This paper compares factors governing the implementation of the IBSYS/IBJOB *operating systems for the* 7090/94 *and* 7040/44 *processing systems.*

Operation of the 7040/44 *system's monitors and the means of communication between them are described.*

Familiarity with the content of the previous five papers of this series is assumed, though for the most part, the paper can be read independently.*

Design of an integrated programming and operating system

Part VI: Implementation on the 7040/44 data processing system

by B. White and J. Trimble

Following the decision to produce IBSYS/IBJOB for the 7090/94, it was decided to implement a similar system for the 7040/44. The transfer of a system of the size and complexity of IBSYS/IBJOB to another machine required a re-evaluation of all parts of the system and considerable new design work. Although generally the 7040/44 Operating System is externally compatible with the 7090/94 IBSYS/IBJOB system, there are significant differences in the internal structure. This paper describes the 7040/44 Operating System as well as the differences between this system and the 7090/94 IBSYS/IBJOB system.

The first step in planning the 7040/44 Operating System was a survey of the 7090/94 system to determine what elements could be copied directly. It became evident almost immediately that very little of the 7090/94 system could be used without change. In addition to obvious differences in hardware, there were other important considerations not common to both machines. At this point, a number of basic decisions concerning compatibility goals were made. It was decided that compatibility from the view of the application programmer (in the use of FORTRAN, COBOL, MAP, and SORT) was vital. Compatibility from the view

* Parts I through V have been published in the *IBM Systems Journal* as follows: Parts I and II in Volume 2, June 1963; Parts III, IV, and V in Volume 2, September–December 1963.

of the system programmer (in the use of operating system control cards[1]) should not be arbitrarily different. However, internal compatibility was deemed impracticable. Although initial plans called for IOCS compatibility at the buffering system level, later developments led to an alternative approach. All incompatibilities introduced were due either to the different 7040/44 environment or to advances in programming technology.

System differences

historical considerations

The designers of 7090/94 IBSYS/IBJOB had to contend with a number of historical considerations. Many 7090/94 users had progressed from the 704 to the 709 and then to the 7090 or the 7094. These users had an investment in many systems: FORTRAN II, FAP, Commercial Translator, etc. The implementation of these systems under a common monitor, the IBSYS basic monitor, had to be accomplished without significantly changing their operating characteristics. Thus, the design of the IBSYS monitor, and later the design of the IBJOB monitor, were significantly influenced. The IBSYS monitor was forced to play a minimal role, whereas the IBJOB monitor had to assume functions that otherwise could have been performed at the IBSYS level.

The programming support for the 7040/44 did not include these earlier systems. It was therefore feasible and desirable that the 7040/44 IBSYS monitor assume more control of the operation of subsystems, with the imposition of additional standards. All 7040/44 subsystem components depend upon IBSYS for inter-system communication, all use 7040/44 IOCS, all are assembled by IBMAP, all are maintained by the IBEDT system, all accept input and produce output in a standard peripheral format, and all are capable of operating within a fully labeled system. These standards, though they make possible a greatly simplified system and one which is more fully integrated, do not prevent expansion of the system to include new processors.

16 and 32K memories

The factor which had the most widespread effect on the design of the 7040/44 system was the requirement that it operate within 16K of memory. The 7090/94 IBJOB system as originally distributed required approximately 9000 locations for supervisory and IOCS subroutines during subsystem execution. In a 32K system, this allows 23,000 locations per subsystem phase. Transferred to a 16K 7040/44, this structure would allow only 7000 locations per subsystem phase, with curtailed IBJOB facilities, impaired efficiency, and unreasonably small processor tables. Instead, four changes were incorporated to make more space available for subsystem phases.

First, the input/output editors were written as relocatable subroutines and placed in the *relocatable subroutine library*. In this way, the *input editor* is included in only those subsystem phases which require records from the *system input unit*, the *output editor* is included in only those subsystem phases which write on the *system output unit*, and the *punch editor* is included

only in those subsystem phases which use the *system punch unit.*

Second, in a similar manner, the file control blocks for the system I/O units were placed in the relocatable subroutine library; hence they are included, with their buffers, only in those subsystem phases which require them.

Third, a *post execution routine* (a housekeeping routine which assures proper return to the monitors) was written as a relocatable subroutine and placed in the relocatable subroutine library for inclusion in object programs. It is not included in the compilers, assembler, or loader and is the only part of the IBJOB monitor in memory during execution of object programs.

Fourth, a small set of nucleus control subroutines (used by all subsystems) was developed that can locate and load subsystem phases from the *system library.* This set of control subroutines is capable of operating with abbreviated control information over a single subsystem. In any situation where control must be transferred outside the current subsystem, the control subroutines locate and load the IBSYS monitor and return control to it.

With these four changes, the amount of storage required for permanent subroutines was reduced to approximately 5000 locations, leaving 11,000 locations available for subsystem phases.

The 11,000 locations of storage were considered sufficient for implementation of the subsystems; however, considerable redesign of each of the subsystems was necessary. In general, each subsystem component required more phases than its 7090/94 counterpart, and in many instances information handled on the 7090/94 in internal tables required external file handling on the 7040/44.

In addition, each subsystem had to be planned for effective usage of additional memory if available. For example, IBMAP symbol and macro tables are automatically increased, and IBLDR processes files internally instead of externally when run on a 32K machine.

cycle time

The 8-microsecond memory cycle of the 7040, as contrasted with the 2.18-microsecond cycle of the 7090, also posed a critical problem. To compensate for this, the 7040/44 compilers went a step further than their 7090/94 counterparts in reducing the amount of character manipulation performed internally. Whereas in the 7090/94 system the interface between the compilers (FORTRAN and COBOL) and IBMAP is in the form of BCD text, the equivalent interface on the 7040/44 system is in the form of internal binary records and prepared tables. This required two first phases for the 7040/44 IBMAP, one for MAP language inputs, the other for the compilers' outputs. IBMAP processes after the first phase are the same for both input types.

The decision to change the compiler-assembly program interface affected only internal processing; however, another decision, which was primarily a result of the speed differences of the machines, did have direct effect on the users. The original plan to implement the 7090/94 *buffering system* on the 7040/44 was

abandoned. A minimum of 300 cycles was required for each entry to the generalized read-write routines of the 7090/94 buffering system. Implementation of the same buffering system on the 7040/44 would have required approximately the same number of cycles per entry, and hence significantly more time, particularly on the 7040. Timing studies revealed that under these conditions, input/output on the 7040 would become process-limited in the buffering system at an unreasonably low blocking factor. An alternate buffering system, oriented toward processing logical records (commonly known as GET-PUT logic) was developed for the 7040/44. It is possible with this buffering system to process a logical record in approximately 60 cycles.

multiple I/O devices

Another factor which heavily influenced the redesign of IOCS was the requirement that the system support a multiplicity of input/output devices. This was known at the beginning of the 7040/44 project, whereas additional devices were attached to the 7090/94 after implementation of an IOCS designed only for card/tape. For this reason, it was possible to make the IOCS for the 7040/44 device independent for all sequential files. An intermediate level of IOCS was created to handle all problems of device type. All of the higher levels of IOCS (the file labeling and label checking procedures, and the buffering system) operate through this intermediate level and are therefore unconcerned with the type of device used. Accordingly, all programs (either user programs or system programs) that make use of IOCS for operations on sequential files are device independent. The 7040/44 IOCS currently supports multiple card equipment, an on-line IBM 1401 data processing system, 1301 disk storage units, 7320 drum units, and 729/7330 tapes. I/O operations on nonsequential files are supported through standard IOCS calling sequences. The design of IOCS is such that new devices can be added with minimum change to the intermediate level of IOCS and without change to any other system part.

memory protect, storage clock

The 7040/44 programming system includes support of two hardware features that did not have to be considered by 7090/94 programming: memory protect and a storage clock. Support of the storage clock affected only the design of the monitors. The memory protect feature, however, required serious consideration in the design of IOCS and the monitors, and imposed conventions on all subsystems.

instructions and channel commands

A number of instruction set and channel command differences exist between the 7040/44 and the 7090/94. The channel command differences were absorbed by the redesign of IOCS and the formatting of the system library. The lack of a convert instruction, though counteracted somewhat by 7040/44 character handling instructions, made infeasible the use of many 7090/94 scanning techniques. The lack of sense indicators, the existence of the transmit instruction, and the existence of the transfer and store location counter instruction led to other differences in coding techniques.

later design

Implementation of the 7040/44 Operating System was begun considerably after the start of the 7090/94 system design. This "time lag" was both advantageous and disadvantageous to the 7040/44 system. On the one hand, many design problems had been solved by the 7090/94 planners before the 7040/44 work had begun. Also, problems common to both systems were uncovered during the extensive testing of the 7090/94 system. On the other hand, maintaining compatibility with a system undergoing change was a difficult task for the 7040/44 programmers. In addition, portions of 7090/94 coding that might have been copied, for the most part were not yet debugged when they were needed by the 7040/44 programmers.

Another advantage was the opportunity to view customer reaction to the initial version of the 7090/94 system. For example, the need for better job definition and for an improved method of symbolic input/output channel and unit assignment was made apparent. It was possible in the 7040/44 system to satisfy the first of these requirements immediately. The nature of the 7040/44 input/output unit assignment procedure made possible the use of the system without symbolic channel assignment abilities until an improved channel assignment scheme could be defined.

Another system difference appears in the 7040/44 SORT program. A new internal sort technique, which increases the probability of creating long strings, had been developed. The advantages offered by this new technique, coupled with the ease of moving records via the transmit instruction, and the lack of scatter-gather input/output channel commands made desirable a change from 7090 SORT techniques.

Organization and operation

The remainder of the paper describes the general organization and operation of the 7040/44 Operating System centering around the novel organization of monitors and special facilities.

We start with a description of the system library format since it is so closely associated with the operations of the various control routines. The system library is composed of *phases* (absolute core storage loads), each of which consists of a series of physical records (blocks).

Figure 1 illustrates the format of a program phase in the system library. The first word of the first block contains the name of the phase (NAME). This word is checked whenever a phase is requested to ensure that no mispositioning of the library unit has occurred. The blocks constituting a phase are chained together using the last word of each block. In every block except the last, this chain word is signed positive and contains the load address (LAk) and the length (LNk) of the following block (the kth). In the last block, this word is negative and contains the address of the first instruction to be executed in the phase (EP). Note particularly that the load address and length of the first block are not present in the records.

Figure 1 System library phase format

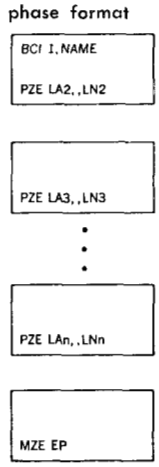


table of contents

The table of contents consists of a three-word entry for each phase of every system component that has been edited into the system library. These entries are placed in the table in an order which reflects the usual flow of control from phase to phase. For a particular component, some phases may also be used by other components, e.g., the phase of MAP which is used by both the FORTRAN Compiler and the COBOL Compiler. Such phases have multiple entries in the table.

Each three-word entry of the table of contents contains the name of the phase, the load address and length of the first block, a code number which indicates the appropriate library or utility unit, and the physical record number of the first block of the phase. Given this information and knowing the current position of the unit involved, any phase in the system library or on a utility unit is readily found.

index

In addition to the phase entries described above, the table of contents also contains an *index*. The index delimits the table of contents entries for each component in the system. An index entry is made up of two words. The first word contains the name of the component (such as IBMAP, IBLDR, etc.). The second word points to the table of contents entry for the first phase of the component and also indicates the total number of associated entries.

The table of contents and the index are updated automatically by the system library editor whenever an edit of the system library is performed. They are placed in the system library as a separate phase (IBTOC) so that they can be loaded either with the system editor or with the subsystem monitors. In order to facilitate locating IBTOC, it is always placed at a fixed position in the system library.

combined monitor

In the 7040/44 Operating System, the *supervisor* (corresponding to IBSUP in the 7090/94 system) and all subsystem monitors (except the SORT monitor) are contained in a single phase[2] and are referred to as the *combined monitor*. The table of contents (IBTOC) is loaded whenever the combined monitor phase is loaded. By combining the monitors, three things are accomplished:

- Communication between monitors is facilitated.
- System speed is improved.
- Use of common subroutines for similar monitor operations is made possible.

system monitor

The *system monitor* in the 7040/44 Operating System is the counterpart of the IBSYS *basic monitor* in the 7090/94 system. It consists of two parts, the *nucleus* (IBNUC) and the *supervisor* (IBSUP). The nucleus contains routines of general use to subsystems, and data and table areas necessary for system continuity. The supervisor processes system monitor control cards and is responsible for system continuity.

nucleus

The nucleus contains the following functional sections:

- Words allocated for machine use.
- System transfer points.
- System data areas.
- Control blocks.
- Nucleus routines.
- Lower levels of IOCS.

Table 1 shows the approximate core storage allocation for the nucleus and higher levels of IOCS.

Two subroutines, the *system library loader* and the *system return routine*, perform the main functions of the nucleus. They operate upon information from two tables, the *abbreviated table of contents* and the *recognizable control card table*, and from a work area, the *system save area*.

Table 1 The nucleus and higher levels of IOCS

Section	*Approximate size*
IBNUC	
Machine Functions	93
Pointers and Data Words	130
*Symbolic Units Table	1 word/device
*Unit Control Blocks	9 words/device
*System Control Blocks (Unit record or magnetic tape)	4 words/device
*System Control Blocks (1301)	8 words/unit
Nucleus Routines	600
Customer Engineer Monitor	50
IOEX	
Basic Functions	510
*Basic Functions Channel Tables	14 words/channel
*1301 Functions	140
*1301 Functions Channel Tables	4 words/channel 1.5 (approx) words/module
Interrupt Scheduler	150
*Interrupt Scheduler Channel Tables	1 word/channel 11 words/level
Memory Protect	40
IOOP	
Header	320
Magnetic Tape	350
*1301	500
*Unit Record	170
Common	120
IOLS	
Reel Processing	290
Labels	360
System	70
IOBS	940

*Variable, based on configuration.

The abbreviated table of contents is a nucleus table large enough to contain all of the table of contents entries for a single subsystem. Each subsystem monitor, utilizing the table of contents, initializes the abbreviated table of contents for the subsystem whenever it is entered.

The system library loader (not IBLDR) loads only absolute programs (the system component phases). It maintains phase-to-phase continuity of the system. A calling sequence to the loader determines which of the following functions are to be performed.

- Position the I/O device specified in the abbreviated table of contents, load a phase from the device, and verify the accuracy of the positioning.
- Initiate positioning of the device to the next phase indicated in the abbreviated table of contents.
- Transfer control to the phase loaded.

The system library maintains a pointer to the appropriate entry in the abbreviated table of contents and normally progresses through the table sequentially. The pointer, however, can be adjusted by a subsystem phase to permit execution of phases in any sequence. The system library loader uses the intermediate level of IOCS for all loading and positioning of the library.

The recognizable control card table contains a list of control cards that can be processed within a subsystem without returning to the monitor. This table, like the abbreviated table of contents, is made up by a subsystem monitor when it is entered. Each entry in the recognizable control card table contains the name of a control card and the location in the abbreviated table of contents of the first phase for the subsystem component associated with that control card.

The system return routine is the routine to which all subsystem monitors and their components return control when they desire to transfer control to the next appropriate component. It maintains component-to-component continuity of the system. The system return routine utilizes information in the system save area, which normally contains the information from a control card that has been read ahead by the program previously in control. The system return routine compares the card image in the system save area to the list of card names in the recognizable control card table. If the card name is recognized, the system return routine uses the system library loader to load the required subsystem component and to pass control to it. If the next card has not been saved or if it is not in the table, the system return routine causes the system library loader to load the supervisor and to return control to it.

The system library loader, the system return routine, the abbreviated table of contents, the recognizable control card table and the system save area, taken together, constitute a miniature monitor capable of operating over a segment of the library without any intervening help from the supervisor or the

subsystem monitors. Hence, it is possible within the 7040/44 Operating System for a subsystem to move freely from component to component without requiring the recall of the supervisor or the subsystem monitor between components. This feature has resulted in an appreciable time saving in the operation of the subsystems. The importance of this time saving can readily be observed in a FORTRAN application requiring the compilation of many subprograms.

In addition to the system library loader and the system return routine, five other subroutines exist in the nucleus. These are the *interrupt routine*, the *interrupt test*, the *change communication region routine*, the restore portion of the *restart routine*, and the bootstrap to the *dump routine*. The interrupt routine and the interrupt test are used to process interrupt signals initiated by the machine operator. The change communication region routine is used to modify the contents of protected memory. It is used whenever cells in the nucleus must be changed; for example, when a control card must be stored in the system save area. The restart routine is used to reinitialize core storage and to pick up processing at the checkpoint described in the calling sequence. The dump routine is used to take a system dump when an unusual condition occurs during processing. Both the restart routine and the dump routine have their main program parts in the system library.

In addition to these system routines, it is possible to include specialized routines in the nucleus, e.g., an installation's accounting routine.

The nucleus also contains hardware trap cells, a table of system transfer points, system data areas, the symbolic units table, the unit control blocks, and the system control blocks. The hardware trap cells are initialized in such a way that, upon trapping, control is transferred to the appropriate trap supervisor routine. The table of system transfer points contains transfer instructions to entry points in the nucleus routines and the IOCS routines. The system data areas contain constant information describing the system as well as variable data set by subsystem components for communication to other subsystem components. The symbolic units table, the unit control blocks, and the system control blocks provide information about the organization and availability of input/output units. These tables are described more fully in the discussion of the IOCS routines.

supervisor

The *supervisor* consists of a series of routines which are part of the combined monitor phase. Its principal functions are to dynamically maintain the configuration status description in the nucleus, to keep track of which subsystem is currently in control, and to screen control cards to ensure that no subsystem monitor receives an inappropriate control card. The supervisor also includes several subroutines of general utility to the combined monitors.

Maintenance of the system input/output configuration description in the nucleus is accomplished by a set of subroutines

in the supervisor. There is a subroutine for each of the following control cards:

- $ATTACH, which specifies that a certain unit be made available for use.
- $DETACH, which specifies that a certain unit be made unavailable for use.
- $SWITCH, which specifies that two units be interchanged.
- $CLOSE, which specifies that certain end-of-file functions be carried out.
- $RESTORE, which specifies that the standard installation I/O configuration be reestablished.
- $UNITS, which specifies that a description of the current I/O configuration be typed out.

Just as the system contains a table of contents for the entire system, the supervisor includes a master table of valid control cards for the system. This table contains information indicating the subsystem for which each control card is appropriate and the correct subsystem entry point for the processing of each card. Whenever a monitor, either the supervisor or a subsystem monitor, wishes to read a control card, it utilizes a general purpose control card read routine in the supervisor. This routine reads a control card, finds the appropriate entry in the master table of valid control cards, compares the subsystem indicated in that entry to a switch which indicates the subsystem in control, then either returns control to the appropriate entry point or initiates appropriate error-correction procedures. Generalized subroutines are available in the supervisor to assist any of the combined monitors in the creation of the abbreviated table of contents and the table of recognizable control cards, to check for uniqueness of utility units, to type control cards, to scan control cards, and to set the subsystem control switch.

The supervisor also contains routines that process control cards not associated with any particular subsystem. Included are the following:

- A routine that types the message from a $* comments card.
- A routine to acknowledge and process the operator's interrupt procedure—to change unit assignments or to start a special job.
- A routine to link to the installation accounting routine upon recognizing a $ID card.
- A routine to pause or suspend processing upon recognizing a $PAUSE card.
- Routines to process the $TIME card and to control the interval timer associated with the storage clock.
- Routines to initiate or suspend the typing of control cards upon the recognition of a $LIST or a $UNLIST card.
- A routine to terminate an interrupt job or the series of jobs on the system input unit after recognizing a $STOP card.

To facilitate convenient access to production programs that

are in the system library, the supervisor recognizes a $EXECUTE card.[3] When a $EXECUTE card is processed, the supervisor constructs the abbreviated table of contents for the program named and passes control to the program through the system library loader.

To facilitate *job* definition, the supervisor recognizes a $JOB card as initiating a series of related *applications*. Upon detecting this card, the supervisor performs all housekeeping necessary to initialize a new job. Job skipping facilities are provided as well as automatic skipping to the next job upon the occurrence of a terminal error within a series of applications.

processor

The processor monitor corresponds to the IBJOB monitor in the 7090/94 system. Due to the unique functions of the nucleus and the increased functions of the system monitor, the processor monitor has fewer functions to perform than those of the 7090/94 IBJOB monitor. The processor monitor is part of the combined monitor phase. It is not in memory during execution of its components or during execution of an object program. It more nearly resembles a housekeeping subroutine than a monitor with full system control properties.

The supervisor transfers control to the processor monitor when a $IBJOB card is read. The processor monitor has the following functions:

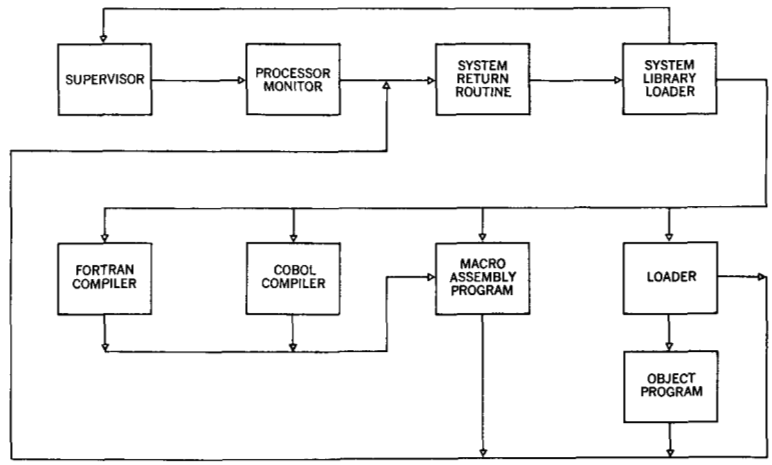
- It decodes the $IBJOB card information and saves it.
- It sets up the input/output utility units for its components and ensures availability of the units.
- It creates the abbreviated table of contents and the table of recognizable control cards using the $IBJOB card parameters.
- It processes relocatable loader preprocessor cards, such as $FILE and $LABEL, preparing the information for the loader and saving it on a utility unit.
- Upon encountering the first component control card, such as a $IBFTC card, it places the card information in the system save area and transfers to the system return routine. It should be noted that this is the same procedure, utilizing the same nucleus subroutines, as that performed by a component upon encountering a $control card following its input.
- It initiates and terminates alternate-input-unit and alternate-output-unit processing for processor components upon encountering $IEDIT or $OEDIT cards.

Figure 2 indicates schematically the flow of control during a processor application. Note in the illustrations that all components and the processor monitor give up control to the system return routine which may return directly to a component, to the supervisor, or to the processor monitor through the supervisor. Which return is made depends upon the contents of the system save area.

system editor

The *system editor* is used to maintain the system library of the 7040/44 Operating System. With the proper control cards, the user can add, delete, replace, or modify any phase of any absolute

Figure 2 Control paths for a processor application



program on the system library or add, replace, or delete any subroutine on the relocatable subroutine library (IBLIB). The system editor itself is incorporated in the system library as a monitored subsystem that is called via the system monitor.

The *system editor monitor* is part of the combined monitor phase. It is entered directly from the system monitor when a $IBEDT card is encountered. The system editor monitor processes the edit control cards and sets up the edit run in much the same manner as the processor monitor sets up a processor application run. The system editor monitor provides an example of the flexibility and integrated nature of the 7040/44 Operating System. It is possible in a single run to edit onto the system library a component written in any of the source languages which the processor monitor supports or any language for which a processor has been added by an installation. The system editor monitor, upon encountering a $IBJOB card, for example, gives control to the processor monitor, which remains in control until the processor operation is complete and then returns control to the system editor monitor. The operation of the processor under system editor monitor control is only slightly different from its operation in a normal application. The relocatable loader, operating under the processor monitor, uses the same format in preparing an absolute program for the system library as that used when an object program overflows memory. The relocatable loader, when loading a "large" object program, dumps the object program on a utility unit and lets the system library loader complete the loading function. Figure 3 indicates the flow of control during an edit run which makes use of the processor.

The system editor can accept the following types of input:

- MAP, FORTRAN, or COBOL source language coding.
- OCT instruction cards.
- Absolute column binary cards.
- Absolute code in system library format.

Whenever an edit is performed, the system editor inserts, deletes, replaces, or reorders the specified programs on the system library and creates a new master table of contents to reflect the changes. The system editor also maintains the relocatable subroutine library used by the relocatable loader.

SORT

The SORT monitor is the only subsystem monitor not included in the combined monitor phase. A small routine, entered by the supervisor upon encountering a $IBSRT card, is included with the combined monitor phase. This routine passes control to the SORT monitor. The procedures used in transfer of control between its components and return of control to the supervisor are identical to those described previously for the other subsystems.

IOCS

The 7040/44 Input/Output Control System has four logical parts:

- Input/Output Executor, IOEX.
- Input/Output Operations, IOOP.
- Input/Output Labeling System, IOLS.
- Input/Output Buffering System, IOBS.

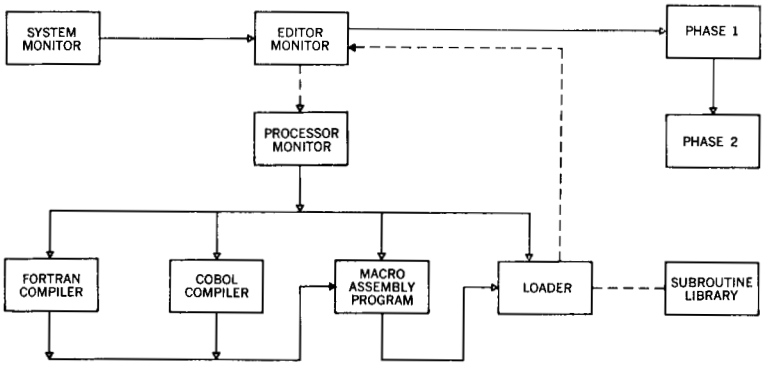
IOEX processes all data channel traps and schedules the use of data channels for input/output operations. It also services central processing unit traps and schedules the use of the central processing unit for programs of various priorities.

IOOP performs all reading, writing, and non-data-transmitting operations required by IOCS. IOOP handles all peculiarities of device type. It consists of two parts: IOOP1 which contains the calling sequence interpretation routine, the unit synchronizer routine, and the select and error recovery routines for 729/7330 Magnetic Tape units, 1301 Disk Storage, and 7320 Drum Storage; IOOP2 which contains the select and error recovery routines for the 1402 Card Read Punch, the 1403 Printer, and the 1622 Card Read Punch.

IOLS performs all processing necessary to verify and create IBM *standard labels* and performs necessary reel handling and unit switching functions. It uses IOOP to read and write labels.

IOBS processes logical records and performs the blocking,

Figure 3 Use of processor by the system editor



deblocking, and buffer switching functions of a typical GET-PUT buffering system. This buffering system is capable of handling fixed or variable length records. It also provides the user with an automatic means of preparing or accepting variable length, mixed mode records in the 7040/44 standard peripheral format. IOBS uses IOOP to perform all required reading and writing of physical records. It uses IOLS for all reel handling and label processing.

This structure of IOCS makes possible three distinct approaches to input/output programming. The programmer may use IOBS (requires the presence of IOEX, IOOP, and at least part of IOLS) or IOOP (requires the presence of IOEX; IOLS is optional) or IOEX (IOLS is optional—but if used, requires IOOP). Figure 4 illustrates the use of IOCS by the various system components and its optional use by an object program.

I/O configuration definition

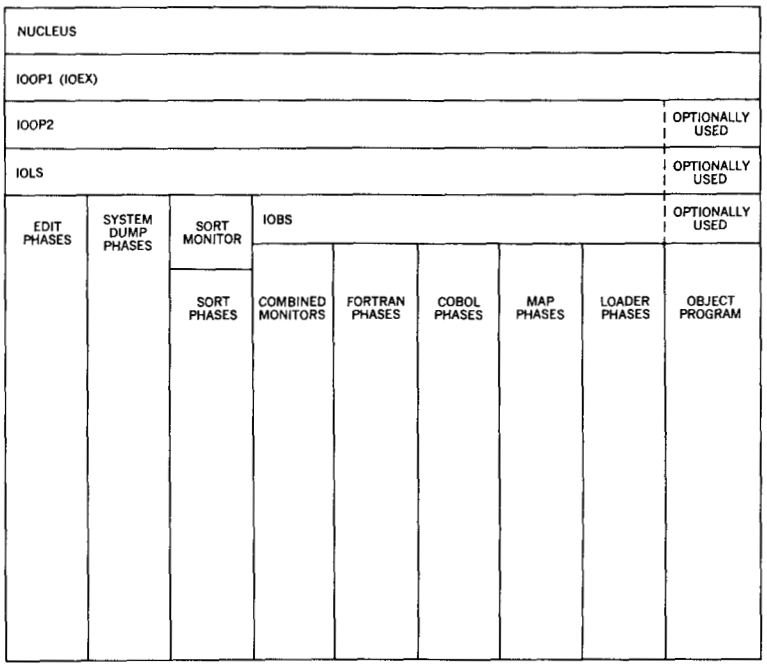
Every input/output device attached to the 7040/44 must be defined in the nucleus for IOCS and the Operating System. Three tables form the definition of an installation configuration.

- Symbolic units table.
- Unit control blocks.
- System control blocks.

These tables, and IOCS itself, are hand tailored through assembly to represent the particular installation.

The symbolic units table contains a one-word entry for each logical unit to be referenced. For magnetic tape, there is a one-to-one correspondence between logical units and physical units

Figure 4 7040/44 operating system use of core storage



(drives). However, for a 1301 disk, many logical units may be defined within the same physical unit (module). Each logical unit has a name, either a system unit name (e.g., S.SIN, S.SOU) or a utility unit name (S.SU*00*, S.SU*01*, ... , S.SU99), and each such name is associated with a particular entry in the symbolic units table.

Each entry in the symbolic units table contains two pointers, one indicating a particular unit control block and one indicating a particular system control block. For each symbolic units table entry (and thus for each logical unit) there is a unique system control block. For each physical unit there is a unique unit control block.

All references to units in the calling sequences to IOCP (or higher levels of IOCS) are in the form of system unit or utility unit names. From the associated symbolic unit table entries, IOCP can determine the system control block and unit control block to be used. Since these control blocks contain all necessary device-dependent information, IOCP can then convert the device-independent calling sequence into the correct sequence of operations. The particular control blocks associated with a given symbolic units table entry can be changed by the use of the $ATTACH, $DETACH and $SWITCH control cards.

additional features

The design of the 7040/44 Operating System is flexible and adaptable. A number of features have been implemented, and others, which go beyond the required functions of the initially released system, are in the planning stage.

For example, an object program may use the system dump facilities. The *system dump* program is normally used by system components when terminal errors occur. It can produce selective memory dumps, error messages, and formatted dumps of the I/O configuration tables. It may, upon completion of its dump function, return control to some specified return point or give control to the supervisor for initiation of the next job. An object program involved in certain standard error situations (e.g., violation of protected memory, parity errors) automatically uses the system dump procedures. However, an object program may use these facilities or add facilities for any type of detected error situation. This means that the user program need not be concerned with treating standard errors nor with transition procedures to the next job when such errors occur. Also, since the particular action taken for a given error condition is contained in the dump program itself and is easily accessible for change, an installation can design its own standard error procedures.

The 7090/94 IBSYS/IBJOB system includes an *overlay* feature which facilitates segmentation of an object program. The 7090/94 overlay method requires a highly sophisticated algorithm in the relocatable loader and was not planned to be included in the initial 7040/44 system. It was clear, however, that segmentation facilities were seriously needed by the 7040/44 users. A detailed study of some basic design features of the 7040/44 Operating System

revealed a simple way of obtaining the segmentation features required. The resulting scheme, less sophisticated than the 7090/94 overlay methods, is named CHAIN. It was easily implemented due to the ability of the relocatable loader to turn out absolute core storage loads in the format of the system library. The various segments of the object program are simply spilled onto a utility unit in system library format, an abbreviated table of contents is created to represent the series of segments, and a CHAIN subroutine is made up which controls the loading of the various segments (through the use of the system library loader). The implementation of CHAIN was accomplished in approximately five man-months, including testing, and was released with the system. It was completed early enough for use by nearly all of the subsystems.

It was mentioned previously that the entire 7040/44 system is assembled by IBMAP. All system parts, except nucleus and IOCS, are relocatable. When a user assembles nucleus and IOCS for his particular configuration, he re-edits the entire system, and all components are given new origins accordingly. This means that no two installations are likely to have exactly the same system. It is therefore impossible to release absolute changes during maintenance of the system. The 7040/44 Operating System is completely maintained in symbolic form. The entire system may be reassembled at any time, and a user is assured of having up-to-date IBMAP symbolic input. User changes to the system to further adapt it to his needs are greatly facilitated. Symbolic update programs are provided to assist the user in maintaining the system and his own programs.

The operation and maintenance of the 7040/44 Operating System (16K/32K) is fully described in the literature.[4,5,6] A fuller discussion of many of the ideas touched upon in this paper may be found there.

CITED REFERENCES AND FOOTNOTES

1. Control cards contain the information necessary to ensure an even flow of processing from application to application. Reference 5 describes the many types of control cards that are usable in the 7040/44 Operating System. All control cards have a $ symbol in the first column.
2. The combined monitor phase is not in memory during execution of subsystem components.
3. In the 7040/44 Operating System, a $EXECUTE card is used for production programs which have been edited into the system library. It is not used to pass control to subsystem monitors. Subsystem monitor control cards contain the subsystem name following the $ symbol.
4. *IBM 7040/7044 Operating System (16/32K): Programmer's Guide*, Systems Reference Library C28-6318, International Business Machines Corporation, 1963.
5. *IBM 7040/7044 Operating System (16/32K): Operator's Guide*, Systems Reference Library C28-6338, International Business Machines Corporation, 1963.
6. *IBM 7040/7044 Operating System (16/32K): Systems Programmer's Guide*, Systems Reference Library C28-6339, International Business Machines Corporation, 1963.