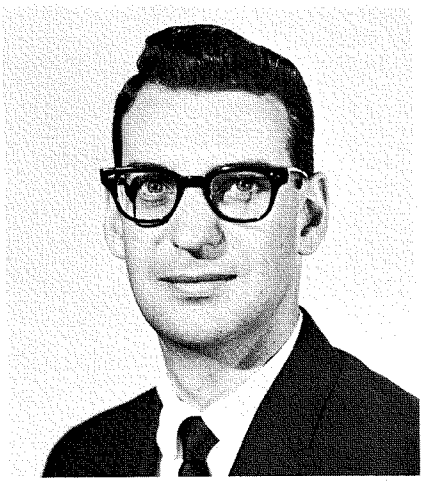
EMULATION TECHNIQUES ON THE RCA SPECTRA SYSTEMS

The very rapid advances in computer technology have made it desirable for users to update their existing data processing equipment with third-generation hardware. To accomplish this transition, a user must consider the cost of converting his programs—a major project for which the typical user seldom has a sufficient programming staff. In the past, simulation software has been supplied by the manufacturers to facilitate the program conversion effort; however, this technique is very inefficient. Simulation in general is economically practical only for infrequently run programs, and the more frequently run programs must immediately be recoded before the old system can be replaced. Emulation is a technique involving both hardware and software which permits existing programs written for the old system to be efficiently run directly on the new system. The hardware cost is small and the performance is at least an order of magnitude better than simulation.

T. A. FRANKS, MGR. *and* **C. S. WARREN, LDR.**
Computer Projects *Advanced Systems*
EDP Engineering, Camden, New Jersey



T. A. FRANKS received the BEE from Rensselaer Polytechnic Institute in 1956 and the MS from Stanford University on a fellowship program in 1957. Since joining RCA in 1956, he has taken part in the development of solid-state memory circuits; worked on various drum buffer memories; participated in the systems architecture, logic design, and application of several advanced real-time processing system; and was a project engineer on a BMEWS computer installation. Upon joining EDP in 1962, he was the engineering project leader of the RCA 3301 and Spectra 70/45-55 systems; then was manager of emulator microprogramming activities; and is currently manager, computer projects. Mr. Franks is a licensed professional in the state of New Jersey and a member of the IEEE, Tau Beta Pi, Sigma Xi and Eta Kappa Nu.

C. S. WARREN received the BSEE in 1952 from Virginia Polytechnic Institute. He joined RCA in 1952 as an engineer on the specialized training program, following which he spent seven years in advanced development working on magnetic memory devices and transistor circuit design and development. Also during this period, he was responsible for the design of several memory systems used in BMEWS. In 1959, Mr. Warren was transferred to the Van Nuys Division of RCA where he was made leader of a logic design group and was responsible for the design of a small real-time computer, initially used in instrumentation radars. In 1963, Mr. Warren transferred to his current position as Leader, Advanced Systems, in EDP Systems Engineering. In his current capacity, he has played a major role in the architecture and design of the Spectra 70 Emulators. Mr. Warren is a member of the IEEE, Eta Kappa Nu, and author of several papers and patents.

COMPUTER TECHNOLOGY has continued to experience rapid advances in machine performance, range of application, and varieties of peripheral devices. Accordingly, data processing users have, at various times, had to consider the economics of trading in outmoded equipment and replacing with the latest units. As users have become more sophisticated in the application of computers, a substantial investment exists in their library of programs and data files. The considerations of how to make a smooth transition from one system to another with minimum loss of investment has typically been traumatic.

There are currently several techniques that aid in this conversion, but none is a panacea:

Higher Level Languages: Any portion of the work that has been programmed in a machine-independent language can be run on a new computer, if a compiler is provided for that language. However, compilers have not been truly machine independent and often the language compilers are not available with delivery of the equipment. Further difficulties arise when machine language coded subroutines are mixed with the higher level language statements to patch in a special feature or fix a bug.

Translation: Each instruction or group of instructions can be automatically changed from the machine language of the old computer to that of the new. This translation is performed from either source

Final manuscript received May 2, 1967.

language or object program decks. Translators process the instructions before execution time; hence, it is difficult to translate an instruction that will be modified or replaced by the program itself before its actual execution. Translation is generally less successful between two computers that differ widely in architecture.

Simulation: Simulation differs from translation in that each instruction is interpreted in sequence at execution time. An image of the entire state of the computer being simulated (memory contents, registers, indicators, etc.) is maintained on a bit-for-bit or character-for-character basis in the simulating computer. Strict adherence to this rule of interpretation at execution time removes the conceptual problems present with translation. Because of their simplicity, interpretive routine simulators have been quite successful. Unlike a translator, a simulator must interpret an instruction each time it is executed rather than just once. Thus, there is a large performance degradation due to the interpretive overhead. Further limitations are often required on the fidelity of the interpretive routines to keep the simulation program size within reason. Most simulator programs have not been fast enough to be of significant value in the conversion efforts.

Reprogram: As a last resort, a problem may be reprogrammed in the new computer language. This reprogramming is costly, time consuming, and is just a brute force solution to the conversion problem. Eventually, all programs of value will be reprogrammed. However, reprogramming and debugging to this extent during a tolerably short transition interval is usually not feasible.

Emulation: An emulator is a package supplied by the computer manufacturer that includes special hardware and a complementary set of software. The package runs in the manner of an interpretive routine simulator program but is approximately an order of magnitude faster. The combination of emulator and the new computer in effect create an extended computer which, to the user, is the computer for which the original object code was written. The advantages of emulation are simplicity of operation and system thruput performance. Thruput can typically range from equal to four times better than the original system depending on the application.

EMULATION

In the Spectra computer series, emulation is used to perform the interpretive processing on all non input/output instructions by hardware microprogram. Interpretive processing of input/output instructions is handled by the complementary set of software. Microprogram computers generally have two sequence counters, one for the macro-instruction sequence, with which the machine language programmer is familiar, and one for a micro-instruction sequence, which interprets and executes the macro-instruction. These micro-instruction sequence steps are called elementary operations (EO's) and are pre-wired in a read-only memory.

The economic feasibility of read-only memories has allowed the Spectra 70/35 and 70/45 computers to employ these techniques for system control. Additional read-only memories may be optional add-ons to the system; each one containing the micro-instruction steps corresponding to a particular order code. Several earlier machines of the microprogram type have been described in the literature and include the TRW-133[1], Packard Bell 440[2] and Collins 8401[3]. The Spectra 70/45 and 70/35 have also been described.[4,5]

A microprogram simulation is invariably more efficient than a macroprogram simulation, since the interpretation is being executed at a point of least difference between the computers. On a macroprogram level, computers have widely different structures: instruction formats, fixed or variable data fields, binary or decimal character codes, etc. On a microprogram level, the ability to process bit-by-bit or character-for-character allows the more complex macro-level operations to be implemented by relatively straightforward sequences of elementary-bit or character manipulations.

To take a different view of the efficiency of micro versus macroprogramming, a given data processing problem coded in machine language for several computers causes a wide variation in the total number of memory cycles. The most efficient routine is a result of the computer with the best instruction repertoire for the original problem. This has been evident in the performance of software simulators. Emulators or microprogrammed simulators allow routines which, in terms of memory cycles, show little difference from the original computer's operation. Interpretive routine simulation by hardware microprogram allows the process to be done at the point of minimum difference.

The complementary set of software in emulation has a large part to play. In several cases, it becomes uneconomical to do a function by microprogram. Software control of input/output sequences allows a flexible approach to performance enhancements such as buffering, look-ahead, multiprogramming, and additions of facilities for handling peripheral devices not included in the initial microprogram release. Execution of appropriate error-recovery procedures is a problem best handled by software. Also, to the operator, the systems are totally dissimilar; the software provides an interpretive console to allow execution of operating procedures in the most convenient manner.

IMPLEMENTATION

Emulation is provided on the 70/35 and 70/45 by the inclusion of additional read-only memory banks, special software package, and some minor hardware modifications to the respective processors. The added read-only memory is used in two ways: interpret the instruction repertoire of the machine being emulated and to provide some special Spectra 70 instructions to assist the software portion of the emulator. The wired program within the read-only memory is referred to as the Emulator Microprogram (EMP).

The software portion of the emulator is called the emulator control program (ECP) and consists of two components: the emulator monitor system (EMS) and the emulator interface program (EIP). The EMS is a special operating system designed to run any of the Spectra 70 emulator features. The EIP is a special software package designed for each emulator to interface the EMP with the EMS. Both the EIP and EMS are programs written in Spectra 70 code. The primary responsibilities of the EIP are:

1) Translate the I/O instruction of the machine being emulated to equivalent Spectra 70 operations;
2) Perform any peculiar code translations;
3) Interpret I/O termination conditions and set any necessary emulated indicators; and
4) Interpret and execute all emulated console functions relating to errors, normal operation intervention, initialization, etc.

The hardware modifications consist of the control logic necessary for selecting and addressing the added read-only memory banks, and some added functions, to the microprogrammed structure of the machine to facilitate or enhance the emulation capability. In the implementation discussion which follows, it is assumed that the reader is generally familiar with normal Spectra operations and technology.[5,6]

The operation of the emulator is controlled by two control bits in the Interrupt-Status Register. The first bit is the *Emulator ON (EON)* and the second bit is the *Bank 3 (BK3)* control; the *BK3* control selects one of two possible emulators. The 70/45 allows up to two different emulators to be added to the system; the 70/35 allows only one. With the *Emulator ON (EON-1)* the four normal program states of the Spectra 70 processor take on the following functions:

Program State 1 fetches and executes all instructions out of the added read-only memory (bank 2 or 3 depending on the state of the *BK3* emulator control).
Program States 2, 3 and 4 fetch and execute all Spectra instructions out of the original Spectra read-only memory bank and allow a special set of EIP instructions unique to the emulator.

By designing the program states to operate in this manner, the emulator can

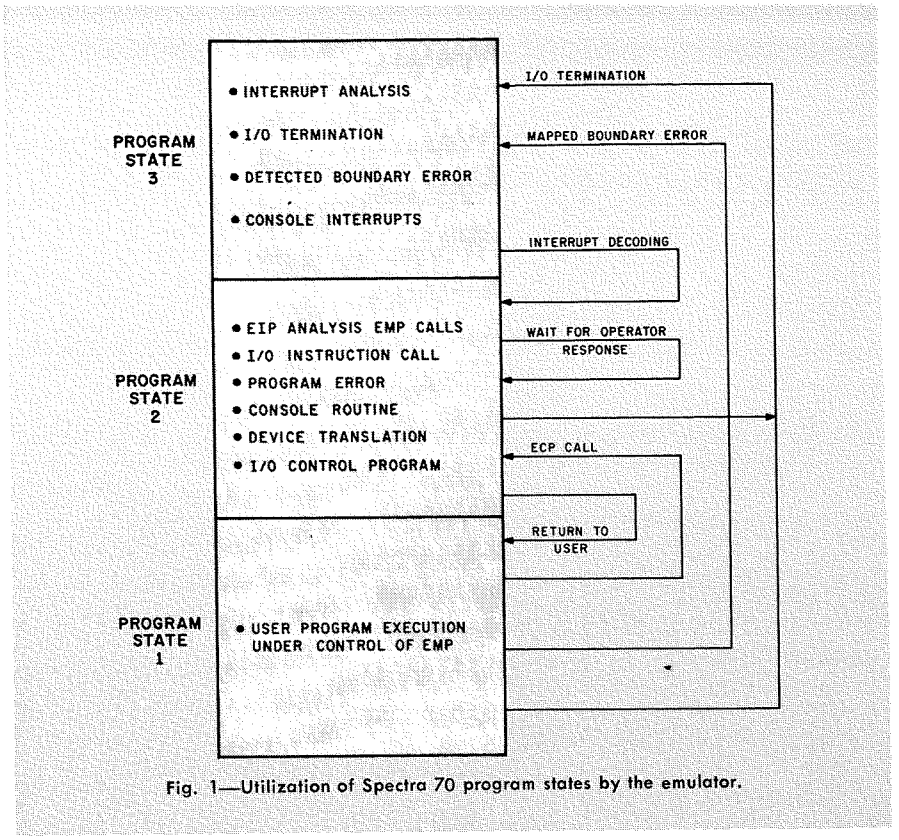


Fig. 1—Utilization of Spectra 70 program states by the emulator.

incorporate normal Spectra 70 coding for the ECP which must be executed in program states 2 and 3 and execution of the user's program being emulated in program state 1.

Fig. 1 illustrates how the program states of the Spectra 70 are used by the emulator. Program state 1 is the state in which the user program is executed under control of the emulator microprogram (EMP). State 3 is used by the EMS to analyze all program interrupts. All interrupts, once analyzed, are turned over to either the emulator monitor system (EMS) or emulator interface program (EIP) in program state 2 for proper handling of the interrupt. In program state 2, the bulk of the EIP and EMS are executed. An EIP routine analyzes all I/O instructions encountered by the EMP and prepares them for execution. Program errors encountered in the user's program are handled in program state 2; the console routine for operator communications is contained in program state 2 and the I/O control program which is responsible for issuing all I/O commands are also contained in program state 2.

During execution of a user program, the EMP will turn control over to the ECP for any of the following reasons:

1) An I/O instruction is staticized and decoded;
2) A condition is detected that would result in a machine halt or malfunction in the emulated machine; or
3) The instruction fetch or data fetch/store was outside permitted memory bounds.

For the first two conditions, the EMP will turn control over to the EIP in program state 2 directly. Communication with the EIP portion of the emulator is through a communication constant which is used by the software to determine the reason for the EIP call. In the case of an I/O instruction, the EIP will return control back to the EMP in program state 1 as soon as the I/O instruction has been executed and/or the device started. In the case of error conditions, the ECP will generally be required to give a printout on the console and wait for a response by the operator. Under any condition, the ECP will return control to the EMP in program state 1 only when the emulated program is able to continue its operation. The third condition is an automatic interrupt to program state 3 where it is handled like all other interrupts by the ECP. Special hardware has been added to the processor to facilitate the detection of the memory boundary used by the emulated program. This is a special addressing error condition detected only when the emulator operation is in effect.

MAPPING

The operation of the emulator depends on an instruction-for-instruction interpretation of the program being emulated. To do this, the characteristics of the native machine must be mapped one-for-one into the Spectra processor. To date, all of the machines being emulated on the Spectra equipment have been character oriented and, with the exception of the RCA 501, are decimally addressed machines. Spectra is a combination word and character machine with binary addressing and 8-bit characters. These characters of the Spectra and the machines to be emulated permitted a rather straightforward mapping of memory on a character-for-character basis. A contiguous portion of the Spectra main memory represents the main memory of the system being emulated. This portion of the Spectra memory is bounded on both sides by a special character which is used by the EMP to detect an operation which crosses the boundary. The ECP or software portion of the emulator resides outside these boundaries. To permit multiprogramming and facilitate two emulators in the system, the mapped memory is floated.

In addition to main memory, all registers and indicators used or referenced by the instruction repertoire of the native machine must be mapped or allocated equivalent storage in the Spectra 70 system. In all the emulators designed to date, the Spectra floating-point registers were assigned the function of representing the native machine registers and indicators. These were chosen for two primary reasons:

1) The floating point registers were not needed by the software portion of the emulator, and therefore, would not require storing and restoration every time the program state is changed;
2) The software system operating in program states 2 and 3 can access the floating-point registers as readily as the EMP. This later was a key consideration since both the software and, particularly, microprograms require frequent access to the mapped registers.

The ready availability of these registers is a major factor in the performance of the emulator.

In addition to the mapping of registers and memory, the emulator requires storage for conversion and translation tables. Address conversion and code translation tables are required to enhance the performance of the emulator. In all the Spectra emulators to date, the general registers for program state 1 were allocated to address conversion tables, since these registers are most readily accessible by the EMP, and address constants are the most frequently accessed arguments. Other tables needed by the microprogram utilize a special portion of main memory which is not accessible to the programmer. Normally, this portion is allocated to the multiplexer subchannel register. This restricted the number of subchannel registers that the emulator system could use to 64. However, this has not affected any system configurations to date since 64 still provides adequate room for expansion. This approach has the advantage that more of the main memory space is conserved for the programmer.

EMULATOR MICROPROGRAM

The emulator microprogram (EMP) consists of the following parts:

1) Instruction fetch and decode;

2) Individual execution algorithms for emulated instructions; and
3) Decode and execution of all special Spectra instructions.

The instruction fetch and decode portion of the emulator microprogram is entered automatically whenever an instruction in program state 1 is completed or whenever control is returned to program state 1. This portion of the EMP is common to the interpretation or execution of all emulated instructions. It fetches from mapped memory the next instruction, performs any necessary address conversions, updates all simulated machine registers, decodes the operations, and enters the execute portion of the instruction.

The instructions consist of individual microprogram routines. In the case of non-I/O instructions, these routines perform the operation and leave all simulated registers, indicators, and memory locations exactly as in the native machine. In the case of an I/O instruction, the EMP decodes the particular operation into an ECP communication constant which is utilized to select an appropriate EIP subroutine. Control is then switched by the EMP to program state 2 so that this selected EIP subroutine may proceed with the translation into an equivalent Spectra operation. The EIP will return control to the EMP only after the I/O operation is completed or the device is initiated and I/O simultaneity is specified and permitted.

The third portion of the EMP consists of the microprogramming routines for special operations to assist the EIP. For example the 301 emulator uses a move and translate instruction to move data of specified length from one memory location to another, translating the codes from one representation to another via a specified conversion table. In the case of this special operation, the EIP can translate I/O data from 301 codes to Spectra codes, thus permitting standard Spectra devices to be substituted for 301 devices. The special operations are in the format of Spectra 70 instructions and can only be executed when in States 2, 3 or 4 and the *EON* control is set to one. If the emulator is not *ON* or the emulator bank specified is not installed, these special operation codes (op-codes) will cause a normal illegal op-code error to occur. The special op-codes are fetched and staticized as a normal Spectra 70 instruction using the Spectra read-only memory bank. In the Spectra 70 staticizing microgram, a group of unused op-codes are assigned to these special operations and are designed to automatically branch to a fixed read-only memory location in the emulator bank. If the emulator is not installed or the emulator control is not *ON*, this branch will result in an op-code error. At this fixed read-only memory location, the EMP must decode all the special operations for this particular emulator. Once decoded, the EMP completes the execution of the special operation. This technique of expanding the Spectra repertoire permits each emulator to have its unique set of special instructions.

EMULATOR SYSTEM PERFORMANCE

In an emulator system design, performance is predominantly determined by 1) similarity of the processor being emulated and the emulating processor and 2) degree of fidelity desired in the processing functions. These factors can be separated into two categories: those related to the internal instructions and those related to the input-output functions. A partial list of items found to be of principal significance is tabulated below:

Internal Instructions

1) Conversion of decimal memory addresses to a binary equivalent;
2) Differences in character codes between the system being emulated and Spectra representations.

Input-Output Functions

3) Per-character processes required on data exchanges with peripheral devices;
4) Emulator software overhead required for interpretative translation of input/output requirements; and
5) Peripheral equipment speed differences.

Memory Address Conversion

All Spectra data processors employ straight binary-coded address references. The 70/45, for example, allows binary addresses ranging to 18 bits. However, the 1410 system employs a five-digit decimal field. The 1401 and 301 systems employ six-bit characters with a combination of decimal digits and zone bit combinations within each character of the address field to designate a memory reference. A conversion must thus be performed for each operand address fetched from the native address format to an equivalent Spectra binary reference. Since the base of mapped memory or equivalent location of the emulated system's zero address is relocated to some other non-zero value, this offset must be incorporated in the address conversion algorithm. These conversions have generally been implemented through series of decimal-to-binary table lookups so as to reduce the execution time. To further reduce the extra addition cycle required for the mapped memory base offset, one of the decimal-to-binary tables is offset an equivalent amount by the emulation software at initial load time. Thus, the decimal-to-binary weight and base offset are available in one table lookup operation.

Character codes

Character-code representations in the system to be emulated and Spectra equivalents are generally not the same. Operations such as comparisons and addition/subtractions therefore follow different rules. For example, the six-bit binary-coded decimal (BCD) representations used in the 1410 and 1401 systems differ from Spectra formats in two ways:

1) The collating sequence used in compare operations does not correspond to the actual binary weights; and
2) The binary representation of the decimal digit zero is 1010, not 0000 as in Spectra.

To eliminate special translations for every compare or arithmetic operation, the 1400 BCD characters are represented internally in extended BCD interchange code (EBCDIC) whose binary codes correct the above peculiarities. Since EBCDIC is a standard for Spectra equipment, compatible translation facilities are included in all input/output devices to convert card, tape, etc. codes into EBCDIC as the data is entering or leaving the processor. Thus, the codes are automatically translated as data is read or written and internal emulator processing of the characters is compatible with the Spectra hardware arithmetic unit.

Some internal operations still require conversion of the EBCDIC codes into BCD, and vice versa. Examples are the move-digit/zone or branch-on-bit-equal instructions. However, the additional penalty to these instructions is small compared to the gains in other algorithms.

Similar problems, although not as severe, occur with 301 and 501 codes. Here the difficulty is concatenation of a six-bit character into an eight-bit byte. Special provisions must be made to insure proper propagation of carries in arithmetic operations.

Per-Character I/O Processes

Input/output operations in the Spectra system are count controlled. For instance, a tape write command employs a starting memory address and a specific number of bytes to be written out. Input/output instructions available in the 1410, 1401, and 501 systems, however, are symbol controlled. That is, the length of a field to be written out (or read in) is determined by the location of a special terminating symbol in memory. Thus, before an emulated I/O instruction can be converted into an equivalent Spectra command, the field must be scanned to determine a byte count. This scanning operation adds to the time required to execute an I/O instruction and constitutes additional overhead. The time involved is determined on a per-character basis and if performed with conventional

instructions, would range from approximately 10 to 25 μs, depending on application. However, by utilizing special microprogrammed routines designed for these processes, the per-character times range from 1.2 to 5.5 μs. Through microprogramming techniques, almost an order of magnitude improvement was possible. When compared with the data rate from a 60KB tape station, this processing typically represents an increase of only 15% in data transfer time, which is exclusive of start/stop delays. Thus, the apparent increase in total type time is not excessive.

Emulation Software Overhead

The complimentary software package employed in an emulator (called ECP) provides the translation linkage to execute I/O instructions. To perform this function, a variety of actions are required 1) to initiate the equivalent Spectra device operation and 2) to translate termination conditions into corresponding mapped emulation indicators. Where appropriate, routines have been included to enhance system performance. These include such facilities as:

1) Advanced card reading into a buffer area;
2) Advanced print release by moving data to be printed into a buffer area;
3) Error recovery through retries; and
4) Provision for multiprogramming two emulation operations.

Typically, execution of the ECP routines adds 0.5 to 1.5 ms to the execution time of an I/O operation (exclusive of any per-character processing). This overhead can be minimized by overlapping the processing with input/output operations and using the above enhancements.

Peripheral Speed Differences

Device speeds in I/O-bound programs have as direct a bearing on overall thruput as execution time of internal instructions in compute-bound programs. By selecting peripheral devices which are faster than the corresponding units in the system being emulated, substantial improvements in thruput can be achieved. The start/stop characteristics of I/O devices can be as significant in system performance as the data rates; particularly where short records are involved. It is difficult to predict I/O performance or system thruput due to the complex inter-relationship between the device time, ECP overhead, and internal instruction mix being executed through microprograms.

Performance Measurements

The level of emulator performance is usually measured by total program thruput. This performance is determined by inter-relationships between the following:

1) Execution time of the microprogrammed internal (non-I/O) instructions;
2) I/O device character rates and start/stop times;
3) ECP overhead time required to translate I/O requirements into Spectra equivalents; and
4) The timing distribution of the above three items in actual operations.

The "average" performance ratios of native instruction times to emulated instruction times are listed below:

Parameter	*1410*	*1401*	*301*	*501*
Memory Speed Native System (μs)	4.5	11.5	7.0	15.0
Performance Ratios:				
Native time of 70/45	2.3	4.1	2.6	2.4
Native time of 70/35	*	3.8	1.6	*

* 1410 and 501 emulators are not available for the 70/35 system.

These figures were derived from a weighted mix of typical internal instructions, and hence are considered "average." The ratios vary from 1.6 to 4.1 and in no case is emulation slower than the original system; these figures illustrate the efficiency of emulation at the microprogrammed level versus simulation at the instruction level. Typical performance of simulation programs would be approximately an order of magnitude slower.

If programs only consisted of internal instructions and essentially no I/O activity, then the above ratios would properly represent thruput. However, most commercial data processing activities are quite dependent on extensive I/O activity. Thus, performance of the peripheral devices is a very significant factor in determining thruput. Peripheral device performance between the native system and emulating system may generally be compared on two accounts: per-character data rates and start/stop times. Since I/O activity can be (and usually is) overlapped with internal processing functions, an emulating system with devices twice as fast as the native configuration and a minimum of non-overlapped processing will not have an equivalent increase in thruput.

Any ECP overhead time required to translate I/O requirements into Spectra equivalents adds to the execution times of the I/O instructions. ECP device initiation/termination sequences are by their very nature non-overlapped with any emulation processing time. Device initiation or termination times may only be overlapped with I/O activity on another I/O channel if such activity is required by the program being emulated. To this extent, ECP overhead does not quite add directly to total program execution time. However, the effects of ECP overhead are most noticeable in I/O bound programs handling short records. Per-character I/O processing is an ECP function and since it occurs at initiation or termination time, can be considered as an extension of the ECP overhead factors.

The most complex factor affecting performance is the timing distribution of internal processing, I/O activity, and ECP overhead functions. These interrelationships can cause idle processor or I/O time at points in the emulated program where none existed in the native system. Evaluation of this factor in the design of emulators was only attempted through timing simulation runs. The effects and interrelationships were too complex to predict by a simple set of ground rules. Actual program running times have been measured on the 301 and 1410 emulators for the 70/45. The thruput ratios range from 1.30 to 2.95 on the 301 and from 1.89 to 2.66 on the 1410. The variability of thruput reflects the interplay between the four performance factors described above.

CONCLUSIONS

The emulators, as described in the article, are currently operational. Field experience has proven them to be a valuable program conversion technique. It has been demonstrated that a user program can be run directly on the emulator without reprogramming and with an actual increase in performance. Emulation has been made economically feasible, primarily because of the use of read-only memory in the design of the computer control and the improved cost-performance ratio of third generation hardware.

Emulation, on the other hand, has not solved the reprogramming problem faced by a user who wants to take full advantage of third generation hardware. It does, however, permit him to spread his reprogramming investment over a longer period of time. Although the primary purpose of emulation is to ease the user's burden of reprogramming, it has, in certain instances, actually reduced the total rental cost.

ACKNOWLEDGMENTS

The emulator design has been the joint effort of many groups and individuals within the EDP organization. Special mention is made of Engineers and Programmers within the Emulation Design group and the designers of the Spectra 70/35 and 70/45 systems, without whose cooperation this program would not have been possible.

REFERENCES

1. McGee, "The TRW-133 Computer," *Datamation*, Vol. 10, No. 2, p. 27,
2. Boutwell, "The PB-440 Computer," *Datamation*, Vol. 10, No. 2, p. 30,
3. Beck and Keeler, "The C-8401 Data Processor," *Datamation*, Vol. 10, No. 2, p. 35,
4. Cambell and Neilson, "Microprogramming the Spectra 70/35: *Datamation*, Vol. 12, No. 9, p. 64,
5. Yen, "Internal Logic Structure of the 70/45," RCA reprints PE-257, PE-258.
6. Form 70-35-601, *70/35—70/45—70/55 Processors Reference Manual*, RCA, Electronic Data Processing Div., Cherry Hill, N.J.