GENERAL MOTORS/NORTH AMERICAN MONITOR FOR THE IBM 704 COMPUTER

Robert L. Patrick

January 1987

P-7316

The RAND Corporation, 1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138

ABSTRACT

The history of software in the United States has been somewhat under documented. In an attempt to capture the essence of some of this early work before it becomes lost, the Committee organizing the 1987 National Computer Conference (June 15-18, Chicago) has designated June 17, 1987 Pioneer Day.

Robert L. Patrick, a RAND consultant for 27 years, was invited to prepare and present this paper on some operating system work he accomplished in the mid-1950s. The developments reported here contributed to a series of operating systems for major IBM computers: SOS, IBSYS, and with the substitution of disk for tape, OS/360.

FIGURES

GENERAL MOTORS/NORTH AMERICAN MONITOR FOR THE IBM 704 COMPUTER

The roots of the General Motors/North American Aviation effort go back to 1954. At that time I was working for an aircraft foundry in Ft. Worth, Texas. Picture a bunch of miscellaneous college graduates (computer science hadn't been invented yet) trying to understand the physics of aircraft design, formulate mathematical design models, and program these solutions for a giant brain. Only 19 IBM 701s were manufactured. The Ft. Worth machine was serial #7. The 17th machine, which I will discuss later, was installed at General Motors in Detroit. The 701 would only execute about .15 million instructions per second. It rented for about $23,750 a month, and filled a 40 foot by 40 foot room.

Architecturally it was a single sequencing machine and could do only one thing at a time (cycle-stealing channels hadn't been invented yet). Thus when a 701 addressed an input/output (I/O) device, that's all it could do until the data transfer was completed. It was a batch machine and a typical configuration consisted of a 150-card-per-minute reader, a 100-card-per-minute punch, a 150-line-per-minute printer (48 characters), and an internal memory of two thousand 36-bit words. It also had four magnetic tapes whose transfer rate was 7.5 thousand characters-per-second, and a magnetic drum which held two thousand words. The mean time to failure was about 30 minutes if you were lucky. (Fig. 1.)

The typical mode of operating was programmer present and at the operating console. When a programmer got ready for a test shot, he or she signed up on a first-in, first-out list, much like the list at a crowded restaurant. The programmer then checked progress frequently to estimate when he would reach the top. When his time got close, he stood by with card deck in hand. When the previous person finished or ran out of allotted time or abruptly crashed, the next programmer rushed in, checked that the proper board was installed in the card reader, checked that the proper board was installed in the printer, checked that the

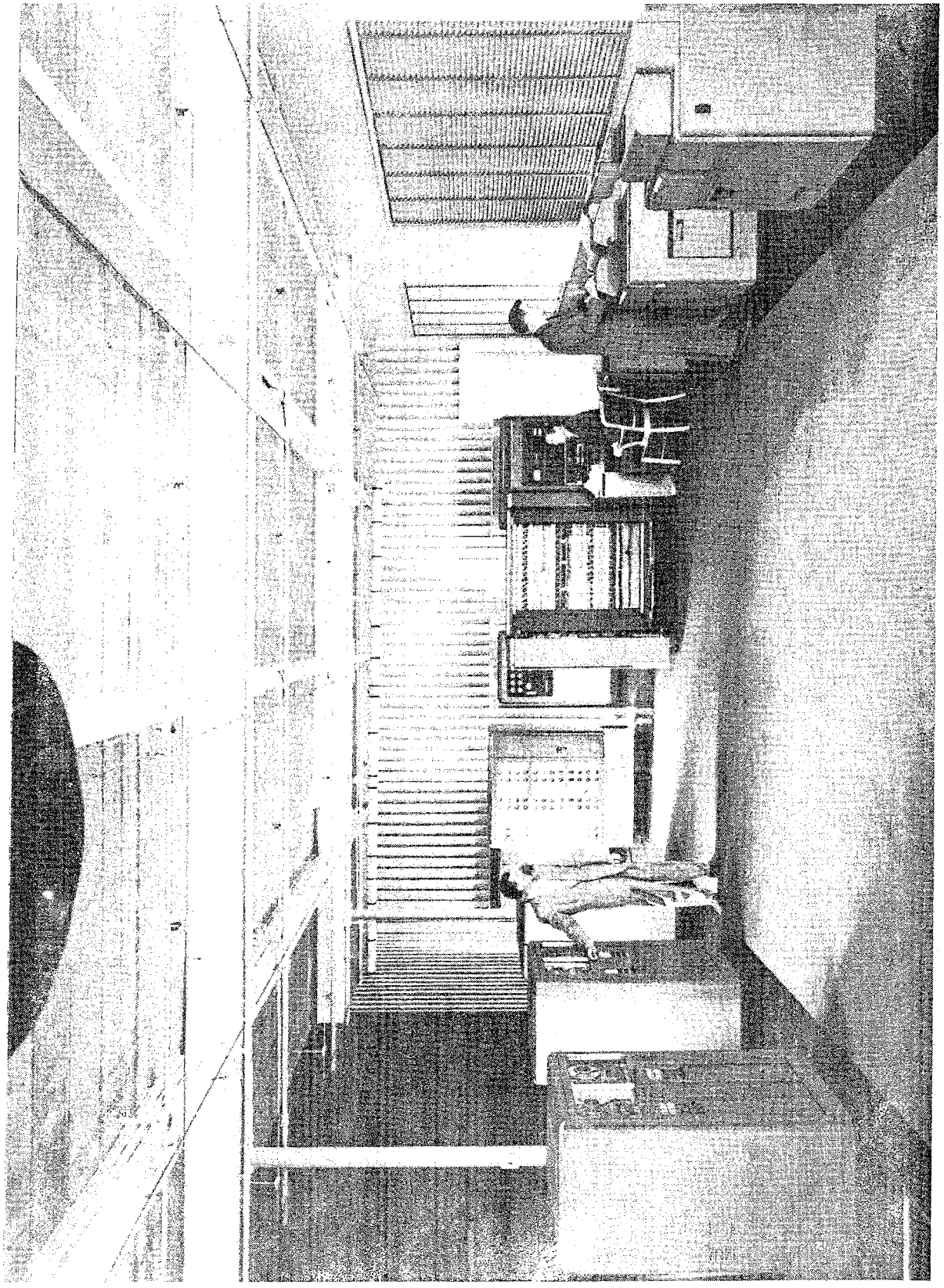

Fig. 1. -- IBM 701

proper board was installed on the punch, hung a magnetic tape (if he was going to use a master tape), punched in on a mechanical time clock, addressed the console, set some switches, loaded his punched card deck in the card reader, prayed the first card would not jam, and pressed the load button to invoke the bootstrap sequence.

If all went well, you could load a typical deck of about 300 cards and begin the execution of your first instruction about 5 minutes after entering the machine room. If only one person did all this set up and got going in five minutes, he bustled around the machine room like a whirling dervish. (Fig. 2).

Not always did things go smoothly. If a programmer was fumble-fingered, cards jammed, magnetic tapes would not read due to defective splices, printer boards or switches were incorrectly set up, and it took 10 minutes to get going; or worse -- you lost your opportunity and the next guy took the machine when your time ran out. Usually the machine spent more time idle than computing. We programmers weren't paid very much and although the machine was fairly costly, its capacity was even a more precious commodity since there were only 17 in the whole world and only one in the entire state of Texas.

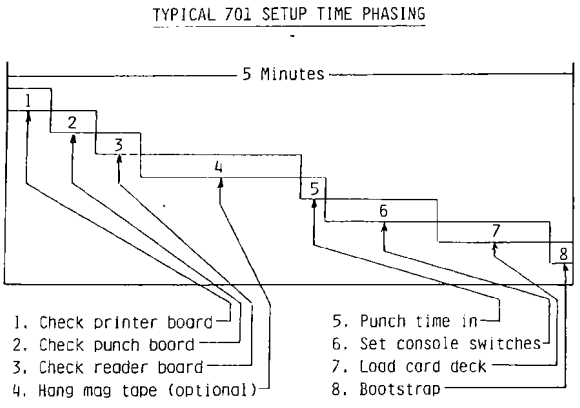

Fig. 2. -- Typical 701 setup time phasing

Now I had never heard of Henry Laurence Gantt, the father of industrial engineering, but if I had, this story would be a lot shorter. Seeing idle time on a precious resource caused several of us in various locations around the country to start a drive for efficiency that is still with us today. Informally we programmers started operating in teams so each programmer would have an assistant to help him get set up and going faster. That helped the fumble-fingered the most. We also gradually standardized on plug boards for the reader, printer, and punch to eliminate some of the board changing. We did some preventive maintenance on our card decks to reduce card jams and we tried to get programmer-operators better organized so they weren't so befuddled at the console (not entirely successful). All of this reduced the worst-case idle time between runs, but the average productive use of the machine was still poor.

About this time I was an obnoxious young hot shot and found a way to get my test shots run in the idle time between other programmer's scheduled runs. I ran only with standard plug boards so I never changed any boards. I bootstrapped my programs from an unused magnetic tape instead of punched cards and this reduced my program loading time from about three minutes to ten seconds. Further, each input record loaded to a unique address so I could store the program on tape and overload patches through the card reader. I could get a full test shot in three minutes or less if my output did not exceed 150 lines of printing. (Fig. 3).

Now this may not sound like much in 1987, but I got a test shot whenever I wanted just by squeezing in on the schedule. I was getting four and five shots a day when the others were lucky to get two. Furthermore, my total machine time was only 25 percent of what everyone else used.

Programming in the early 1950s could best be described as organized chaos. The typical programmer used assembly language, chose his favorite binary-to-decimal conversion routines from a library, figured out how to assemble and debug his programs as best he could from the

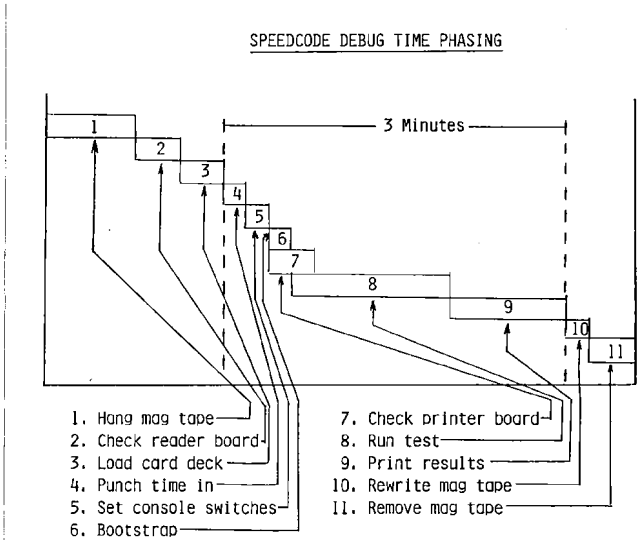

Fig. 3. -- Speedcode debug time phasing

primitive tools available, and at long last got a program checked out which may have been a work of art, but was always later than when the customer wanted results.

I was a premier user of an interpretive programming package developed by John Backus and his colleagues at IBM that was called the IBM 701 Speedcoding System. Speedcode was an integrated set of programs and provided packaged I/O, and a simplified three-address language with floating-point operands for programming. The purist programmers treated Speedcode with disdain, but I got a lot of work done and learned several lessons about machine room operation. Specifically:

1. Get the programmer off the console and out of the machine room.
2. Standardize on machine setups to eliminate setup steps.
3. Avoid reloading your whole job each time, maintain a magnetic tape copy, and update storage with a change deck.
4. The use of an integrated package of programs produced results much faster than with 100 percent custom programming and worked satisfactorily for all but the biggest compute-bound jobs.

In 1954 I got tired of defense applications and moved to General Motors Research in Detroit. They were just installing 701 serial #17, and I took Speedcode and my mode of operation with me. The results were much the same even though my engineering applications then dealt with automobiles and automotive gas turbines.

In mid-1955 General Motors had an IBM 704 on order and George Ryckman was assigned to lead the team to plan for its installation and operation. I was on that team and played a role which today we would call architect. About that same time the IBM user's group, SHARE, was being formed to allow the member installations to exchange programs and ideas for their mutual benefit. The third meeting of SHARE was held in Boston on November 10 and 11, 1955.

Prior to that meeting General Motors published a description of an operating system for the IBM 704. There was a competing proposal from North American Aviation. After reviewing the system sketched by General Motors, the North American representatives decided to join with us, convinced us to make some modifications, and as a result we both got our operating systems for about 50 percent of the price, less than 50 percent of the elapsed time, and with much higher quality because of the talent available in the combined team.

The resulting operating system built upon my experience with the 701 and exploited the 704 hardware. The General Motors/North American Aviation effort was eventually used in about 20 of the IBM 704 installations. There were people associated with the system as developers or as users and improvers, who moved on to be participants in SOS, IBSYS, the Direct Couple for the 7094, OS/360, IMS/360, and JES.

Before proceeding with a discussion of the operating system we produced, let me momentarily digress and describe the IBM 704. The 704 was a grown up 701. It was faster, and the storage tube memories on the 701 had been discarded in favor of coincident-core memories on the 704. That simple expedient, coupled with the next generation of electronic technology, increased the mean free-error time to about eight hours.

The 704 instruction set was enhanced with some logic instructions and a set of binary floating-point arithmetic instructions. (It had no built-in decimal capabilities.) Internally the 704 was more than twice the speed of a 701.

In January 1957 the 704 installed at General Motors in Detroit had 8KW of core, four drums of 2KW each, eight mag tape units, and the same slow card reader, printer, and punch as had adorned the IBM 701. The base rent was $35,550 a month. Thus the available memory was greater, the speed was up, but the primary I/O equipment was the same. Further, the mainframe was still single sequencing (cycle-stealing channels didn't come along until the IBM 709), so one devoted 100 percent of the central processing unit (CPU) to any input/output operation undertaken. (Fig. 4). We recognized we had to change our mode of operation to achieve full benefit of the 704 over the 701.

IBM had realized that some large commercial accounts would have a lot of cards to read and had many checks to print. So they offered, as an extra cost option, three separate stand-alone devices to convert card images to magnetic tape images, to take tape images and print them, and to take tape images and punch cards. Since these were stand-alone, they were in some ways better than cycle-stealing channels as there was absolutely no interference to the processing being performed by the mainframe in the next room.

The following highlights of the operating system we developed were extracted from the 1955 SHARE proposal (referred to earlier) and a programmer's manual we found that was dated January 31, 1957. Thus fourteen months after the proposal, we put together a team, polished the design, implemented the system, and documented it. The following list contains the highlights of the joint General Motors/North American effort:

1. All input and all output were processed on the off-line card-to-tape equipment. These produced and received tapes containing files now known as SYSIN and SYSOUT. The mode of operation was then known as tape-to-tape since the reader-printer-punch attached to the mainframe were used only as extensions to the operator's console and for maintenance. They were not used for any production input or output.

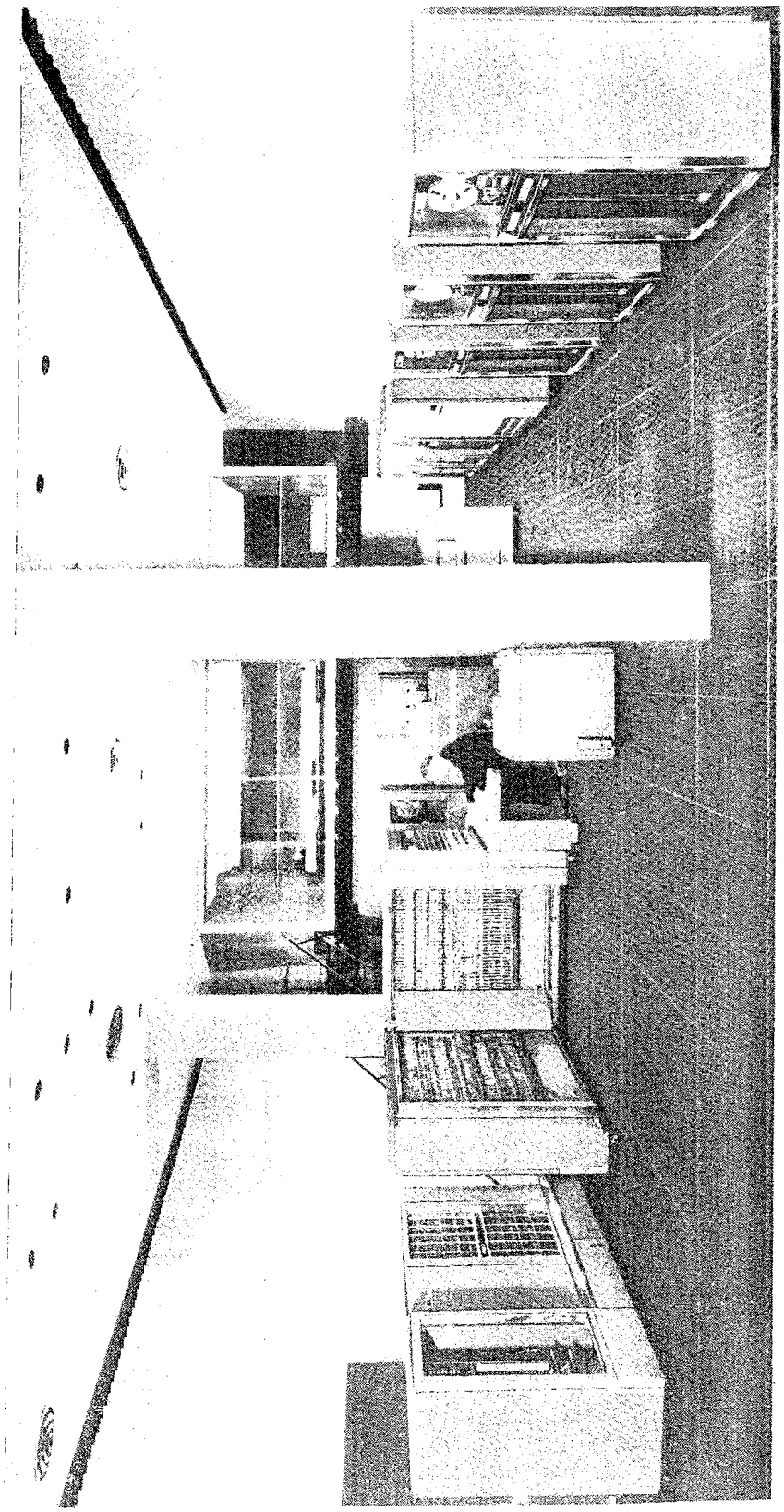

Fig. 4. -- IBM 704

2. As in current practice, the SYSIN tape contained a batch of independent jobs, and each job was identified by control cards which we would now recognize as JCL. The sequence started with a job card which contained accounting information and the programmer's name. Each pack of cards that followed the job card contained header information which controlled its conversion from decimal to binary, and then its subsequent processing.
3. Although IBM provided plugboards in every reader, printer, and punch, we set up installation standards and figuratively welded them into the machine. Thus all the data reorganization was done by programming and not by board wiring. This standardized the setup and reduced the unproductive inter-job time from minutes to milliseconds.
4. We threw the programmers out of the machine room, hired professional computer operators, and standardized the communication between the programmer and the operator. The programmers caught on to the new mode of operation rather quickly since we instructed the operator to proceed to the next job when there was confusion or something didn't work out.
5. We gained programming efficiency through the use of standard decimal-to-binary/binary-to-decimal conversion routines and standard (built-in and integrated) debug tools. This reduced the programmer's effort since it didn't require understanding the library routines. He merely had to write some job control statements to use them. This had another important benefit to the programmer. The memories on these machines were relatively small, only 8KW. By providing all of the decimal-binary translation outside the programmer's domain, we provided an execution time environment that had binary input, computed in binary, and produced output in binary. This gave more usable space to the application than would have been possible for any but the most experienced programmers handling their own input and output, in customized fashion.

6. We augmented the IBM hardware with a custom designed, locally produced, binary time-of-day clock. Further, the system was programmed, just as modern operating systems are, to sample the clock whenever a job card was read for a new job, and to record the resources used during the run so proper accounting records could be maintained. Today we call these SMF records. We even went a step further because we placed an advisory invoice on the trailing page of each printout so the programmer was aware of the resources used and the cost of those resources each time a run was made. We found the resulting self-discipline of great benefit since programmers naturally do more desk checking when a wasted shot at the machine costs more than a day's wages.
7. With the primitive JCL we had on the SYSIN file, it was possible to assemble a program, load it into memory, present it with a data deck, and record the results of the assembly and the execution in a single pass on the machine. It was well into the '60s before this "load-and-go philosophy" became generally popular.
8. Further, we designed the system, within limits, to be extensible so it was possible to add other language translators to the menu of input conversion routines. After I left the project, the other members of the system's team at General Motors added Fortran as an input translator so it was possible to compile and execute in a single pass canned library routines, assembly code, and Fortran programs.

The implementation of this system was largely dictated by the small core memory available and the speed of the I/O devices. As discussed earlier, the peripheral card reading equipment produced card images on tape. The data on these tapes was still decimal as no conversion had taken place. Further, the printer and the punch required decimal tape images as they too lacked any ability to convert. The 704 mainframe only had an 8KW core memory. Due to the small size of this memory and due to the large size of a general purpose decimal-to-binary/binary-

to-decimal conversion routine, the system was conceived and implemented in three distinct processing phases. (Fig. 5).

In the first phase, the binary coded decimal card images were read from tape, control cards were interpreted, and a binary file was created. Logically the binary file was the direct analog of the BCD image tape that had been created on the peripheral card reader. However, it was more compact and was in the native language, e.g. binary, of the CPU. A binary read program took only about 50 instructions. A decimal-to-binary conversion program took almost 2,000.

Similarly, each job during execution wrote a binary output file. Control statements, just like JCL, were imbedded in the output stream. These control statements caused the output translator to convert and format the binary stream properly and subsequently to include parameters on the decimal output file to control printer spacing, skips to the head of form, and insert separator cards between decks of punched card

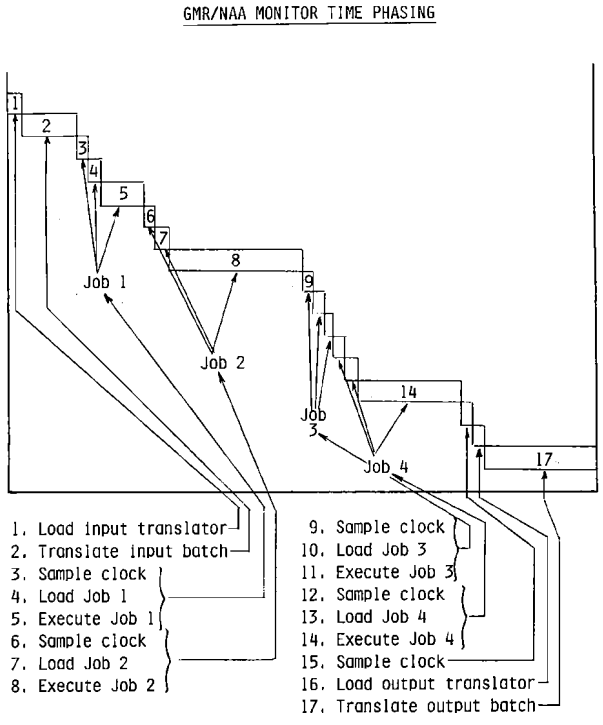

Fig. 5. -- GMR/NAA monitor time phasing

output. Again to conserve memory and gain speed, the output translator was given the whole machine and conversion was optimized.

The resulting operating system had an input-translation phase which converted data from decimal to binary, and programs from source to object language (symbolic assembly language was initially available, soon after we added the then-new Fortran); an execution phase which was almost exclusively under the programmer's direct control; and an output-translation phase which processed line printer output, punched card output (both decimal and binary), and accounting records. To recap, the advantages of the three-phase design were:

1. It provided the maximum amount of unencumbered memory to the application programmer during execution.
2. It was efficient since the conversion routines were bigger than the sub-files for individual jobs. The big conversion programs were loaded into memory only once per batch, and all of the little data files were passed by them, resulting in a smaller number of total cycles than if the large translator routines were fetched and executed at the beginning and end of each job.
3. By giving the translators the full machine during the conversion process, the translators could be made as large as necessary to gain efficiency in execution and hence were optimized to be faster for the conversion tasks they had to do.

When the resulting code was written, 90 percent of the work was devoted to the input and the output translators, and only about 10 percent of the effort was devoted to supporting programmers during the execution phase.

When North American Aviation and General Motors Research decided in the winter of 1955 to jointly develop the system, we split it right down the middle. North American developed the integrated programs that became the input-translation phase, and GM developed the output translator. In the spring of 1956 GM extended the North American input translator to call Roy Nutt's symbolic assembly program (SAP) as an imbedded conversion subroutine during phase 1 input translation.

There was an intellectual disagreement concerning the services the operating system should provide to the programmers during phase 2, execution. At this point the software installed at North American and at General Motors was dissimilar. The software for phases 1 and 3 was identical at both locations, but we each had our own separate execution time facilities.

When we first started planning this system, we were concerned about protecting the execution-time facilities. The 704 had no limit registers or any other way to partition off and protect the operating system from the application programmers. (These didn't come in the IBM world until the advent of System/360.) Furthermore, we had only one class of instructions and every programmer had access to all of them. (The concept of privileged instructions also came about in 1964 with the System/360.) As it turned out, we needn't have worried on either count. Our application programmers were company-loyal and not mischievous. Most of them were delighted to have someone else worry about conversion translators and modes of operation. While we had a few application programs that ran away during execution and destroyed the operating system in low core, this was a relatively infrequent occurrence. I cannot remember a single instance where a programmer maliciously tampered with the operating system.

We were also worried about programmer acceptance of operating system standards. During the 701 days, programmers had unfettered access to all of the 701's hardware features. Some questioned whether programmers would fight the system since it denied them access to certain machine features (the operating system claimed a tape as a SYSRES device and the first 300 words of core during the execution phase). Perhaps it was because we installed the operating system when we installed the hardware and had no legacies to deal with, but in Detroit we had no serious attempts by headstrong programmers to read cards on-line, demand all of the tapes for their own personal use, or otherwise contravene the standard configuration we set up. This may have been partially due to our willingness to allow any programmer to violate the system constraints, but he was advised that if he did, his jobs would only run programmer present and only in the wee hours of the morning.

Of the direct contributors to this pioneering effort there were several. In the past 30 years many of us have moved several times. Recently some of the team has retired, others of us are still plugging away. Fig. 6 lists all of the original players with their original affiliations.

CONTRIBUTORS TO THE
GMR/NAA MONITOR

Individual	1955 Affiliation
Penny Barbe	NAA
Robert Christiansen	GMR
Jim Fishman	GMR
Dale Hanks	NAA
Don Harroff	GMR
Don Hart	GMR
Owen Mock	NAA
Bob Patrick	GMR
George Ryckman	GMR
Kei Shimizu	NAA
Frank Wagner	NAA

Fig. 6. -- Contributors