the adequacy of four lines, even while assuming highly unfavorable operating parameters. Additionally, by modeling with "Gibson instructions," we were able to acknowledge that the Dinkiac—while short on "bandwidth" in comparison with large machines—is certainly adequate for its intended purpose.

REFERENCES

1 D J THEIS L C HOBBS
Mini-computers for real time applications
DATAMATION No 39 March 1969

2 J W COHEN
Mini-computers
MODERN DATA No 55 August 1969

3 J H EVELETH
A survey of ultrasonic delay lines operating below 100 Mc/s
IEEE Proc Vol 53 No 10 October 1965

4 R B MILLER
Response time in man-computer conversational transactions
AFIPS Conf Proc Vol 33 p 267 1968

5 G GORDON
A general purpose systems simulation program
EJCC Proc p 87 1961

6 J J CLANCY
Notes on the 'bandwidth' of digital simulation
SIMULATION Vol 8 No 1 January 1967