J. D. Colas

Operational Program for the IBM 2750

Abstract: This paper describes the operational program of the IBM 2750 Voice and Data Switching System. The program runs in the supervisor unit of the two network controllers used in the 2750. This program controls: (1) the duplexing of the network controllers; (2) the switching network; (3) the data collection and transmission operations passing through the system, including interconnection with IBM System/360; (4) the on-line error handling and system testing.

The IBM program uses all available core storage. It is flexible, is tailor-made to each customer's requirements, and runs continuously without customer assistance.

Introduction

The IBM 2750 Voice and Data Switching System extends IBM's information handling activities to include private automatic branch telephone exchanges. The 2750 is first a modern PABX offering the user wide and flexible telephone service. It is also a data collection system using telephones as input or inquiry terminals and paper tape as an output medium. Finally, direct linkage between a 2750 and a customer's IBM System/360 allows applications that combine data processing and line switching. Accompanying papers in this issue describe the capabilities of the system[1] and its switching network.[2]

The IBM 2750 system is program controlled. The program which operates it—the operational program—must reconcile two activities: it must control the switching network of a telephone exchange in real time and must handle message exchanges with the data processing system to which the IBM 2750 is linked.

This is not the first time that a stored program is used to control an electronic telephone exchange. The specific qualities that are expected from this kind of program have already been defined in the journal literature, for instance when the ESS No. 1 stored program was described in *The Bell System Technical Journal*[3] in 1964 or, more recently, the programming of the Périclès-Michelet exchange.[4] Other examples published recently include the development work on the Aristote project[5] and on L. M. Ericsson's AKE 12 telephone system.[6]

The various developments in the past ten years in the area of program-controlled telephone exchanges, however, differ with respect to the proportion of functions assigned to the program. The design of any exchange is the result of an economic balance between the cost of core storage and other hardware equipment and the severe requirements relating to reliability, flexibility and, especially, response time.

The design of the IBM 2750 system is the result of a balance which adequately fits the dual purpose of the system: line switching and data collection for some 700 lines. Had the objective been a bigger system, the balance would have been different and probably less onus would have been put on the program for scanning and distribution functions.

After an introductory survey of the control element containing the program, and of the characteristics of the program, this paper will describe the various functions accomplished by the operational program in the IBM 2750, including network control, data applications, duplex control, error handling, and on-line maintenance.

Organization of the program

• *Controllers*

The control element of the IBM 2751 Switching Unit, the main unit of the IBM 2750, is in duplicate. Each controller includes a network control unit and a supervisor unit, i.e., a built-in binary processor[7] with the following characteristics: storage cycle, 4μs; storage capacity: 32,768 words; 12 interruption levels; 3 index registers; and a 30-instruction directory (Fig. 1).

Instructions may be located in one or two storage words, according to their addressing scheme. The logic words are 16 bits long, but the store contains a 17th bit for parity checking and an 18th for program protection.

The author is located at IBM Centre d'Etudes et Recherches, La Gaude, France.

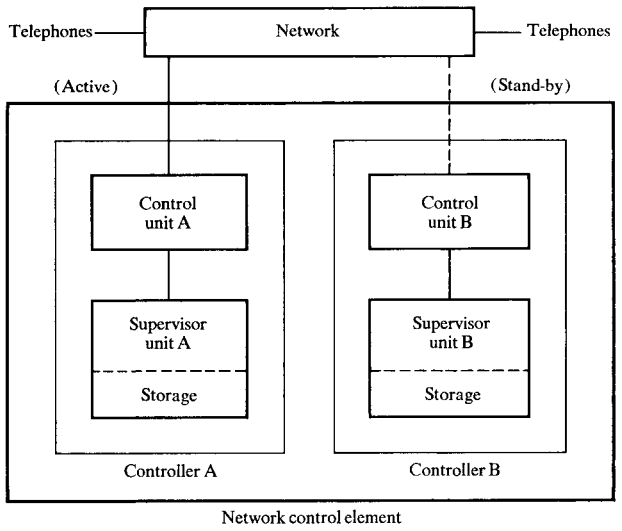


Figure 1 The IBM 2751 Switching Unit.

The programs cannot be modified by mistake, although they are loaded in the same storage as the data and tables concerning the state of the system.

The control units execute network commands sent by the supervisor unit and perform the scanning and elementary marking operations, such as firing the crosspoints of a network path, or modifying the status of a group of eight tone senders or eight operator desk lights.

- *The programs*

The program loaded in each controller is entirely written by IBM and is tailored to each customer's requirements. Each customer may choose a particular system size and feature configuration. In fact, each program in the field is different and generated specially from a single master tape. The program, however, is written in machine language and has been optimized to provide the best possible service compatible with the storage limits.

The operational program consists of instructions in logical sequences, forming routines or programs and data in logic structures, forming tables.

The tables contain two types of information. Some information is required more or less permanently. The permanent tables are loaded with the rest of the program, and can be updated manually through an I/O device; they include control blocks, registers, buffers and queues, used dynamically by the program, and overwritten at each new call; and inhibition tables to identify faulty elements that must not be used any longer.

The same program is loaded in the two controllers and uses all the core available. But the two controllers do not operate synchronously. While one is active, the program in the other is standing by and performs a limited number of supervisory tasks, namely the cyclic tests and the GO AHEAD signal control.

- *Task scheduling: controller load*

The most stringent requirement imposed on the system is to respond quickly, i.e., fast response (0.2 to 1 second) must follow the action of the telephone user. To satisfy this constraint, the operational program has been made entirely core-resident, except for maintenance programs; programs are not called from a drum or disk.

Various priorities have been established in the program. The program makes extensive use of the hardware interruption levels available in the controller, which allow a fast but discriminating reaction to immediately identified events. At the normal level, i.e., when no interruption is in progress, programmed priorities have been added, since not all the tasks need the same response time.

To control the network properly, the operational program must

a) detect incoming signals sent on the lines by external agents and cause changes in the network circuits;
b) carry out the network switching actions called for by these signals.

Most signals are transient and must be detected before they disappear. Hence these circuits must be scanned, not only at regular intervals, but at a frequency such that no change can pass unnoticed. A clock signal every 8.13 ms has been chosen (interruption level 1).

Scanning is done by a set of synchronous programs, running at the interruption level 6; they depend on, but have a lower priority than, the clock signal, so that they may be interrupted by the status change interruptions (level 4), the end-of-scan interruption (level 5) and the following clock signal (level 1), which protects the synchronous program against an abnormal overrun.

Once incoming signals have been detected, their analysis to determine the switching actions required, and the execution of these actions, need not be immediate providing they are carried out within a reasonable time.

These tasks are therefore accomplished by network control programs, at the normal processing level, that run in the time left over by the synchronous programs. The distribution of tones, ringing voltages and pulses, being periodic, is also controlled by the synchronous programs once the network control programs have decided that a tone, a pulse or a ringing voltage must be sent.

The network control programs are interrupted at each clock signal to let the synchronous programs run and are then resumed, as indicated in Fig. 2. In addition to the levels used by the synchronous programs, other interruption levels are used for duplex control and error handling routines, and for the service of standard I/O

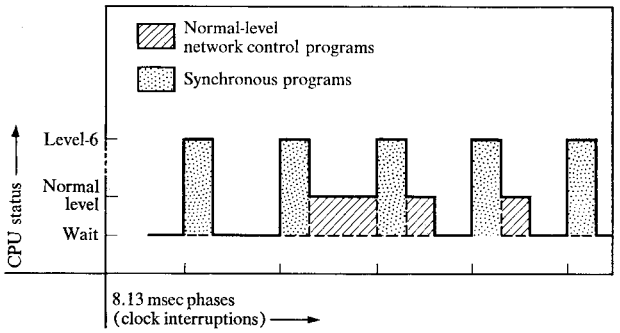


Figure 2 Sharing the controller time.

devices. At the normal level, the network control programs must share the controller time with other programs, for duplex control, error handling or on-line maintenance.

These programs are all given different priorities and are called by an asynchronous scheduler program.

Every normal level program is designed not to run more than 400 ms and ends by returning to the asynchronous scheduler. The scheduler examines a control word and calls the program of highest priority whose run has been requested or scheduled. In this manner, some twelve levels of program-controlled priorities come below the hardware interruption levels. However, the scheduler cannot interrupt a running program. If no program is ready to be called in the control word, the scheduler waits. At the next clock interruption, the controller restarts and the control word will be examined again.

Tables 1 and 2 indicate the assignment of the interruption levels and of the program-controlled priorities at the normal level. Figure 3 summarizes the program organization under normal operation. The synchronous programs (scanning, distribution) and various permanent tasks take up about 45% of the controller time. In addition to this fixed load, there is a traffic-dependent variable load. To ensure that the traffic expected through a given IBM 2750 system does not exceed its capacity, the average load of every type of telephone call has been evaluated and the controller load can be predicted. In normal cases, the "voice" calls leave room for the data applications.

Synchronous programs

The synchronous programs scan the system for changes in the terminal circuits (incoming signaling), distribute the periodic outgoing signaling, and provide timing bases for other programs. These programs must be fast and accurate. The order in which the synchronous programs are executed and the frequency at which they are called are governed by a program called the *synchronous task scheduler*, activated every 8.13 ms by a clock interruption. Since the frequency at which tasks, or programs, need to be performed varies, the task scheduler is divided into

Table 1 Interruption level assignment.

0	Error in supervisor unit
1	Clock signal (8.13 ms)
2	Switchover control
3	Error on network command
4	Status change (scanning)
5	End-of-scan, end-of-distribution
6	Synchronous programs
7	BSCA operations
8	Paper tape operations
9	Keyboard/printer operations
10	Debugging I/O units
11	Customer engineer console key

Table 2 Assignment of priorities for the normal-level programs under the control of asynchronous scheduler.

0	100 ms timer (top priority)
1	GO AHEAD signal control
2	Functional tests : inhibition of network elements
3	Recovery program
4	Keyboard message analyzer
5	On line loading of maintenance programs
6	Traffic analysis
7	Network control (event assembler, event processor)
8	Data input-output program
9	Maintenance programs
10	Cyclic tests
11	Dis-inhibition of network elements at IBM Customer Engineer's request

phases and cycles; each clock interruption initiates a phase, and twelve phases comprise a cycle (97.6 ms). At each phase, the tasks scheduled for that phase are executed in the order in which they appear in a special table, for instance out-pulsing, extension scanning and tone distribution. Each task has a duration that depends on the traffic. Therefore, the tasks called first are executed with the best possible accuracy.

- *Scanning*

The scan programs issue network commands that use tables containing the list of items to be scanned and their previous status. When the control unit detects a change in status, it causes an interruption. The changes are placed in a scan queue, the status is updated in the tables, after which the scan is resumed from where it was interrupted. Extensions already engaged in a call are scanned every 16.3 ms, i.e., at every other clock interruption, and confirmed 8.13 ms later, as a protection against surges. Idle lines are scanned every 100 ms, the public network lines are scanned every 100 ms, and multifrequency receivers and operator desks every 16 ms.

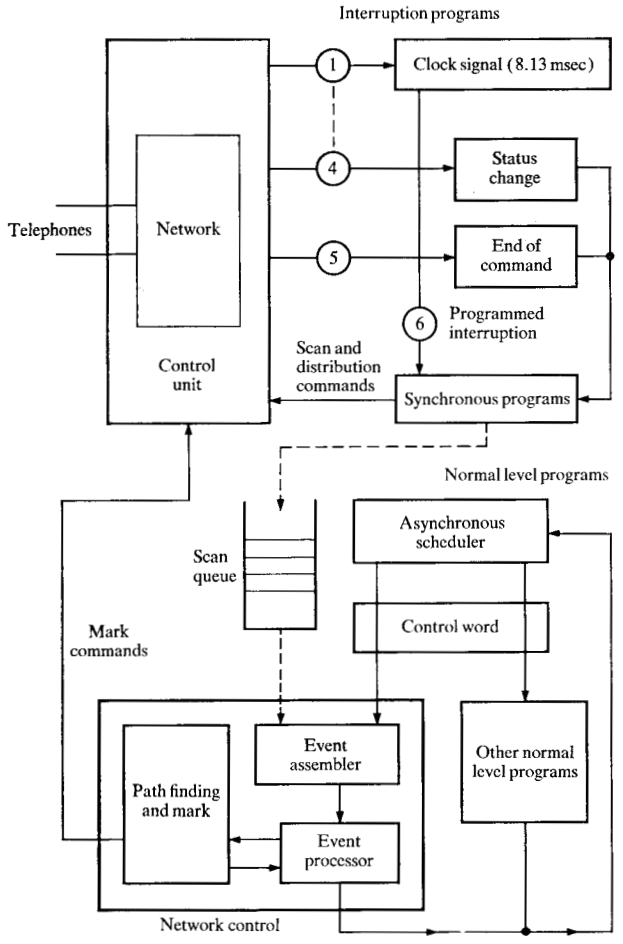


Figure 3 Network control programs.

• *Distribution*

The various outgoing signals to be distributed are:

1) Ringing pulses to extension lines
2) Tone pulses to extension or public network lines
3) Lamp control pulses to the operator control desks
4) Outgoing pulses on public network lines.

The distribution programs use tables containing lists of terminals to which the pulses must be sent. The contents of the distribution tables are updated by the network control programs.

Outgoing pulses on public network lines must be separately generated by a synchronous program because 1) the signals produced by the operator dialing keys and push-button extension sets cannot operate the selector circuits of the public exchanges, and 2) although rotary sets produce the right pulses, these cannot be allowed to go through the network since they would unmark established paths.

All the digits to be sent out on public network lines are therefore detected first by the scan programs. They are then pulsed out by the outpulsing program, which is called every 8.13 ms. Two line loops are opened and two closed at each phase to spread the load on the public exchanges.

• *Scan queue*

The scan queue is the interface between the synchronous and the network control programs: the changes in status discovered by the synchronous programs are stored in it until such time as the network control programs can process them. The entries in the queue are created, one at a time, by the various status-change interruption processing routines and read one at a time by a network control program called the *event assembler*.

The scan queue has 50 entries. Each entry contains, in more or less unchanged form, information collected during a scan, so that the interruption processing routines can be kept short and the scans held up only briefly (0.4 ms on the average per interruption).

This arrangement allows a large number of status changes to be handled in a given 8.13 ms phase, the remaining processing load being dealt with by the network control programs. In this manner, higher peaks may be accepted without disturbing the scanning and distribution timing. The synchronous programs place a timing mark in the scan queue every 100 ms to separate status changes that have occurred in one cycle from those occurring in the next one. Such a single scan queue, divided by timing marks, is a powerful method of storing the exact and relative chronology of all the status changes, and of processing them correctly in their right order, even if this processing must be delayed for a few hundred milliseconds in cases of synchronous peak load.

Network control programs

The asynchronous network control programs are a set of normal level programs that carry out the network switching actions called for by the status changes discovered by the synchronous scan programs. These programs are therefore called whenever there are any entries in the scan queue. Their design is based entirely on a simple concept, the event, which is defined as an occurence that requires a switching action in the network.

An event assembler examines the contents of the scan queue and with this information prepares events. It works on tables called *dial registers*. One after the other, the events are passed by the event assembler to an *event processor*. The event processor is divided into different pages, addressed by *decoders*. Each page processes an event in a particular context. It operates on tables called *transaction blocks*, and when necessary calls path finding

Table 3 Event processor decoders.

1	Digit dialed on rotary telephone set
2	Take-off
3	Hang up
4	Flashing
5	Digit dialed on multifrequency telephone set
6	Timer run out
7	Tone on public network line
8	Ringing current on public network line
9	Metering pulse on public network line
10	Operator desk key

party-conference calls, direct access to an integrated paging system from every extension, abbreviated dialing, and external number repetition, as described in Ref. 1.

The main requirement for the event processor is flexibility: it embodies the telephone procedures, which vary from one system to another, and memory saving, which accounts for the major part of the core storage load.

These objectives have been achieved by building around the "event" concept two programming techniques: the *page decoder* and the *page interpreter*. The first provides adaptability to any procedure; the second, making use of one-word statements for subroutine linkage, reduces storage requirements.

and marking programs to carry out the switching actions required by the event, as indicated in Fig. 3.

• *Event assembler and dial registers*

The event assembler is in charge of converting "raw" status changes, as found in the scan queue, into typical events, with which the event processor is able to deal.

Some status changes in the scan queue may be defined immediately and unambiguously as an event, e.g., a ringing signal on a public network line, a digit emitted by a push-button telephone set and received on a multifrequency receiver, and pressing or releasing an operator key.

Other events, which concern extensions only, require more than one status change before they can be correctly identified. The opening of an extension line-loop (break) for instance, can be caused by hanging up, by flashing or by part of a digit dialed on a rotary set. Such status changes must therefore be accumulated and evaluated against a time scale in order to be assembled into significant events. This is done in the dial registers.

As soon as an event has been processed, the event assembler takes a new entry in the scan queue and examines it. If it appears that an idle extension circuit has been closed, i.e., that the handset has been lifted off its hook, a dial register is assigned to that line.

A dial register is a temporary work area in which data are collected while the extension user is dialing; it is used to keep a count of the pulses making up each digit and the value of each digit making up the number required. In addition, it contains a timer (which is updated by means of the 100-ms timing marks in the scan queue) and a status indicator.

• *The event processor*

The event processor is designed primarily to support the standard facilities offered to customers in conventional telephony but optional "voice" features may be added easily to the program at generation time, such as three-

• *Page decoders*

During the static period of a call, when no processing is required, various parameters are gathered for each line in a *transaction block*, a table that is assigned to a line as soon as it becomes active, and is updated at each event. Transaction blocks contain such information as the network path and service circuit attached to the line; the number of the operator dealing with the call, if any; a procedures timer; and a dynamic *line status*.

Each time the event assembler has recognized an event, it calls the event processor to take the appropriate action. This action depends on the nature of the event and the previous line status, and consists of a limited number of things to do, arranged in many possible combinations:

1) Marking or resetting a path in the network between two terminals (extensions, public network lines, service circuits).
2) Preparing the data for tone, ring, lamp or public exchange dialing pulses to be distributed by the synchronous programs or stopping the distribution of these pulses.
3) Setting a timer to limit the duration of a procedure, or stopping it.

Each combination is an event processing *page*. A page is completely executed before a new event is accepted. But the event processor must be able, one event after another, to process simultaneously several calls in progress.

Therefore the events (e.g., a digit for one call, a timeout for another) arrive in random order to the event processor, which needs, for each type of event, a number of pages equal to the number of *line* statuses for which the occurrence of this event is possible. These pages are classified and addressed through a page decoder, one per type of event. The take-off decoder, for instance, indicates a page for each of the following statuses: *idle*, *ringing* (called by an operator), and *ringing* (called by an extension). Table 3 lists the decoders used by a basic program.

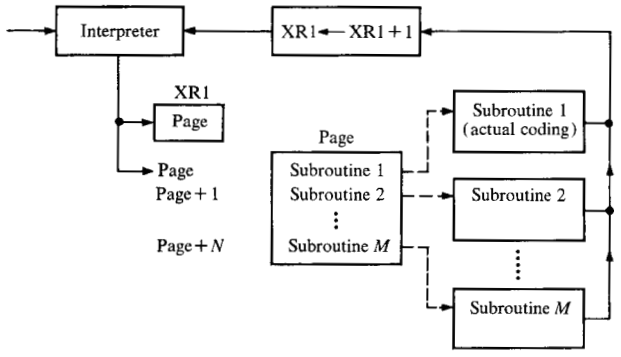
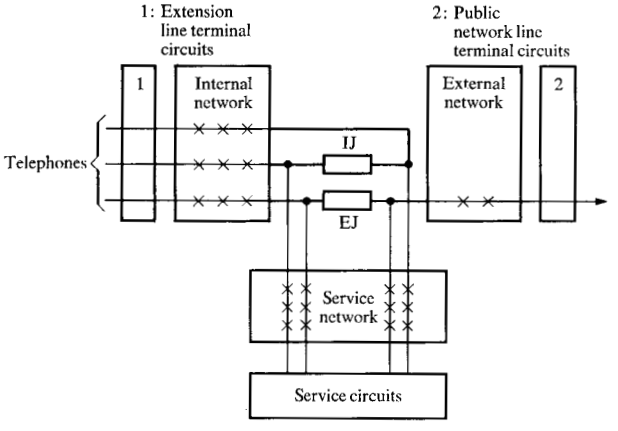


Figure 4 Page interpreter.

Figure 5 The switching network.



The decoders list all the existing pages: they form a kind of tabulated counterpart of the experience of a manual switchboard operator, "What to do in a given situation"? They are the key to the event processor flexibility. Every feature may be added separately to the basic program and requires special pages (also some pages are modified when a feature is required). A conditional assembly in the program generation process retains only the pages which are necessary for the system being generated. No other page needs to be loaded in that system. The decoders are finally built to take into account the existing pages only. Each program is completely adapted to its purpose at minimum storage cost.

•. *Page interpreter*

If the pages were coded in machine language, they would be mostly sequences of instructions to call subroutines. To avoid this, a page interpreter has been developed using an index register as a statement counter. The pages have been designed as source programs, and have been written as lists of defined constants using the subroutine names, which are considered as statements.

The index register 1 (XR 1) is used as a statement counter. To start with, it is loaded with the address of a page by the page decoder. The first word of the page is the address of the first subroutine (see Fig. 4). The interpreter causes an indirect branch to the address in XR 1, so control is given to the first subroutine. The subroutine ends with an instruction that returns control to the interpreter. The latter increments XR 1 and branches to the next subroutine. The process is repeated until all the subroutines called by the page have been executed.

The interpreter provides other facilities, such as branching from a page to another page, mixing interpretive and machine modes, and providing a subroutine with a constant. An interpretive approach minimizes the number of storage words used to call a subroutine: the sequence is reduced to one word. In addition, all subroutines work on the same common table, called the communication area. The address of the communication area is loaded, as a base in the index register 2 (XR2) and the various items of the area are addressed by a short displacement. This reduces most of the subroutine operations to one-word instructions.

•. *Finding and marking paths*

To connect two terminals, the event processor must first find a free path through the switching network, after which it can establish the electrical connection between them.

The switching network (Fig. 5) is divided into three areas: the internal network, the external network, and the service network. These networks can be interconnected by 1) the internal junctors, which close a path between two extensions through the internal network, or 2) the external junctors, which close a path between a public network line and an extension through the external and internal networks.

The service circuits are attached to the service network, and both sides of each internal or external junctor are connected to the service network. Each of the three networks consists of two or three levels of crosspoints arranged in matrices; the outputs of one level of matrices are wired to the inputs of the next level and these connections are referred to as *links*.

To find a free path through the network between two terminals, the event processor calls on a pathfinding program. This program is based on the following principles:

1) There is only one possible path through the networks between a given terminal and a given junctor, although each link is common to several paths.
2) The terminals at both ends of a path are fixed.

3) The network elements must be used in rotation to allow them to be tested dynamically and to spread the traffic over the whole of the network and not concentrate it in one part. Furthermore, this will reduce the likelihood of blocking and speed up pathfinding.

To select a path, the pathfinding program needs to record and know the status of all the path elements in the network, especially the links, the status of which cannot be tested by hardware means. The information concerning these elements is stored in *occupancy tables* which show whether an element is free or busy and which are updated every time a path is reserved or cancelled.

The pathfinding program has been designed so that the more the network and the controller are congested by a traffic peak, the faster this program runs. A straightforward approach leads usually to the reverse effect, which increases the congestion.

Once a path has been found and reserved, it can be marked, that is, established electrically by a pathmarking program. To mark a path, the program first tests that both ends of the path are effectively free; it then marks the path, after which it tests that both ends of the path are busy. Both the tests and the marking are executed through network commands.

When a path is no longer required, it is reset by a path-unmarking program. The occupancy table entries concerning the elements used for the path are then updated by a pathcancelling program to show that the elements are now free.

• *Example of an internal call*

To illustrate the operations accomplished by the network control program, the procedure used to deal with an internal call, typical of all calls, will be summarized. Let us assume that both the extensions that will be engaged in the call, X and Y, are idle. When X picks up his handset, the scan programs detect a change in line status and place it in the scan queue.

A few milliseconds later, the event assembler assigns a dial register to line X, recognizes the status change as a take-off and passes this event to the event processor. The event processor checks the previous status of X and since it was idle, asks the take-off decoder for the idle page. This page assigns a transaction block to the line and attempts to give it the dial tone. To do this, the event processor page must find a free tone sender and a free path in the network, including a junctor, and finally the page must mark that path. Pathfinding and marking programs are successively called.

The page then notes that the status of the extension is now "dialing" and starts a timer in the transaction block to cut the connection if X does not start dialing within 10 seconds.

As soon as X starts dialing, the event assembler cuts the dial tone and assembles the line status changes into digits. When the last digit forming the number of Y has been received and assembled, the event assembler passes the information on to the event processor. The latter asks the rotary digit decoder for the page internal dialing. That page checks that the status of Y is "idle" and looks for a path through the internal network and a free junctor to connect the two extensions.

Once the path is found, the event processor must ring Y and give the ring-back tone to X. To do this, it puts the appropriate data in one of the ring tables and the corresponding ring-back tone table, so that the synchronous distribution programs can distribute these signals at the correct intervals. At the same time, it sets the timer in the transaction block to 1 minute, to limit the ringing time if Y should not reply, and updates the statuses of X and Y to "receiving ring-back" and "receiving ring," respectively.

When Y picks up his handset, the event assembler again detects a take-off. However, since in this case the previous status of Y was "receiving ring," the event processor does not give him the dial tone, but removes the data from the distribution tables to stop the ring and ring-back signals, sets the middle of the junctor to connect both parties, and updates both statuses to "talking".

When one of the parties hangs up, say Y, the event processor frees both half-paths and the junctor, and gives the dissuasion tone to X. Should X not hang up within ten seconds, the dissuasion tone is disconnected and the line is placed in "fault" status. If X hangs up, the event processor sets the status of X to "idle".

Figure 6 illustrates the procedure just described and also shows what happens if X hangs up before dialing or before Y answers and what happens if the transaction block timer runs out while a tone is being sent.

Data applications

There are two types of data features. The first type provides connections between data terminals (autoconnection and teleconnection[1]) and senses or activates a line circuit (contact monitoring). From a control point of view, it resembles the voice features. Additional pages in the event processor suffice to support this type of data feature.

The other type of data feature consists in storing and buffering data collection messages entered on pushbutton telephone sets or multifrequency terminals. To enter a message on a pushbutton telephone set, for instance, a user lifts his receiver and keys a special code. Up to this point, the call is handled by the event processor as a regular call. When the code is identified, the multifrequency receiver digit decoder routes the next characters to specific pages, which accumulate messages in 32-charac-

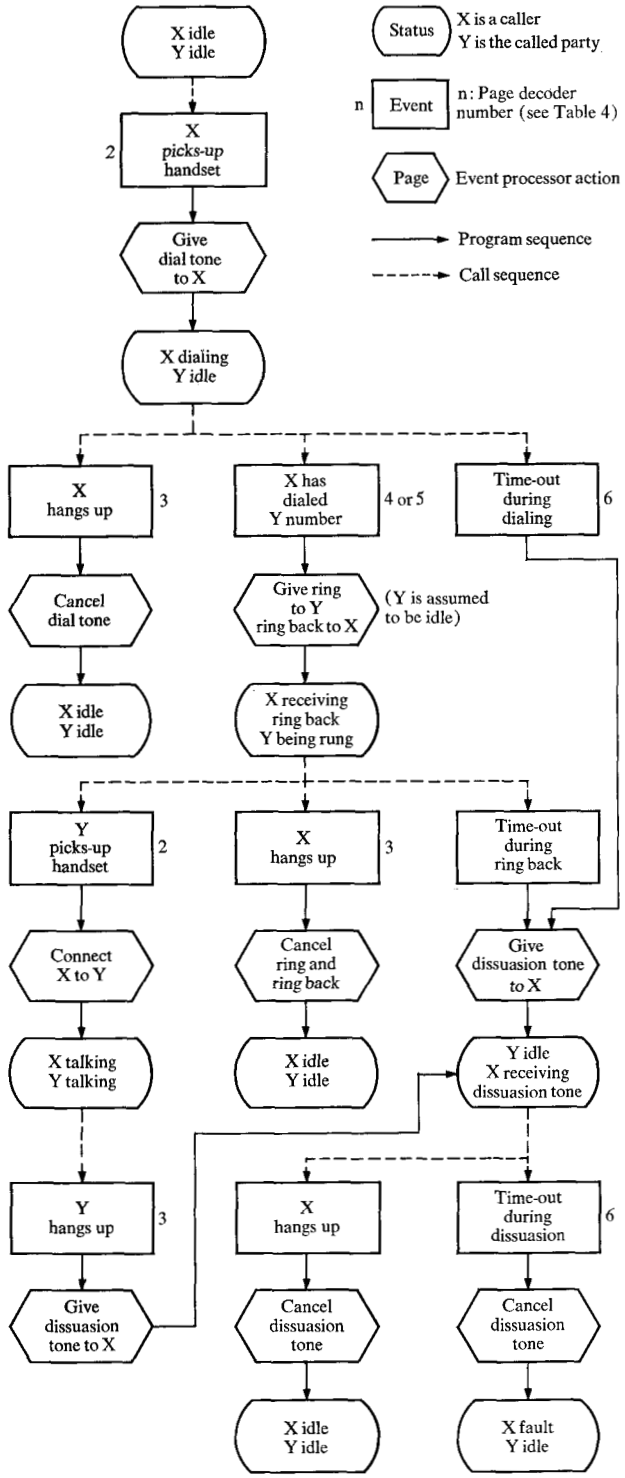


Figure 6 Simplified procedure for an internal call.

ter buffers. Once a message is completed, the event processor checks its length, converts the character code and adds such information as data, time and originating line identification. The message is then punched on a paper tape or sent to a data processing system, immediately after it is received. A tone signal informs the user of the result of the punch or send operation.

Between the 2750 System and a System/360, the data exchanges follow the standard IBM Binary Synchronous Communication (BSC) procedure. BSC link permits the unique combination of voice and data processing facilities, which is the main novelty of the 2750 link. The link not only permits data collection and outgoing call recording messages to be sent to the System/360, but also allows contact monitoring and auto-connection orders to be received from System/360. These orders are handled like events detected by scanning telephone lines; however, acknowledgment messages are returned to System/360 instead of tone signals.

The BSC adapter causes level-7 interruptions in the 2750 operational program. A typical message to System/360 is transmitted as follows:

1) The 2750 sends an inquiry, by issuing an I/O command: this initializes transmission.
2) System/360 is interrupted and sends an acknowledgment.
3) The 2750 is interrupted and sends the message text.
4) System/360 sends an acknowledgment.
5) The 2750 is interrupted and sends an END OF TEXT signal.
6) The BSC line is then idle.

The operational program has to process these interruptions and, at the normal level, has to perform initialization and termination tasks for the messages sent to System/360, and termination tasks for the messages received from System/360 (for the latter, the initialization is under System/360 control). For instance, termination consists in freeing a buffer and sending a tone to the user. A queue of outgoing messages and a queue of incoming messages are dedicated to BSC transmission.

Punching a message or sending it over the data link is a type of job that the event processor cannot handle directly. It is assisted by a *data input-output program*, which has the next priority after the network control program at the normal level. Every 100 ms, this program is called by the asynchronous scheduler. It processes the queues and, according to their contents, performs one or more of the tasks with the following order of priority: termination, initialization of an outgoing BSC message, termination of an incoming BSC message, termination, initialization of a paper tape message. The data input/output program performs appropriate recovery routines whenever transmission failures or hardware faults have been detected.

Duplex control

At this stage, it is necessary to summarize the other functions which the program is expected to monitor: the

duplexing of the two controllers, the tests that check that the system is operating correctly, and the on-line maintenance.

The electrical switchover from one controller to the other is carried out by hardware logic, but the changes in the operational program in the two controllers is governed by a duplex control program.

Each controller sends a pulse to the switchover logic every second to indicate that it is operating correctly. This pulse, called the GO-AHEAD signal, is produced by one of the synchronous programs, which must first test a word set by one of the normal level programs; thus the signal indicates that both the synchronous and the normal level programs are running correctly.

The test programs may request a switchover by causing a burst of three-consecutive GO-AHEAD signals.

If the GO-AHEAD signal from the active controller does not occur regularly once every second, the switchover logic executes a switchover, unless the standby controller is not in a state to take over control of the network.

Whenever a switchover occurs, the switchover logic forces an interruption level-2 in both controllers, which calls a switchover control program. This program ensures that the appropriate programs will be run in each controller according to its new state (active or standby) and gives control to a recovery program in the active controller. The recovery program is called whenever the tables in the storage of the active controller do not reflect the state of the network, as the result of either a switchover or a program error in the computer. All established internal and external conversations are maintained and their "image" reconstructed in the program tables.

Error handling

To make sure that the 2750 is working properly, both the switching network and the control element are checked continually by test routines that run in both the active and the standby network controllers. These test routines are divided into functional tests and cyclic test routines.

- *Functional tests*

These routines are entered when an error has occurred either in the execution of a synchronous or marking program (when the controller is active) or in the execution of a cyclic test in the active or stand-by controller. For instance, should either a driver anomaly or a negative answer on test occur during a marking procedure (both resulting in a level-3 interruption), the same network command is repeated. Should the command still not be successful, the functional tests inhibit one or several network elements and print a message, while the event processor tries an alternate procedure, the same as the one used in case of network blocking.

Depending on the type and gravity of the error, the functional tests can also require that the customer engineer be called (a light on the operator's desk) and/or that a switchover be executed.

- *Cyclic test routines*

These routines normally run every three seconds in both the active and the standby controllers when no other more urgent task needs to be performed.

Switching network. Some of the cyclic test routines check the scan detectors, which cannot be tested while they are in use. These test routines are run only in the active controller, since the standby has no access to the scan detectors which are in the switching network. Whenever an error is detected, a message is printed.

Network controllers. The other cyclic test routines check the control element and are run in both controllers. They test the core storage, the operation codes, the interruptions, and certain control unit circuits. Any error detected is printed and a counter records how often it occurs. A switchover is called if the repetition exceeds an acceptable frequency.

Program errors

Successive levels of protection against a system interruption being caused by a program error, i.e., a logical error remaining in the program, have been built into the program design:

1) The program is thoroughly checked out before delivery by means of simulations and very exacting test equipment.
2) No HALT instruction exists in the program, which has a functional test routine ready for all types of detected hardware error.
3) If in some circumstances, an erroneous way of handling a particular type of call leads to an incorrect response because of a program imperfection, its effect will be limited to one call and one person. Once the dissatisfied user hangs up, the trace of the error disappears, since the transaction block which contains the call data is released at the end of the call. If it is frequent, the trouble will be reported by the customer and corrected by IBM, but in no case will it interrupt the system.
4) Under very improbable circumstances, not simulated at the debugging time, a more serious inconsistency in the program logic may result in a completely erroneous sequence. Experience has shown that every time an erroneous program sequence occurs, the program finally attempts to write in a protected area or to execute a data word as an instruction. In the first case, the memory protection has proved to be a powerful safeguard against program errors. All the program instructions are protected, i.e., if the program attempts to store anything in a protected word, a supervisor unit error interruption (level 0), will

immediately occur. In the second case, the program very quickly encounters an invalid operation code, which causes the same type of interruption. In both cases, the recovery program is executed. The operational program then generally restarts on a sound basis.

5) Should the program have to repeat the above process too many times, it finally requests a switchover, and the second controller takes over. The most serious effect of the logical errors described above would be an undesired loop, to which any system is equally vulnerable, whatever its storage medium is (e.g., READ ONLY storage, etc.). Means exist in the network control program to detect most of such loops but in the worst case a switchover occurs through the absence of the GO-AHEAD signal.

It must be emphasized that a program inconsistency in the active controller cannot, *at the same time*, affect the standby one since both controllers operate independently: for instance, while the active one processes an outgoing call, the standby controller may run a cyclic test.

6) If the second controller does not run satisfactorily and also calls for switchover, the system is eventually interrupted. But a final protection has been devised: FALL BACK protects the system against program errors which could remain undiscovered in a version of the program including recently delivered features. In case of system interruption (both controllers down), the customer is allowed to press a FALL-BACK key, which puts the controllers in FALL-BACK mode.

The upper part of storage is abandoned and the program runs only on the lower part. Since recently installed or exotic feature routines are all loaded in the upper part, they are now ignored. In this way, all lines will continue to benefit from a basic telephone service, including outside calls until the customer engineer diagnoses the trouble.

On-line system maintenance

The operational program allows the customer engineer to load programs and certain diagnostic programs in a special "CE area" with an IBM 1134 paper tape reader. These programs are executed under control of the CE in a time-sharing mode with the automatic functions of the operational program.

The *service programs* comprise:

1) A table initialization program, used to initialize the tables containing information supplied by the customer whenever a new operational program is loaded into the system.

2) A table updating program, used to modify or update the permanent tables at the customer's request.

3) I/O utility programs, allowing various sorts of dumps, either on paper tape or on an IBM 1816 printer, as well as copying paper tapes and checking them for validity.

4) A traffic analysis program, to measure the traffic load on various parts of the system.

5) The debugging aids, which record events occurring in the system or allow a snapshot dump on the 1816 printer.

Most of the *diagnostic programs* are designed to be executed off-line, i.e., when the complete system is stopped, or in the standby controller, while the active controller operates the switching network normally. However, to confirm a fault in the network detected by the functional tests, to check a repair or to execute preventive maintenance in the network, the customer engineer needs access to the switching network from the active controller. In order not to stop the system under these circumstances, a special diagnostic program, the network verification test, has been designed and can be loaded in the CE area, like the service programs. It runs under the control of the operational program and allows the customer engineer to test on-line the network matrices, the junctors, the multifrequency receivers and the tone senders.

Conclusion

The IBM 2750 operational program is now running satisfactorily, its performance objectives have been met, and its design options have already proved to be sound. Should this program need to be rewritten, only a few changes would need to be made to take advantage of experience from the installation of the first models, and these would all be made to enhance the most significant advantage of controlling the exchange by a stored program, i.e., its flexibility.

Changes in the program are often required at short notice because of national PTT* requirements, special customer requirements, or the redefinition of an operating feature; all these changes must be carefully tested and controlled before being accepted. Telephony is a world of specific detail and local practices, and the objective is to react promptly to any such requirements solely by means of program changes.

Acknowledgments

The author acknowledges the efforts of J. F. Kennedy, C. Beudrion, F. Bohy, S. Huon, and D. Lake, who were responsible for the design and development of the IBM 2750 programs. The program description is based on technical drafts prepared by Ph. Meyer.

References and footnotes

1. B. Corby, "IBM 2750 Voice and Data Switching System: Organization and Functions." *IBM J. Res. Develop.* **13,** 408 (1969), this issue.

* "PTT" is the official designation of telephone administration agencies in Europe.

2. R. Reynier et al, "Electronic Switching Network for the IBM 2750," *IBM J. Res. Develop.* **13,** 416 (1969), this issue.
3. J. A. Harr, E. S. Hoover and R. B. Smith, "Organization of the No. 1 ESS Stored Program." *The Bell System Technical Journal* **43,** No. 5, Part 1, 1923–59 (1964).
4. C. Abraham, G. André, J. C. Mahieux and F. Robert, "The programming of the PERICLES-MICHELET EXCHANGE. The monitor program." *Commutation et Electronique,* pp. 7–15 (January 1969).
5. P. Lucas, A. J. Profit, M. Rouzier et J. Pouliquen, "L'Autocommutateur Électronique de Lannion. Projet Aristote," *Colloque International de Commutation Electronique,* Paris 1966, p. 105.
6. K. Katzeff, A. Svensson, "The stored program in the L. M. Ericsson telephone system AKE 12," *Colloque International de Commutation Electronique,* Paris 1966, p. 195.
7. The basic design of this processor is due to the early work of L. Roy Harper, Brian G. Utley, and their associates.

Received February 3, 1969