SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

TOS/TDOS PROGRAMMING SYSTEM

INFORMATION MANUAL

RCA

TAPE OPERATING SYSTEM (TOS)

TAPE-DISC OPERATING SYSTEM (TDOS)

PROGRAMMING SYSTEM INFORMATION MANUAL

SPECTRA 70

RADIO CORPORATION OF AMERICA · ELECTRONIC DATA PROCESSING

TAPE OPERATING SYSTEM (TOS)

TAPE-DISC OPERATING SYSTEM (TDOS)

PROGRAMMING SYSTEM INFORMATION MANUAL

RADIO CORPORATION OF AMERICA

70-00-503

January 1966

First Printing: January, 1966

CONTENTS

CONTENTS
(Cont'd)

CONTENTS
(Cont'd)

CONTENTS
(Cont'd)

INTRODUCTION

♦ The Spectra 70 Tape Operating System (TOS) is a tape-oriented multiprogramming system designed to facilitate the efficient operation of a wide variety of installation problems. TOS is comprised of three groups of programming components: the Control Systems which have total responsibility for monitoring the processor environment; the Language Systems which provide a number of languages to assist in the preparation of programs; and the Utility Systems which consist of diagnostic routines, peripheral routines, and library maintenance routines that simplify testing and production operations.

The Control System Executive controls the internal environment of the processor. Program loading, memory allocation, interrupt analysis, operator machine communication, and multiprogramming control are functions which are the responsibility of the Executive. The Executive permits up to six independent programs to execute concurrently requiring only that sufficient memory and devices be available for program use. The Control System File Control Processor (FCP) is responsible for controlling all peripheral devices connected to the 70/35-45-55 Processors. Random access capabilities are inherent in the FCP. Varying levels of FCP usage are available to provide programming flexibility. Included are such facilities as label checking, logical data manipulation, and error recovery. The Control System Monitor provides a dynamic program sequence which uses job stream parameters to control the sequential execution of an unlimited number of programs. Operator intervention is minimized in order to optimize program thruput.

The Language Systems provide a complete set of programming languages that provides an installation with a language most suited to its requirements. All languages are integrated and a single program may be composed of elements from all languages. Languages provided consist of COBOL - an English language oriented system; FORTRAN - a scientific language; Report Program Generator designed to facilitate report preparation; and the Assembly System - a machine-oriented pseudo code system.

The Utility Systems provide four basic sets of routines normally required in a given installation. Diagnostic routines provide tools to the programmer which assist him in program testing. The Sort/Merge system provides an efficient sophisticated sort package. Peripheral routines are designed to provide standard functions such as card to tape, random access to tape, etc. Library maintenance routines are provided to permit efficient maintenance of all TOS library systems.

The Spectra 70 Tape/Disc Operating System (TDOS) is a powerful extension of the TOS system. In addition to all TOS functions, TDOS provides a sophisticated communications package as well as providing the capability to execute the Control System, RCA-supplied routines, and installation programs using the 70/564 Disc or 70/565 Drum as the basic storage media. The TOS Multichannel Communication Program provides a system which can be tailored to meet the specific communications requirements of the installation. Use of the 70/564 and 70/565 as a program storage media significantly enhances thruput capabilities. TDOS is designed to significantly enhance capabilities of the TOS user without requiring transitional programming or reassembly/recompilation efforts.

INTRODUCTION (Cont'd)

The TOS and TDOS* are designed to be run on a Spectra 70 processor with at least 65,536 bytes of memory (model E) and with the following minimum peripheral equipment:

1 Console Typewriter, Model 70/97

1 Card Reader, Model 70/237, or Videoscan Document Reader, Model 70/251 with Card Read Feature

1 Printer, Model 70/242, or 70/243

5 Magnetic Tape Devices, Model 70/432, 70/442, or 70/445

(At least two of these magnetic tape devices must be 9-channel, one for the System Tape and one for the Load library. The remainder may be 9-channel or 7-channel with the Pack/Unpack feature. All tapes and random access devices must be connected to selector channels.)

SYSTEM COMPONENTS

◆ Control System:

Executive

File Control Processor

Monitor

Language Processors:

Assembly System

Report Program Generator

FORTRAN

COBOL

Utility Routines:

Sort/Merge

Peripheral Routines:

Card or Paper Tape to Punch and Printer

Card or Paper Tape to Tape

*In the TDOS, a Disc Storage Unit, Model 70/564, or Drum Storage Unit, Model 70/565, is used in place of one tape.

SYSTEM COMPONENTS (Cont'd)

Card or Paper Tape to Punch

Card or Paper Tape to Printer

Tape to Card

Tape Edit

Tape to Printer

Tape to Tape

Random Access Data Transcription Routines

Diagnostic Routines:

Automatic Integrated Debugging System (AIDS)

Console Routines

Memory Print

Self-Loading Memory Print

Tape Compare

Test Data Generator

Trace

Library Maintenance:

Program Section Library Maintenance

Linkage Editor

EXECUTIVE ROUTINE

FUNCTIONAL DESCRIPTION

♦ The Executive routine is responsible for complete control of the computer's internal environment and associated peripheral devices. Control of the internal environment consists of memory allocation, interrupt analysis and control, check-point recording, rerun, control of programs in test status, and multiprogramming control. Associated peripheral device control includes device allocation, channel scheduling of I/O operations, console typewriter control, and error recovery.

Program Initiation

♦ Programs can be initiated by an operator type-in to the Executive by the termination of a previous program or by control of the Monitor routine.

1. *Operator Type-in* - An operator request for program initiation informs the system which program is to be run, which device contains the load library tape, and what priority the program is to receive.

2. *Successor Program Initiation* - On termination, a program can request the Executive to initiate a successor program. The program can specify file names for devices to be passed to the successor program. The priority of the successor program and the location of the load library containing the successor program is assumed to be the same as the calling program.

3. *Monitor Routine* - The Monitor routine can initiate programs requested by control cards in the Monitor job stream which is discussed on page 17.

Executive Components

Executive Lists

♦ The Executive maintains and uses a set of dynamically controlled lists in performing its multiprogramming and device processing functions. These lists and their functions are as follows:

1. *Operation List* - The operation list contains an entry for every program that is eligible for execution, is waiting for an I/O termination, or is waiting for the completion of an Executive operation.

2. *Current Operation Slot* - The current operation slot describes the code being executed and the priority of the program to which the code belongs.

3. *I/O Request List* - The I/O request list contains requests for I/O initiation. The list is subdivided into separate lists for each channel.

4. *Device List* - The device list contains an entry for each device on the computer. This entry contains information about the device and information recorded by the executive relating to initiation and termination of activity on the device.

Executive Elements

♦ Executive elements consist of resident and nonresident control elements. Resident control elements are responsible for the continuous control of the processor during program execution. The nonresident elements are responsible for those control functions that are required infrequently. Nonresident elements are loaded into memory only when required. The various elements and functions are described as follows:

1. *Successor Program Initiator* - On termination, a program can request the Executive to initiate a successor program. The program can specify file names for devices to be passed to the successor program. The priority of the successor program, the location of the load library containing the successor program, and the memory allocated is the same as the calling program.
2. *Locator* - The Locator routine locates the requested program on the load library tape, and loads the program's header block.
3. *Loader* - The Loader routine reads in the program text and modifier blocks, floats the text in accordance with the memory assignments for the program, and enters the program on an Executive operation list.
4. *Overlay Loader* - The Overlay Loader routine locates, loads, and floats a requested program overlay.
5. *De-Allocator* - The De-Allocator routine is entered at program termination. The Free Memory table is adjusted, the program is deleted from the Program table, and all devices that are not passed to a subsequent program are entered in the device list as available for subsequent assignment. The program being terminated has the ability to indicate the next program to be loaded.
6. *I/O Dispatcher* - The I/O Dispatcher is responsible for the initiation of all activities directed to an input/output device except the console typewriter. A check is initially made to determine if the particular channel is available. If the channel is free, the I/O Dispatcher creates the necessary control information and issues the instruction.

   If the channel is busy, the request is queued and is issued after all previously scheduled I/O requests for that channel have been issued. If the channel is inoperable, the operator is informed and appropriate status information is provided to the program. If a second channel has been assigned, both channels are checked before queueing the request.
7. *Device Return* - The Device Return routine is responsible for the processing of I/O interrupts. When control is received by this routine, the channel status is checked to determine the type of interrupt that occurred. A normal I/O interrupt causes control to be transferred to the I/O dispatcher so that a queued I/O command for that channel can be executed. An abnormal interrupt causes channel status information to be stored in the program, and control to be transferred either to the error recovery routines in the Executive and/or back to the program. These interrupts can include manual interrupts, Program Control interrupts, and console request interrupts.

Executive Elements (Cont'd)

8. *Priority Relinquish* - The Priority Relinquish routine selects the highest priority entry on the Operation List, puts it into the Current Operation Slot, and branches to the Control Switcher routine.

9. *Control Switcher* - When control is received from Priority Relinquish routine, the Control Switcher routine determines whether the program specified in the Current Operation Slot was also the last to use the machine state to which control is to be given. If it is, control is given to that state. If not, that state's registers are stored in the last program's Executive Information Area and loaded with the current program's settings. The state's P Counter is then set to the address specified by the Current Operation Slot and control is given to that state.

10. *Contingency Control* - The Contingency Control routine receives control when a program generated interrupt or a test mode interrupt is detected. Information required to service these interrupts is placed in the program's Executive Information Area. The interrupted program's Executive Information Area contains the address of the routine to receive control on this condition.

11. *Interrupt Decoder* - The Interrupt Decoder routine obtains control at every interrupt with the exception of power failure and machine error interrupt. The type of interrupt is analyzed and control is transferred to the appropriate Executive control elements.

12. *Program Terminate* - The Program Terminate routine halts the operation of a given program. A program can be halted based upon an internally generated program instruction, an operator request, or a request from the Executive.

13. *Device Assignment* - The Device Assignment routine is given control before executing the first physical I/O command for each file. The Device Assignment routine requests a device number from the operator and enters the file name in the device list. This device is retained by the program until the program releases it, or until program termination.

14. *Console Control* - The Console Control routine controls the console typewriter. Program requests for use of the typewriter are transferred to console control for execution. A program identification is appended to the message to be typed. If a reply is expected, the program is marked WAITING until the message is received. External requests for type-in's (signalled by the console request button) are also accepted by console control. Console control reads the message, determines the program to which the request is directed, removes the program's WAITING mark, and transfers the message to the appropriate program.

    Requests for nonresident console routines are recognized within this element and are passed to the Locator for loading and execution.

Executive Elements (Cont'd)

15. *Tape Error Recovery* - The Tape Error Recovery routine performs automatic retry for transient magnetic tape errors detected during read or write operations. If the retry is unsuccessful, control is returned to the program for further action or the program is terminated, according to a programmer-selected option.

16. *Operator Type-In Initiator* - An operator request for program initiation informs the system which program is to be run. The operator has the option of specifying which device contains the load library tape, the priority it is to receive, the amount of memory to be allocated, and the location of label information.

17. *Monitor* - Program initiation can also be requested by means of control cards in the Monitor job stream.

18. *Memory Allocator* - The Memory Allocator routine assigns memory to the program and records the allocation in the executive area Free Memory and Program tables.

19. *Restart* - The Restart routine locates the program to be rerun and re-establishes memory, priority and device assignments. The memory assignment is the same as that previously held since float information does not accompany a rerun program. The file names and associated devices are typed out and the operator can change any or all of them.

20. *Nontape Error Recovery* - The Nontape Error Recovery routine performs automatic retry for transient nonmagnetic tape errors detected during read or write operations. If unsuccessful, control is returned to the program for further action or the program is terminated, according to a programmer-selected option.

Input/Output Processing

♦ The Executive transfers data between memory and the I/O devices at the physical device level. This type of input/output processing is called Device Control. To perform the input/output functions, the Device Control macros* listed in table 1 are included within the Executive.

*In addition to these macros, the DTFPH and OPEN macros of the File Control Processor allow the processing of labels automatically when Device Control is used. (See File Control Processor.)

Input/Output Processing *(Cont'd)*

Table 1. Device Control Macros*

Macro Instruction	Function
EXCP	This macro is issued to request the performance of physical I/O operations.
WAIT	This macro is issued to determine if the I/O operations, initiated by an EXCP macro, are completed so that the program can continue.
EXCPW	This macro is issued to request the performance of physical I/O operations. Control is not returned to the program until the requested I/O operation is terminated.
CCB	This macro must appear in the program once for each I/O device. It causes a command control block to be created. This block is required to communicate information to the Supervisor so that a physical I/O instruction can be initiated by means of the EXCP and EXCPW macros.

*In addition to these macros, the DTFPH and OPEN macros of the File Control Processor allow the processing of labels automatically when Device Control is used. (See File Control Processor.)

Multiprogramming

♦ Multiprogramming provides the programmer with the capability of executing a maximum of six programs concurrently. The normal program, in most of its operations, is *suspended* from time to time while awaiting the termination of an I/O instruction. To permit a more effective use of the powerful characteristics of the Spectra 70 computer, the Executive permits the programmer to balance more effectively the use of peripheral devices. This balancing is performed by transferring control from a locked program (one which must wait for I/O termination) to a program that can effectively use the processor functions.

Multiprogramming capabilities are provided so that the programmer need not be concerned, while programming, with the characteristics of other programs that will be executed on a multiprogramming basis. The Executive has established a priority system whereby those programs that must be executed first will obtain control most frequently. Control is transferred to a lower priority program only when the higher priority programs are suspended. A routine is also available that permits the priority of a program to be changed after it has been initiated. The Executive performs the actual transferring of control analysis each time an interrupt occurs. This technique ensures that this analysis is performed without requiring specific transfer of control by the program to the Executive system.

INPUT/OUTPUT DESCRIPTION

Input

♦ The basic input to the Executive is in the form of console typewriter requests.

Output

♦ The basic output of the Executive is in the form of messages to the operator by means of the console typewriter.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor - Model E (65,536)

1 Magnetic Tape Device - Model 70/432, 442 or 445 (9-level)

1 Console Typewriter - Model 70/97 (Includes console)

Optional Equipment

♦ Processors Model F (131,072 bytes), Model G (262,144 bytes), or Model H (524,288 bytes) can be substituted for the Model E Processor.

MEMORY REQUIREMENTS

♦ The memory requirement for the Executive is 16,000 bytes.

RELATED PROGRAMMING COMPONENTS

♦ The Executive is required for operation of all TOS programs. Communication between a program and the Executive is achieved through the use of:

1. An *Executive Information Area* contained in every program.
2. *Supervisor Calls* and specification packages associated with them.
3. I/O Channel Status packages.

A program's Executive Information Area is located at the starting position of the program in memory and must not be overlaid while the program is in memory.

The Executive Information Area contains the following information:

1. Storage for general-purpose and processor-state registers.
2. Storage for floating-point registers.
3. Indicators and register storage for program-generated interrupts.
4. Indicator and register storage for program execution in the test mode.

Programs request Executive services by performing Supervisor calls. Each Supervisor call contains an identifier that specifies the type of executive service required. Supervisor calls refer to a specification package that contains the information the Executive needs to carry out the requested service.

RELATED PROGRAMMING COMPONENTS (Cont'd)

I/O devices return information about I/O activity to the processor. This information is contained in I/O Channel Status packages stored in scratch-pad memory. The Executive delivers channel status packages to the program to which the device is assigned.

TAPE-DISC SYSTEM

♦ The Tape Operating System has the capability to make effective use of the Model 70/564 Disc Storage Unit. A number of components are available.

1. The basic Tape System can store data on disc.

2. The Tape System contains, when requested by the programmer, the capability of executing from the disc. When this option is selected, the Operating System is loaded from tape to disc and remains on disc from that point.

3. The third option available to the programmer is to store and execute programs from disc. When the third option is selected, the Operating System itself must be located on disc. Therefore, the tape-disc system provides the programmer with a technique for rapid access to both the Operating System elements and program overlays. Programs can be permanently stored on disc and initiated from disc by the operator. The tape-disc system contains the same basic elements as are contained in the Tape System. No program modification and/or assembly is required when a program is to be executed from disc rather than from tape.

FILE CONTROL PROCESSOR

FUNCTIONAL DESCRIPTION

♦ The File Control Processor (FCP) is a generalized input/output system that is designed to work with the Executive to control all input/output functions. FCP reduces the amount of the detailed programming required by I/O operations. Input/output operations are controlled on two levels; file control level and device control level.

File Control (Logical FCP) is provided by way of I/O macros included within the Assembler. Under this concept, the programmer always works at the logical record level.

Device Control (Physical FCP) is provided by way of Supervisor Calls included within the assembly which relates to the Executive I/O device macros. Under this concept, the programmer always works at the physical record level.

Logical FCP

♦ Programs make use of logical FCP by issuing certain macro instructions that provide the following functions.

File Definition Macros

1. Describe characteristics of the logical file.
2. Describe the physical device on which the logical file resides.
3. Identify options to be taken under certain predefined conditions.
4. Supply addresses of certain programmer-written routines.

I/O Control Macros

1. Make files available for processing (including label checking).
2. Make logical records available for processing (blocking and de-blocking).
3. Alternate I/O areas (when two I/O areas are used).
4. Store logical records after processing.
5. Perform control operations such as rewind and stacker select.
6. Handle end-of-reel and end-of-file conditions (including label writing).
7. Make files unavailable for processing.

Physical FCP

♦ If the programmer wishes to write his own logical data handling routines, Physical FCP provides macro instructions for the reading and writing of data, processing of standard labels, checkpoint records, nondata operations (for example, rewind, stacker select) and error recovery processing. At this level, the programmer is responsible for all record blocking and de-blocking, data movement to and from work areas and reel swapping.

FCP STANDARDS

General

♦ *Tape files* that are processed by FCP routines must conform to published Spectra 70 Systems Standards with respect to *labels*, *file standards*, and *data record formats*.

Label Processing

♦ In general, the FCP routine for Standard labels performs the following functions:

1. *For header labels on an input tape*
   a. Verify proper volume.
   b. Verify proper identification.
   c. Issue appropriate error messages.
2. *For header labels on an output tape*
   a. Analyze old header label to check expiration date.
   b. Write new header label.
   c. Issue appropriate error messages.
3. *For trailer labels on an input tape*
   a. Verify input block count with trailer block count field.
   b. Analyze the label identifier field for EOV or EOF and take appropriate action.
4. *For trailer labels on an output tape*
   a. Write trailer labels with output block count and an EOF identifier, if the end-of-file condition exists; or an EOV identifier, if the end-of-volume condition exists.

Label Creation and Checking

♦ The FCP will create, write and check standard labels. Files with standard, nonstandard or no labels can be specified. However, the user must construct and check nonstandard labels and the user's portion of standard labels if specified.

The DTFSR entry LABNAME specifies label information that is stored in the user's program for FCP label checking and label creation.

File Processing

OPEN Macro

♦ Before using logical FCP, files must be made available for processing by use of the OPEN macro. The OPEN macro verifies that the correct reel is mounted (if standard labels are used) and also positions the file for processing.

The OPEN/CLOSE, end-of-file and end-of-volume routines follow the DTFEN statement. The programmer may issue OPEN or (CLOSE) macros anywhere in the program.

OPEN Macro (Cont'd)

FCP can overlay the coding generated by the OPEN macro when it is no longer needed. To overlay the OPEN routine, the programmer must specify the operand OVLAY in the DTFEN statement.

1. Multireel tape files are not allowed when the OPEN routine is overlaid.

2. For multifile reels only one logical file on that reel can be handled by the program.

CLOSE Macro

♦ The CLOSE macro deactivates a file previously activated by an OPEN macro. CLOSE is normally written after the end-of-file condition is sensed on an input file. For an output file, any records in memory, a tape mark, and the trailer label (if standard labels used) are written. If the program is to be overlaid by the CLOSE routine, the programmer must also specify the operand OVLAY in the DTFEN statement.

Record Processing

♦ The FCP package handles the following types of record format:

1. Fixed-length, Unblocked.
2. Fixed-length, Blocked.
3. Variable-length, Unblocked.
4. Variable-length, Blocked.
5. Undefined, Unblocked.

All of these formats conform to published Spectra 70 Systems Standards.

Processing Overlapped with Input/Output

♦ The program uses the Executive's I/O functions logical I/O routines in order to read and store records in memory. All of these routines are designed to provide for overlapping the physical transfer of data with processing. The amount of overlapping or simultaneity actually achieved is governed by the program through the assignment of I/O areas and work areas.

An I/O area is the area of memory to or from which a block of data is physically transferred. A work area is an area used for processing an individual record from the block of data.

There are several possible combinations of I/O areas and work areas. These are as follows:

1. One I/O area with no work area.
2. One I/O area with a work area.
3. Two I/O areas with no work area.
4. Two I/O areas with a work area.

In addition, certain devices are buffered, increasing the available amount of processing I/O overlap. In order to achieve maximum overlap, alternate I/O areas are required.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to the File Control Processor is in the form of macro instructions that are written in the programmer's source assembly programs. The macro instructions and their functions are briefly described in this section.

The macro instructions are written in a form similar to other coding in the program. The general format for macro instructions is

```
Name Operation   Operand 1, Operand 2,....Operand n
                          or
Name Operation   Operand 1
                 Operand 2
                 '
                 '
                 '
                 Operand n
```

where *Name* is a valid tag, *Operation* is the name of a macro instruction and Operands 1-n are required parameters to be specified. The macros may be classed as *action* macros or as *descriptive* macros depending on whether or not they lie in the instruction path (action) of the program. In general, the action macros are of the position type while the descriptive macros are of the key-word type.

Table 2. Logical FCP Macros

Macro Instruction	Function
DTFSR	File Definition Macro This macro describes the logical file, indicates the type of processing to be used for the file, and specifies memory areas for the file. These macros follow the START instruction in the program and precede the instruction path coding. There are many different DTFSR detail entries that apply to each type of file when logical FCP is used. The operands of this macro are of the key-word type. They may be omitted if not required. The DTFSR operands are: ALTTAPE, BLKSIZE, CHECKPT, CKPTREC, CONTROL, CRDERR, CTLCHR, DEVADDR, DEVICE, EOFADDR, ERROPT, FILABL, IOAREA1, IOAREA2, IOREG, LABADDR, PRINTOV, READ, RECFORM, RECSIZE, REWIND, TRANS, TYPEFLE, VARBLD, WORKA, WLRERR, and LABDES.
DTFEN	This macro indicates to the Assembler that all files have been defined (by DTFSR entries).

Input *(Cont'd)*

Table 2. Logical FCP Macros (Cont'd)

Macro Instruction	Function
OPEN	This macro activates each file in the program. It is used for all files processed by logical FCP. With physical FCP it is used only for tape files with standard labels.
LBRET	This macro is used with tape files that contain additional standard or nonstandard labels that the programmer desires to check or write. It must be issued at the end of the label routine.
GET	This macro makes the next consecutive logical record from an input file available for processing in either an input area or specified work area.
PUT	This macro writes, punches, or displays logical records that have been built directly in the output area or in a specified work area.
RELSE	This macro is used only with blocked input records read from tape. It allows the programmer to skip the remaining records in a block and to continue processing with the first record of the next block, delivered to him by his next GET macro.
TRUNC	This macro is used only with blocked output records that will be written on tape. It allows the programmer to write a short block of records.
CNTRL	This macro provides physical nondata operations for magnetic tape units, card readers, punches, and printers. Such functions as rewinding tape, card stacker selection, and line spacing may be specified.
CHKPT	This macro causes checkpoint records to be written to a tape for restart purposes.
CLOSE	This macro deactivates any file previously opened in any input/output unit in the system.
FEOV	This macro is used for either input or output files on tape to force an end-of-volume condition.
PRTOV	This macro specifies the operation on a printer overflow condition.
DTFPH	*File Definition Macro* This macro is used only when physical I/O macro instructions are used in a program. It is necessary to define only tape files with standard labels. No other files require definition. The DTFPH detail entries that apply to a file are DEVADDR, LABADDR, and TYPEFILE.

Output

♦ Output of the FCP is the execution of the input, output, or control operation desired.

EQUIPMENT REQUIREMENTS

♦ The number and type of peripheral devices that FCP can accommodate are optional and depend on the requirements of the programmer.

Minimum Equipment

♦ 1 Processor, Model E (65,536 bytes)

1 Magnetic Tape Device: Model 70/432, 442 or 445 (9 level)

Optional Equipment

♦ A Model F (131,072 bytes), Model G (262,144 bytes) or Model H (524,288 bytes) can be substituted for the Model E Processor.

Paper Tape Reader Punch: Model 70/221

Card Punch: Model 70/234 or 70/236

Card Reader: 70/237

Printer: Model 70/242 or 70/243

Bill Feed Printer: Model 70/248 (Continuous Forms only)

Videoscan Document Reader: Model 70/251 (Demand Feed only)

Magnetic Tape Stations: Model 70/432, 70/442 and 70/445

Disc Storage: Model 70/564

Drum Memory: Model 70/565

Mass Storage: Model 70/568-1

MEMORY REQUIREMENTS

♦ The generated FCP requires approximately 4000 bytes. However, this amount is variable depending on the data and device-type definitions of the file definition macros (DTFSR's).

RELATED PROGRAMMING COMPONENTS

♦ FCP works closely with the Executive program by making explicit calls on certain Supervisor routines. Both logical and physical FCP incorporate these calls within their procedures.

The Assembler must be used to generate FCP into the object program upon recognition of FCP macro instructions in the source program.

ACCURACY CONTROL

♦ The FCP system observes the standard set of Spectra 70 accuracy controls.

TIMING

♦ Timing estimates for execution of the different macro routines are not available at this time.

MONITOR

FUNCTIONAL DESCRIPTION

♦ All program preparation runs, including the language processors, Linkage Editor and library maintenance routines are under the control of the TOS Monitor program. The Monitor, which is initiated by operator type-in, operates under control of the Executive and is directed by programmer supplied control cards. Production program execution may also be performed under Monitor control.

Input to the Monitor is a job stream (on cards or tape) consisting of control cards, source language statements and/or program sections. A single job may contain source program sections written in more than one source language. Processing of the total job stream is called a Monitor *session.* Any number of jobs may be performed within a session. Those programs that are executed under Monitor control are called *subprocessors.*

The input job stream is under total control of the Monitor. Any subprocessor or program running under Monitor control requiring access to the job stream must direct its request to the Monitor which reads the card and returns it to the subprocessor or program. Likewise, system printer output is controlled by the Monitor.

Each job to be performed is described by control cards. The Monitor executes the jobs sequentially, requesting the Executive to load each subprocessor and maintaining control over the execution of each. Control cards directing the subprocessors are also included in the input stream and directed by the Monitor to the subprocessor. When the subprocessor execution terminates Monitor determines within the input stream the next subprocessor to be executed.

As defined previously, program preparation includes language translation, linkage editing and library maintenance of both the Object Module Library and the Load Library. Any logical combination of these functions may be performed within a job.

Language translation may include any combination of the following translators:

1. Assembler
2. FORTRAN
3. COBOL
4. Report Program Generator
5. Sort/Merge Generator

Output from each translator consists of one or more program sections, appropriate listings and error notations.

The Linkage Editor binds the program sections into a loadable program. It selects routines that are required by the input program sections, if any, from an Object Module Library and writes a single program Load Library. If no routines are needed to be bound in from the Object Module Library, this library is not required by the Linkage Editor.

FUNCTIONAL DESCRIPTION (Cont'd)

The functions of the Object Module Library Update routine are to create an Object Module Library and to update it by the addition, deletion, or replacement of one or more translated program sections or sets of program sections onto an Object Module Library.

The functions of the Load Library Update are to create a Load Library and to update it by the addition, deletion or replacement of one or more programs in loadable format onto a new Load Library.

Neither the Program Section Library Update nor the Load Library Update is required for program execution.

In addition to program preparation, the Monitor control cards may request production program execution. The Monitor requests the Executive to load the program from the Load Library. On program termination, control will be returned to the Monitor to continue processing the job stream.

The following are some of the functions available through the use of Monitor control cards.

1. Begin job.
2. End job.
3. End monitor session.
4. Comments.
5. Subprocessor call.
6. Specify input stream from cards.
7. Specify input stream from tape.
8. Execute program.
9. Specify dump parameters.
10. Request operation action.

SAMPLE MONITOR SESSION

♦ Figure 1 shows the operation of a Monitor test session consisting of a source program language translation, Linkage Editor, and program execution in simplified form.

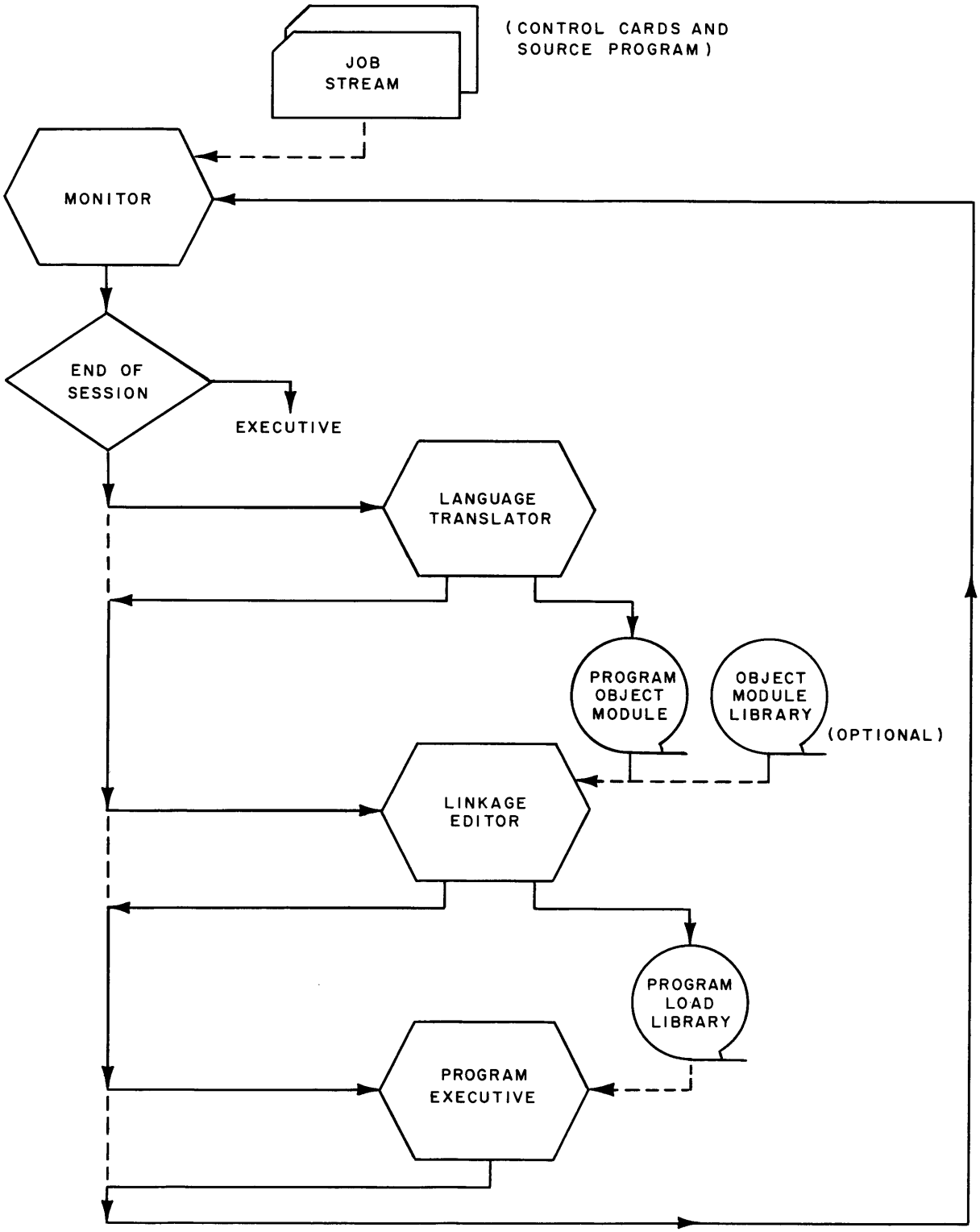

Figure 1. Monitor Test Session

TAPE OPERATING ASSEMBLY SYSTEM

FUNCTIONAL DESCRIPTION

♦ The TOS Assembly System is a machine-oriented, symbolic programming language that expedites the writing of programs for the Spectra 70/35, 70/45, and 70/55 Systems.

Assembler-language programs consist of three types of statements: machine instruction, assembler instruction, and macro instruction.

Machine instruction statements are one-for-one symbolic representations of machine instructions. The Assembler produces an equivalent machine instruction in the object program from each machine instruction in the source program.

Assembler instruction statements provide auxiliary functions that assist the programmer in checking and documenting programs, in controlling storage-address assignment, in program sectioning and linking, in data and storage-field definition, and in controlling the Assembler program itself. Assembler instruction statements specify these auxiliary functions to be performed by the Assembler, and, with a few exceptions, do not result in the generation of any machine language code by the Assembler.

Macro instructions may be used in the source language to generate a series of machine code instructions in the object program from a single source macro statement. Macro instruction statements cause the Assembler to retrieve a unique symbolic routine, to modify the routine according to information supplied within the macro instruction, and to insert the modified routine into the source program for translation into machine language. RCA supplies specially-coded input/output routines as part of the macro library. Macro instructions can be written by the programmer and called from the macro library by use of a programmer-defined macro instruction call line. Furthermore, macro definitions may be included in the source deck for use within the program being assembled.

Mnemonic Operation Codes

♦ Predefined mnemonic codes are provided in the assembly language for all machine instruction, assembler instruction and RCA-supplied I/O macro instruction statements.

All of the operations of the POS Assembly system are contained in the TOS Assembler. Additional mnemonics are contained in the TOS Assembly System to provide for the Channel Command Word (CCW), Conditional No Operation (CNOP), and two additional register-to-register (RR) options for the Branch-on-Condition operation code.

Symbolic Addressing

♦ The Assembly language provides for the symbolic representation of addresses, machine components (such as registers), and actual values required in source statements.

Data Representation

♦ Decimal, binary, hexadecimal, or character representation of machine-language binary values may be used by the programmer in writing source statements. The programmer selects the representation best suited to his purpose.

Program Sectioning and Linkage

♦ The TOS Assembly System provides facilities for generating (optionally) multisectioned programs and for symbolically linking separately assembled programs or program sections.

A control section is a block of coding that can be relocated at load time, independently of other coding, without altering or impairing the operating logic of the control section program. An unsectioned program is treated as a single control section. Dummy sections can also be defined.

The output of the Assembler consists of the assembled control sections and a control dictionary. (See ESD, page 25.) The control dictionary contains information that the Linkage Editor requires to complete cross-referencing between control sections as it combines them into a single object program.

Symbols may be defined in one assembly and referred to in another, thus effecting symbolic linkages between independent assemblies. Specifically, these symbols provide linkages between independent assembled control sections. The Assembler places the required linkage information in the control dictionary (ESD) on the basis of the linkage symbols identified by the ENTRY and EXTRN instructions.

The ENTRY instruction identifies the symbol, within a given section, that is to be used as the name of the entry point from another program. Similarly, the program that uses a symbol defined in some other program must identify it by use of the EXTRN instruction, which will effect linkage to the point of definition.

Base Register Calculation

♦ The base register addressing scheme used in the 70/35, 70/45, and 70/55 requires the designation of a base register (containing a base address value) and a displacement value in specifying a storage location. The Assembly System assumes the clerical burden of calculating storage addresses in these terms for the symbolic address used by the programmer. The programmer retains control of base register usage and the values entered therein by means of the USING and DROP assembler instructions.

Relocatability

♦ The object programs produced by the Assembly System are in a format that enables relocation from the originally assigned storage area to any other suitable area.

Program Listings

♦ A listing of the source program statements and the resulting object program statements may be produced by the Assembly System for each source program it assembles. The programmer can control the form and content of the listing.

Error Indications

♦ As a source program is assembled, it is analyzed for actual or potential errors in the use of the assembly language. Detected errors are indicated in the program listing. Up to six error flags can be printed for each statement.

Fixed-Point Constants

♦ The programmer can define fixed-point constants with appropriate scaling and exponent modifiers. The constants can be aligned on full or half-word boundaries.

Floating-Point Constants

♦ Floating-point constants can be specified with proper scaling and exponential factors appended. This type of constant can also be aligned on full or double-word boundaries.

Source Language Maintenance

♦ Corrections can be applied against a source language library tape for reassembly. Optionally, an updated source library can be produced or an updated source program only.

Macro Definition Language

♦ A macro instruction can represent many machine instructions and/or assembly instructions. The single coded macro instruction is, in turn, transferred into a number of machine and/or assembly instructions, thus reducing detailed coding on the part of the programmer.

A macro instruction can be one of two types: *positional* or *keyword*. A positional macro instruction implies that the values to be substituted must be in a prescribed order. A keyword macro instruction implies that the values to be substituted are paired with keywords defined in the definition of the macro.

Before the programmer can code a macro instruction statement, he must define the series of statements that the macro represents. This is done by a macro definition.

A macro definition is composed of a macro definition header statement, a macro definition trailer statement, a macro instruction prototype statement, and one or more model statements. In addition the programmer can include, in the macro definition, conditional instructions and instructions that define arithmetic values, character values, and logical values. A brief description of each component of a macro definition follows.

Macro Definition

♦ The macro definition header statement indicates the beginning of a macro definition. The macro definition trailer statement indicates that a macro definition is complete.

The macro instruction prototype statement defines the format and the mnemonic operation code of the macro instruction. Because the parameters defined in prototype statements must be general, the entries are referred to as *symbolic parameters.* The difference between a keyword and positional macro is found in the format of the macro instruction. The positional macro instruction prototype statement is generally used in a macro where most, if not all, of the symbolic parameters must be present. The keyword macro instruction prototype statement is generally used in a macro with a large number of symbolic parameters, many of which can be omitted.

The model statements are composed of machine instructions and/or assembly instructions. The Operand fields of the model statements can contain symbols defined in source programs, or symbolic entries incorporated by the macro definition. The symbolic entries are, in turn, replaced by the values they represent. The symbolic entries can be symbolic parameters or variable symbols that are described below.

Macro Definition (Cont'd)

Variable symbols can be equated to arithmetic values, character values, and logical values by using a group of instructions called the SET instructions. A SET variable symbol can be used in the Name field and/or Operand field of any model statement. The value to which a SET variable symbol is equated can be changed at any time by using a SET instruction.

The conditional instructions alter and control the order in which the macro generator processes the statements in a macro definition. The operands of some conditional instructions can contain an arithmetic, character, or logical relation, and a sequence symbol that names the next statement to be processed if the relation is true. It is possible to branch to statements within the macro definition that precede or follow the conditional instruction.

Three system variable symbols, allowing special functions, are provided for the use of the programmer.

A macro definition can contain as a model statement another macro instruction that is generated when it is encountered. The macro instruction within a macro definition is called an *inner macro instruction.*

A macro definition may be placed in the macro library or supplied with the source program. These macro definitions can be referred to by any program to be assembled. Macro definitions supplied with the source deck may only be referred to by the program currently being assembled. The source program definitions must be placed in the deck prior to their first use.

After the macro definition is in the macro library, the programmer can code a macro instruction in an assembly language program. The macro instruction is coded in the same format as the macro definition prototype statement. Substituted in the place of the symbolic parameters of the prototype statement would be the actual values or symbols that stand for actual values.

INPUT-OUTPUT DESCRIPTION

Input

♦ The input to the TOS Assembly System is source program statements consisting of the Statement field, columns 1-71, and the Identification-Sequence field, columns 73-80. The coding of a statement occupies columns 1-71 and, if necessary, columns 16-71 of a single continuation card. The Spectra 70 Assembly Program Form (#28-00-119) is used by the programmer to record the statement field.

Statements consist of one to four entries in the Statement field. From left to right on the form, they are: a Name entry, an Operation entry, and Operand entry, and a Comments entry. Each entry must be separated by one or more blanks and must be written in the order stated. Space is provided on the coding form for an eight-character Name field, a five-character Operation field and a 56-character Operand/Comments field. An optional entry called the Identification/Sequence is also provided. A description of each field follows.

Name Entry

♦ The Name entry (optional) is a symbol used by the programmer to identify a statement. The symbol must consist of eight characters or less. If the Name field is left blank, the assembly assumes no name has been entered.

Symbol definition also involves the assignment of a *length attribute* to the symbol. This is merely the size, in bytes, of the storage field whose address is represented by the symbol. For example, a symbol naming an instruction that occupies four bytes of storage has a length attribute of four. The Assembly System maintains a symbol table in which the values and attributes of symbols are kept. When a symbol is encountered in an operand, the assembly refers to the table for the values associated with the symbol.

Operation Entry

♦ The Operation entry is the mnemonic operation code that specifies the machine operation or assembly function desired. An Operation entry is mandatory and must appear in the first statement line, starting at least one position to the right of the Name field. Valid operation codes consist of five characters or less.

Operand Entry

♦ The Operand entry represents coding that identifies and describes data to be acted upon by the instruction, indicating such things as storage locations, masks, storage area lengths, or types of data.

Depending on the needs of the instruction, one or more operands may be written, with a comma separating each one. Blanks may not intervene between operands and the commas that separate them.

Comments Entry

♦ Comments are descriptive items of information that are to be inserted in the program listing. All valid characters, including blanks, may be used in writing a comment.

A blank must separate the comment from the Operand field and the comment itself cannot extend beyond column 71.

An entire line may be used for a comment by placing an asterisk (*) in column 1. Extensive comments can be written by using a series of lines with an asterisk in column 1 of each statement line or by using continuation lines.

In statements where an optional Operand entry is omitted but a Comments entry is desired, the absence of the Operand entry must be indicated by a comma, preceded and followed by one or more blanks.

Identification/ Sequence Entry

♦ This optional entry may take one of two forms:

1. *Program Identification* - May be used by the programmer to designate his particular ID. The identification is reproduced by the assembly in the program listing.
2. *Sequence Field* - To aid in keeping the source statements in a logical sequence of characters may be included in this field. During assembly, the sequence of the statement cards can be verified by use of the Input Sequence Checking (ISEQ) assembly instruction.

User-Defined Boundaries

♦ If desired, the programmer can disregard the standard boundaries and write the Name, Operation, Operand, and Comment entries in other locations. By use of the Input Control (ICTL) assembly instruction, the programmer may designate his own format.

Output

♦ The normal assembly output consists of the generated object program on cards or magnetic tape (as card images) and a program listing. (See Figure 2.)

Object Program

♦ The object program consists of five types of cards listed as follows:

External Symbol Dictionary Cards (ESD)

Text Cards (TXT)

Relocation Dictionary Cards (RLD)

End Card (END)

Transfer Card (XFR)

A description of each card type is given below:

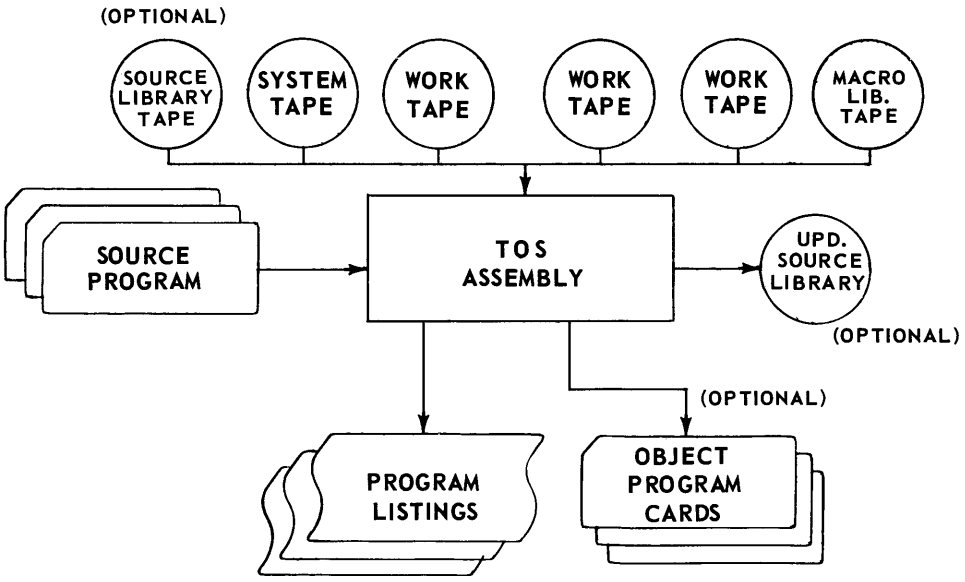

Figure 2. TOS Assembly System, Block Diagram

ESD Card - This card contains all symbol-card storage memory assignments for a program section. ESD cards supply all the necessary information to link together program sections to form an operating program. For instance, the ESD card contains all symbols defined in this section which are referred to by some other section (ENTRY's) and all symbols referred to by this section which are defined in some other section (EXTRN's).

Object Program (Cont'd)

TXT Card - The generated machine instructions to be loaded into storage are contained on TXT cards. The address of the instructions or data and the number of bytes are contained within the card. The TXT cards will be modified as required by RLD information (see below).

RLD Card - The RLD card identifies portions of the TXT card which must be modified due to relocation (i.e. floated). The RLD cards provide the information necessary to perform the relocation and are produced in the same order as the source input.

END Card - The END card is always generated by the assembly and indicates the end of a program section.

XFR Card - The XFR card is only produced by the assembly at the point in the text where specified by the XFR assembly instruction. This card is used by the Program Loader and Linkage Editor routines to define the transfer point or entry point of a phase, or overlay.

Program Listing

♦ The program listing consists of five lists of information: ESD listing, source and object listing, RLD listing, error listing, and symbol table. Options are available to suppress various portions of the printed listings and to control the format of the listings.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ The following minimum equipment is required:

1 Processor:	Models 70/35E, 70/45E or 70/55E
*5 Magnetic Tapes:	Models 70/432, 70/442 or 70/445
1 Card Reader:	Model 70/237
1 Printer:	Models 70/242 or 70/243

Remarks

♦ 1. Additional Memory up to 131K bytes may be used for increased efficiency in processing macro expansions and symbols.

2. The 70/251 Videoscan Document Reader with the card read feature may be substituted for the card reader.

3. Three of the tapes are work tapes and may be 9-channel tapes or 7-channel tapes with the pack/unpack feature. The fourth tape is the System tape and must be a 9-channel tape.

4. If macros are called from the library, either programmer defined macros or RCA supplied macros such as EXCP, a library tape must be used and thus the minimum number of tapes would be five. This last tape must be a 9-channel tape.

5. Magnetic tapes may be substituted for the card reader, and the printer.

---

*This includes the macro library tape. This tape is not required if the macro definitions are included in the source deck. (See 4. above.)

Remarks (Cont'd)

6. A card punch may be substituted for magnetic tape output.
7. One or two additional tapes are required for the source library maintenance option.

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ The assembly operates under control of the Tape Operating System.

ACCURACY CONTROL

♦ The assembler observes all of the standard set of program halts of the Tape Operating System. In addition extensive error checking is performed on the source program.

Table 3. Assembly System Controlling Codes

Name	Mnemonic
Start Program	START
Set Location Counter	ORG
End of Program	END
Define Storage	DS
Define Constant	DC
Identify Entry Point	ENTRY
External Symbol	EXTRN
Identify Control Section	CSECT
Drop Base Address Register	DROP
Identify Dummy Section	DSECT
Start New Page	EJECT
Equate Symbol	EQU
Input Format Control	ICTL
Input Sequence Check	ISEQ
Begin Literal Pool	LTORG
Print Optional Data	PRINT
Reproduce Following Card	REPRO
Space Listing	SPACE
Identify Assembly Output	TITLE
Use Base Address Register	USING
Generate a Transfer Card	XFR
Punch a Card	PUNCH
Define Channel Command Word	CCW
Conditional No Operation	CNOP

ASSEMBLY SYSTEM CONTROLLING CODES

START
Start Program

♦ The START statement must be the first statement in a program. The Name field of the statement names the program and the Operand field sets the Assembly Location counter to its initial value.

ORG
Set Location Counter

♦ The ORG statement allows the programmer to change the Assembly Location counter from its present setting.

END
End of Program

♦ The END statement indicates to the Assembly System that all statements have been processed and is used to indicate the starting address of the object program.

DS
Define Storage

♦ The DS statement allocates storage for such things as input/output areas, temporary and working storage. This statement does not generate instructions but does reserve memory for the area defined.

DC
Define Constant

♦ The DC statement generates data constants within an object program. The constants may consist of any valid EBCDIC character, hexadecimal digit, or memory address.

ENTRY
Identify Entry Point

♦ The ENTRY statement identifies a linkage symbol that is defined in this program but may be used in some other program. The ENTRY and EXTRN statements provide the facility to load and execute as a unit, programs that are independently assembled. The linkage between the programs is provided by defining a symbol in one program and referring to the symbol as an operand in the second program. The location associated with the symbol in the defining program is called the entry point. A reference to an entry point in the other assembled program is called an external reference.

EXTRN
External Symbol

♦ The EXTRN statement refers to an entry point in another Control section.

CSECT
Identify Control Section

♦ The CSECT statement identifies the beginning or the continuation of a control section or segment of a program. All statements following a CSECT are assembled as part of that control section until a statement identifying a different control section is encountered. The first section of a program is preceded by a START statement rather than a CSECT statement.

DROP
Drop Base Register

♦ The DROP statement specifies a previously available register that may no longer be used as a base register. The registers previously used were defined using a USING statement.

DSECT
Identify Dummy Section

♦ The DSECT statement identifies the beginning or resumption of a dummy section. A dummy section represents a control section that is assembled but is not part of the object program. A dummy section is a convenient means of describing the layout of an area of storage without actually reserving the storage.

EJECT
Start New Page

♦ The EJECT statement causes the next line of the assembly listing to appear at the top of a new page.

EQU
Equal Symbol

♦ The EQU statement defines a symbol by assigning to it the attribute of an expression in the Operand field. This statement is the means of equating symbols to register numbers, immediate data, and other arbitrary values.

ICTL
Input Format Control

♦ The ICTL statement allows the programmer to alter the normal format of the source program statement.

ISEQ
Input Sequence Control

♦ The ISEQ statement instructs the assembler to sequence check the input following the ISEQ statement.

LTORG
Begin Literal Pool

♦ The LTORG statement causes all literals thus far encountered in the source program to be assembled at appropriate boundaries starting at the first double word boundary following the LTORG statement.

PRINT
Print Optional Data

♦ The PRINT statement controls the printing of the assembly listing and controls printing of this listing until further print statements are found in the source program.

REPRO
Reproduce Following Card

♦ The REPRO statement causes the assembler to punch into the object deck a duplicate of the statement immediately following the REPRO. This can be used to insert information required by the Linkage Editor into the object program immediately at assembly time.

SPACE
Space Listing

♦ The SPACE statement inserts one or more blank lines in the assembly listing.

TITLE
Identify Assembler Output

♦ The TITLE statement enables the programmer to identify the assembly listing and the assembled output cards.

USING
Use Base Address Register

♦ The USING statement indicates to the Assembly System that one or more general registers are available. This statement also specifies the base address values that the assembly may assume will be in the registers at object time. This statement tells the Assembly System that it may assume that the current value of the Location Counter will be in one general register at object time, and that the current value of the Location Counter will be in another general register at object time.

XFR
Generate A Transfer Instruction

♦ The XFR statement is provided to cause the generation of a transfer card in the assembled text in the same location that the XFR statement appeared in the source program. A transfer card is used by the Loader and Linkage Editor routines to define the transfer or entry points of a control section.

PUNCH
Punch a Card

♦ The PUNCH statement is similar to the REPRO statement except that the Macro Generator can substitute values for symbolic parameters or SET variable symbols in the operand of the PUNCH statement if it appears in a macro definition. This allows the controlled generation of phase names.

CCW
Channel Command Word

♦ The CCW statement provides a method of defining and generating an eight-byte channel command word aligned on a double-word boundary.

CNOP
Conditional No Operation

♦ The CNOP statement enables the programmer to align a statement at a specific halfword. This facility is useful in creating calling sequences consisting of a linkage to a subroutine followed by parameters such as channel command words (CCW).

MACRO DEFINITION LANGUAGE

MACRO
Macro Definition Header

♦ The MACRO macro indicates to the Assembly System that a Macro Definition follows.

MEND
Macro Definition Trailer

♦ The MEND macro indicates to the Assembly System that a Macro Definition is complete.

SETA
Set Arithmetic

♦ The SETA macro assigns an arithmetic value to a SETA variable symbol.

SETB
Set Binary

♦ The SETB macro sets a value true or false to a SETB variable symbol.

SETC
Set Character

♦ The SETC macro assigns a character to a SETC variable symbol.

AIF
Conditional Branch Forward

♦ The AIF macro conditionally alters the sequence in which source statements are processed by the Macro Generator.

AIFB
Conditional Branch Backward

♦ The AIFB macro conditionally alters the sequence in which source statements are processed by the Macro Generator.

AGO
Unconditional Branch

♦ The AGO macro alters the sequence in which source statements are processed by the Macro Generator.

AGOB
Unconditional Branch Backward

♦ The AGOB macro alters the sequence in which source statements are processed by the Macro Generator.

ANOP
No Operation

♦ The ANOP macro facilitates conditional branching to statements that are named by symbolics or set variable symbols.

MEXIT
Macro Definition Exit

♦ The MEXIT macro indicates to the Macro Generator that it terminates processing of the Macro Definition.

REPORT PROGRAM GENERATOR

FUNCTIONAL DESCRIPTION

♦ The TOS Report Program Generator produces an object report program from a problem-oriented source language. Common report features such as input data selection, editing, calculating, summarizing, and control breaks are provided by the generator.

The source program is the input to the Report Program Generator. The language describes to the generator, information concerning the input data format, operations to be performed on the data and the output format of the report. The generator generates an object program required to perform the requested functions.

Source programs written in RPG on the Spectra POS (Primary Operating System) can be compiled without change using this compiler.

Some of the features of the Report Program Generator are as follows:

1. Problem-oriented language designed for ease of use
2. A wide variety of input and output options
3. Up to nine control breaks
4. Write up to eight printed reports
5. Use Table Lookup
6. Checking the sequence of different record types within a control group or checking the sequence of control groups.
7. Exit to a programmer's subroutine written in a language other than RPG.
8. Editing by a mask
9. Random processing of files
10. Obtain data records from single or multiple input files
11. Updated output tapes
12. Perform calculations on data taken from input records or RPG literals.

INPUT/OUTPUT DESCRIPTION

Input

♦ Input to the RPG consists of TOS Job Control statements and an RPG Control Card combined with a source deck. The source consists of the following forms:

1. File Description Specifications
2. File Extension Specifications
3. Input Specifications
4. Calculation Specifications
5. Output Format Specifications

Output

♦ The output of the RPG consists of an object program containing all of the computer instructions and linkage to the control system necessary to prepare the desired report.

A listing of the object program, the source statements, and errors (if present) is produced.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ The following minimum equipment is required:

1 Processor:	Models 70/35E, 70/45E and 70/55E
5 Magnetic Tape Devices:	Models 70/432, 70/442 and 70/445
1 Card Reader:	Model 70/237
1 Printer:	Models 70/242 or 70/243

Remarks

♦ 1. Magnetic tapes may be substituted for the card reader and the printer.

2. A card punch, Model 70/234 and 70/236, may be substituted for one of the five tapes.
3. One of the five tapes is the system tape and must be a 9-channel tape. The other four tapes may be 9- or 7-channel tapes, but if 7-channel tapes are used they must have the pack/unpack feature.
4. The Model 70/251 Videoscan Document Reader with the optional card read feature may be substituted for the card reader.

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ The Report Program Generator operates under control of the Tape Operating System.

ACCURACY CONTROL

♦ The Report Program Generator observe all of the standard set of program halts of the Tape Operating System. In addition all erroneous source-language statements are flagged on the output listing.

TOS REPORT PROGRAM GENERATOR OPERATIONS

Arithmetic Operations

Add (ADD)

♦ This operation adds algebraically the contents of the field or literal in Factor 2 to the contents of the field or literal in Factor 1.

Zero and Add (Z-ADD)

♦ This operation causes the result field to be set to zeros and then causes the data contained in the numeric literal or the field in Factor 2 to be placed in the Result field.

Subtract (SUB)

♦ This operation subtracts algebraically the contents of the field or literal in Factor 2 from the contents of the field or literal in Factor 1.

Zero and Subtract (Z-SUB)

♦ This operation causes the negative of the number contained in the literal or the field in Factor 2 to be placed in the Result field specified after the Result field has been set to zeros.

Multiply (MULT)

♦ This operation causes the contents of the field or literal in Factor 1 to be multiplied algebraically by the contents of the field or literal in Factor 2.

Divide (DIV)

♦ This operation causes the contents of the field or literal in Factor 1 to be divided by the contents of the field or literal in Factor 2.

Move Remainder (MVR)

♦ This operation moves the remainder from a divide operation to a separate field that has been set to zeros by the RPG processor.

Move Operations

Move (MOVE)

♦ This operation causes data characters, starting at the rightmost position, to be moved from the field or literal contained in Factor 2 to the rightmost positions of the Result field.

Move Left (MOVEL)

♦ This operation causes data characters, starting at the leftmost position, to be moved from the field or literal contained in Factor 2 to the leftmost positions of the Result field.

Move High-to-Low Zone (MHLZO)	♦ This operation moves the zone at the leftmost position of Factor 2 to the rightmost position of the Result field.
Move Low-to-High Zone (MLHZO)	♦ This operation moves the zone at the rightmost position of Factor 2 to the leftmost position of the Result field.
Move High-to-High Zone (MHHZO)	♦ This operation moves the zone at the leftmost position of Factor 2 to the leftmost position of the Result field.
Move Low-to-Low Zone (MLLZO)	♦ This operation moves the zone at the rightmost position of Factor 2 to the rightmost position of the Result field.
Testing or Compare Operations	
Compare (COMP)	♦ This operation causes the contents of the field or the literal in Factor 1 to be compared against the contents of the field or literal in Factor 2.
Test Zone (TESTZ)	♦ This operation tests the zone of the leftmost position of the field that is entered in the Result field.
Branching and Exit Operations	
Exit to a Subroutine (EXIT)	♦ This operation enables the programmer to transfer control from the RPG program to a programmer's subroutine.
RPG Label (RLABL)	♦ This operation provides the facility for a subroutine, external to the RPG program, to reference a field in the RPG program.
User's Label (LABEL)	♦ This operation enables the RPG program to refer to a field contained in a programmer's subroutine.
Branching or Go To (GOTO)	♦ This operation enables branching to occur in the object program.
Providing a Label For Go To (TAG)	♦ This operation provides a name to which the program can branch.
Turning Indicators On and Off	
Set Indicators On (SETON)	♦ This operation causes the indicators specified on the calculation specification card to be turned on.
Set Indicators Off (SETOF)	♦ This operation causes the indicators specified on the calculation specification card to be turned off.
Table Operations	
Table Lookup (LOKUP)	♦ This operation causes the field name contained in Factor 1 to be used as the search argument in a table lookup operation.

Conversion Routine Operations	
External Conversion Routines (EXTCV)	♦ This operation indicates the point in the RPG program where the random address conversion routine is to be performed.
Record Key (KEYCV)	♦ This operation establishes the name of the field that is to contain the key of the disc record.

FORTRAN IV COMPILER

FUNCTIONAL DESCRIPTION

♦ The FORTRAN Compiler translates programs written in the FORTRAN source language, a scientific data processing language, into machine language programs.

The FORTRAN source program consists of five basic types of statements that are used to describe the data to be processed and to detail the way in which the data is to be manipulated.

One type of statement calls for input/output operations such as reading of data, printing a result, or punching a result. A second type of statement specifies the arithmetic and logical operations that are to be performed on the data. A third specifies the flow of control through the set of statements or the sequence in which operations are to be performed. A fourth consists of statements that provide certain information about the type and format of data. The fifth type of statement allows one to define FORTRAN subprograms.

Some of the features of the Compiler are as follows:

1. Problem-oriented language
2. Fast compilation speeds
3. Efficient object programs
4. Comprehensive error checking of source input
5. Arithmetic statement functions
6. Subprograms
7. Mode declarations permitting the overriding of the normal mode for a variable or function name
8. Logical expressions and conditional statements to allow decisions to be based directly on the true or false value of a quantity which is logical rather than arithmetic in nature
9. Mathematical relations used to make a comparison
10. Logical operators, variables, and constants
11. Literal constants
12. Adjustable array dimensions in subprograms
13. Variable attribute control: specify data type, length, dimension, and initial values
14. Mixed mode expressions
15. Named I/O list
16. Literal format code
17. Multiple entries into subroutine subprograms
18. Number of subscripts can range from one to seven

INPUT/OUTPUT DESCRIPTION

Input

♦ The primary input to the compiler is the source program. The source program is punched on cards in a standard format. The format is:

Column 1	-	can contain a "C" denoting a comment card.
Columns 1-5	-	contains the statement number.
Column 6	-	indicates a continuation card.
Columns 7-72	-	the FORTRAN statement.
Columns 73-80	-	These columns are not processed by FORTRAN and may contain program identification.

Output

♦ The primary output from the compilation consists of object programs for execution and program listings. If errors are detected, diagnostic messages are listed.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ The following minimum equipment is required:

1 Processor:	Models 70/35E, 70/45E or 70/55E
5 Magnetic Tape Devices:	Models 70/432, 70/442 or 70/445.
1 Card Reader:	Model 70/237
1 Printer:	Models 70/242 or 70/243

Remarks

♦ 1. Magnetic tapes may be substituted for the card reader and the printer.

2. A card punch may be substituted for one of the five tapes.

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ The FORTRAN Compiler operates under control of the Tape Operating System.

ACCURACY CONTROL

♦ The FORTRAN Compiler observes all of the standard program halts of the Tape Operating System.

FORTRAN SOURCE LANGUAGE

Constants

♦ 1. Integer
2. Real
3. Complex
4. Logical
5. Literal

Expression

♦ 1. Arithmetic
2. Logical

Control Statements

♦ 1. Unconditional GO TO
2. Computed GO TO
3. Assigned GO TO
4. ASSIGN
5. Logical IF
6. Arithmetic IF
7. DO
8. CONTINUE
9. PAUSE
10. STOP
11. END

Input/Output Statements

♦ 1. READ (a, b) list
2. WRITE (a, b) list
3. READ b, list
4. PRINT
5. PUNCH
6. FORMAT elements
   a) T, X, P literal, A, I, F, E, D, H, G, L
7. END FILE
8. BACKSPACE
9. REWIND
10. NAMELIST

SPECIFICATION STATEMENTS

- 1. Type REAL, INTEGER, DOUBLE PRECISION, LOGICAL COMPLEX
- 2. DIMENSION
- 3. COMMON
- 4. IMPLICIT
- 5. EQUIVALENCE
- 6. DATA

Subprograms

- 1. Statement functions
  - a) arithmetic
  - b) logical
- 2. FUNCTION
- 3. BLOCK DATA
- 4. SUBROUTINE
- 5. CALL
- 6. ENTRY
- 7. RETURN
- 8. EXTERNAL
- 9. FORTRAN supplied subprograms

COBOL COMPILER

FUNCTIONAL DESCRIPTION

♦ The COBOL Compiler translates programs written in the COBOL source language, a business data processing language, into machine language programs. The COBOL Compiler implements COBOL as defined in detail in this write-up.

The COBOL source language specifies the solution of a business data processing problem. The four elements of this specification are:

1. Program identification of the program.

2. Description of the equipment being used in the processing.

3. Description of the data being processed.

4. Set of procedures which determine how the data is to be processed.

The COBOL source language has a separate division within the source program for each of these elements. The names and descriptions of these divisions are as follows:

Identification Division - The Identification Division identifies the source program and the outputs of a compilation. In addition, the programmer may include the date the program was written, the date it was compiled and any other information desired.

Environment Division - The Environment Division specifies the equipment required. It contains descriptions of the computers for both compiling the source program and for running the object program. Memory size, number of tape units, printers, etc., are among many items that may be mentioned for a particular computer. Problem oriented names may be assigned to a particular equipment.

Data Division - The Data Division uses file and record descriptions to describe the files of data that the object program is to manipulate or create, and the individual logical records that comprise these files. The characteristics or preparation of the data are described in relation to a standard data format rather than an equipment criteria format.

Procedure Division - The Procedure Division specifies the steps that the programmer wishes the computer to follow. These steps are expressed in meaningful English words, statements, sentences and paragraphs. This aspect of the overall system is often referred to as the program but is the only part of the total specification of the problem solution and is insufficient by itself to describe the entire problem. Repeated references must be made to information appearing in other divisions. This division more than any other, allows the programmer to express his thoughts in meaningful English.

FUNCTIONAL DESCRIPTION (Cont'd)

Some of the features of the Compiler are:

1. Chain or (stacked) compilations
2. Comprehensive source language verification. COBOL elements not specified in this section will be recognized and a printout warning to the programmer of these elements will be provided.
3. Compilation speeds are increased when additional memory is available.
4. Copy library facility
5. Facility to bind programs or subprograms written in other languages
6. Comprehensive compiler listings
7. Several types of compilations are:
   a) Compilation without corrections.
   b) Compilations with corrections.
8. A source computer of 65K bytes may produce an object program for memory sizes up to 524K bytes.

INPUT-OUTPUT DESCRIPTION

Input

♦ *Source Program* - The primary input to the compiler consists of source programs on cards or stored on magnetic tape.

Parameter Information - This information designates the type of Compilation, Listings, etc., to the compiler.

Copy Include Library - This tape contains previously created Data Descriptions and Procedure Statements which can be called in by a Source Program and automatically incorporated into the program. With this facility, common Data Layouts and Subroutines can be used thus reducing the total programming effort.

Source Program Corrections - Corrections are accepted from cards or card images on magnetic tape. An indicator on the output listings will designate the source line corrected.

Output

♦ 1. *Loadable Object Program* on magnetic tape.

2. *Detailed Listings* which include the following:
   a. Reference edit listing of the source program
   b. Memory map of the Data Division with cross-reference information to each item of data. Cross references are made to sequence numbers of statements in the Procedure Division.

Output *(Cont'd)*

c. Object program with references to the Procedure Division by sequence number.

d. Diagnostic messages; when present, a listing of all detected source program errors is provided.

3. Updated Copy Library. When changes are applied to the Copy Library, an updated Copy Library is provided.

4. Operator Instructions. Appropriate instructions to the operator are displayed.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ The following minimum equipment is required:

1 Processor:	Models 70/35E, 70/45E or 70/55E
5 Magnetic Tape Devices:	Models 70/432, 70/442 or 70/445
1 Card Reader:	Model 70/237
1 Printer:	Models 70/242 or 70/243
1 Card Punch:	Models 70/234 or 70/236
1 Console Typewriter	

Remarks

♦ 1. Magnetic Tape devices are acceptable substitutes for the card reader, card punch, and the printer.

2. Additional memory results in more efficient processing.

3. When the source program contains copy and/or include statements, an additional tape is required.

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ The compiler operates under control of the tape operating system.

ACCURACY CONTROL

♦ The Compiler performs extensive error checking of the input source program and error message are displayed for incorrect source statements. COBOL elements not implemented by this compiler are flagged for subsequent changing.

COBOL LANGUAGE OUTLINE

Identification Division

♦ PROGRAM-ID. program-name. [any sentence.]

[AUTHOR. any sentence.]

[INSTALLATION. any sentence.]

[DATE-WRITTEN. any sentence.]

[DATE-COMPILED. any sentence.]

[SECURITY. any sentence.]

[REMARKS. any sentence.]

Environment Division

♦ CONFIGURATION SECTION

SOURCE-COMPUTER. RCA-SPECTRA [model-number.]

OBJECT-COMPUTER. RCA-SPECTRA [model-number.]

INPUT-OUTPUT SECTION

FILE-CONTROL

SELECT file-name

ASSIGN TO { DIRECT-ACCESS [device-number UNIT [s]] / UTILITY [device-number UNIT [s]] / UNIT-RECORD device-number UNIT s }

[ACCESS IS { SEQUENTIAL / RANDOM }]

[ORGANIZATION IS { INDEXED / RELATIVE }]

[RESERVE { NO / integer } ALTERNATE AREA [s]]

[SYMBOLIC KEY IS data-name]

[ACTUAL KEY IS data-name]

[SAME RECORD AREA FOR file-name-1 file-name-2 [file-name-3 . . .] .]

[RERUN EVERY { END OF UNIT OF file-name / integer CLOCK-UNITS } .]

[APPLY RESTRICTED SEARCH OF integer TRACKS TO file-name . . .]

[APPLY condition-name TO FORM-OVERFLOW OF file-name.]

[APPLY RECORD PROTECTION TO file-name. . . . .]

[APPLY section-name TO saved-area-name file-name

[FOR integer CYCLES] .]

Data Division

◆ FILE SECTION

FILE DESCRIPTION ENTRIES

RECORD DESCRIPTION ENTRIES

SORT DESCRIPTION ENTRIES

RECORD DESCRIPTION ENTRIES

SAVED AREA ENTRIES

RECORD DESCRIPTION ENTRIES

WORKING-STORAGE SECTION

RECORD DESCRIPTION ENTRIES

LINKAGE SECTION

RECORD DESCRIPTION ENTRIES

REPORT SECTION

REPORT DESCRIPTION ENTRIES

REPORT GROUP DESCRIPTION ENTRIES

REPORT ELEMENT DESCRIPTION ENTRIES

FILE SECTION ENTRIES

FD file-name VALUE OF FILE-ID IS external-name

[BLOCK CONTAINS integer RECORDS]

[RECORD CONTAINS [integer 1 TO] integer-2 CHARACTERS

[WITHOUT COUNT CONTROL]

[LABEL { RECORDS ARE } { OMITTED / data-name }]
[DATA { RECORD IS / RECORDS ARE } { record-name }]

RECORD DESCRIPTION ENTRY

level-number { data-name / FILLER }

[REDEFINES data-name-2]

PICTURE IS { alpha-form / an-form / numeric-form / floating-point form }

[BLANK WHEN ZERO]

[OCCURS integer TIMES [DEPENDING ON data-name]]

Data Division
(Cont'd)

```
[ VALUE IS literal ]

[ JUSTIFIED RIGHT ]

[ SYNCHRONIZED { LEFT  } ]
               { RIGHT }

[ USAGE IS { DISPLAY                } ]
           { COMPUTATIONAL          }
           { COMPUTATIONAL-1        }
           { COMPUTATIONAL-2        }
           { COMPUTATIONAL-3        }
```

Procedure Division

```
♦ OPEN { INPUT  file-name [ WITH NO REWIND ] [REVERSED] }
       { OUTPUT file-name [ WITH NO REWIND ]            }
       { I-O    file-name                               }

       [ INPUT  ] file-name [ WITH NO REWIND ] [REVERSED] ]
       [ OUTPUT ] file-name [ WITH NO REWIND ]            |
       [ I-O    ] file-name                               ]

READ file-name RECORD [ INTO data-name ]

    { AT END         }
    { INVALID KEY    } imperative-statement. . .

WRITE record-name [ FROM data-name-1 ]

    [ INVALID KEY imperative-statement ]

      AFTER ADVANCING { data-name-2 } LINES
                      { integer     }

REWRITE record-name [ FROM data-name ]

    [ INVALID KEY imperative-statement. . . ]

CLOSE { file-name [ UNIT ] [ WITH { NO REWIND } ] } . . .
      {                           { LOCK      }   }

DISPLAY { data-name } . . . [ UPON CONSOLE ]
        { literal   }       [ UPON SYSPCH  ]

ACCEPT data-name [ FROM CONSOLE ]

MOVE { data-name-1 } TO data-name-2. . .
     { literal     }

MOVE CORRESPONDING data-name-1 TO data-name-2. . .

EXAMINE data-name TALLYING { ALL       } 'character-1'
                           { LEADING   }
                           { UNTIL FIRST }

        REPLACING BY 'character-2'
```

Procedure Division
(Cont'd)

```
EXAMINE data-name REPLACING { ALL          } 'character-1'
                            { LEADING      }
                            { UNTIL FIRST  }
                            { FIRST        }

        BY 'character-2'

TRANSFORM data-name-3 CHARACTERS

        FROM { figurative-constant-1 } TO { figurative-constant-2 }
             { non-numeric-literal-1 }    { non-numeric-literal-2 }
             { data-name-1           }    { data-name-2           }

COMPUTE data-name-1 [ ROUNDED ] = { data-name-2           }
                                  { numeric-literal       }
                                  { floating-point-literal }
                                  { arithmetic-expression }

        [ ON SIZE ERROR imperative-statement. . .]

        { numeric-literal       }       { TO    }
ADD     { floating-point-literal } . . . { GIVING } data-name-n
        { data-name-1           }

        [ ROUNDED ] [ ON SIZE ERROR imperative statement. .]

ADD CORRESPONDING data-name-1 TO data-name-2

        [ ROUNDING ] [ ON SIZE ERROR imperative-statement ]

SUBTRACT { data-name-1           }
         { numeric-literal-1     }
         { floating-point-literal-1 }

        FROM { data-name-m [GIVING] data-name-n                    }
             { numeric-literal-m GIVING data-name-n                }
             { floating-point-literal-m GIVING data-name-n         }

     [ ROUNDED ] [ ON SIZE ERROR imperative-statement. . .]

SUBTRACT CORRESPONDING data-name-1 FROM data-name-2

     [ ROUNDING ] [ ON SIZE ERROR imperative-statement. . .]

MULTIPLY { data-name-1           }
         { numeric-literal-1     }
         { floating-point-literal-1 }

       BY { data-name-2 [GIVING] data-name-3                    }
          { numeric-literal-2 GIVING data-name-3                }
          { floating-point-literal-2 GIVING data-name-3         }

     [ ROUNDED ] [ ON SIZE ERROR imperative-statement. . .]

DIVIDE { data-name-1           }
       { numeric-literal-1     }
       { floating-point-literal-1 }
```

Procedure Division
(Cont'd)

```
        INTO {data-name-2 [GIVING] data-name-3                }
             {numeric-literal-2 GIVING data-name-3            }
             {floating-point-literal-2 GIVING data-name-3     }

    [ROUNDED] [ON SIZE ERROR imperative-statement. . .]

STOP {RUN    }
     {literal}

GO TO procedure-name

GO TO procedure-name-1 [procedure-name-2] DEPENDING ON data-name

ALTER {procedure-name-1 TO PROCEED TO procedure-name-2}

PERFORM procedure-name-1 THRU procedure-name-2

PERFORM procedure-name-1 [THRU procedure-name-2]

    {integer          }
    {data-name TIMES  }

PERFORM procedure-name-1 [THRU procedure-name-2] UNTIL test-condition

PERFORM procedure-name-1 [THRU procedure-name-2]

      VARYING data-name-2 FROM {numeric-literal-2}
                               {data-name-2      }

          {numeric-literal-3}
      BY  {data-name-3      } [UNTIL test-condition-1]

    [AFTER data-name-4] FROM {numeric-literal-4}
                             {data-name-5      }

          {numeric-literal-6}
      BY  {data-name-6      } [UNTIL test-condition-2]

    [AFTER data-name-7] FROM {numeric-literal-8}
                             {data-name-8      }

          {numeric-literal-9}
      BY  {data-name-9      } [UNTIL test condition-3]

ENTER LINKAGE

CALL entry-name [USING argument . . . ].

ENTER LINKAGE
CALL entry-name [USING argument . . . ].

ENTER LINKAGE
ENTRY entry-name [USING data-name . . . ].
ENTER COBOL

ENTER LINKAGE
RETURN [VIA entry-name ].
ENTER COBOL
```

Procedure Division *(Cont'd)*

paragraph-name. EXIT.

NOTE comment . . . .

USE FOR CREATING [BEGINNING / ENDING]

LABELS ON OUTPUT [file-name . . .].

USE FOR CHECKING [BEGINNING / ENDING]

LABELS ON INPUT [file-name . . .].

IF condition [THEN] {Statement-1 / NEXT SENTENCE}

[{ELSE / OTHERWISE} {statement-2 / NEXT SENTENCE}]

IF {data-name-1 / arithmetic-expression-1 / figurative-constant-1 / literal-1} IS [NOT] {> / < / = / GREATER THAN / LESS THAN / EQUAL TO}

{data-name-2 / arithmetic-expression-2 / figurative-Constant-2 / literal-2}

IF {data-name / arithmetic-expression} IS [NOT] {POSITIVE / ZERO / NEGATIVE}

IF data-name IS [NOT] {NUMERIC / ALPHABETIC}

IF [NOT] condition-name

Report Section

◆ RD report-name

[CODE "character"]

[{CONTROL IS / CONTROLS ARE} {FINAL / data-name / FINAL data-name}]

PAGE integer-p [LINES][HEADING] integer-h

[FIRST DETAIL integer-d]

[LAST DETAIL integer-e][FOOTING integer-f]

Report Group Description

♦ 01 [data-name]

[LINE { integer-1 / PLUS integer-2 / NEXT PAGE }]

[NEXT GROUP { integer-1 / PLUS integer-2 / NEXT PAGE }]

TYPE { REPORT HEADING / PAGE HEADING / OVERFLOW HEADING / CONTROL HEADING { data-name-1 / FINAL } / DETAIL / CONTROL FOOTING { data-name-2 / FINAL } / OVERFLOW FOOTING / PAGE FOOTING / REPORT FOOTING }

REPORT ELEMENT DESCRIPTION ENTRY

level-number data-name LINE-clause

[COLUMN integer]

[GROUP INDICATE]

[RESET ON { data-name / FINAL }]

{ SOURCE data-name / SUM data-name-1 [UPON data-name-n] / VALUE IS literal }

Report Procedure Division Considerations

♦ INITIATE { ALL / report-name . . . }

GENERATE data-name

TERMINATE { ALL / report-name . . . }

USE BEFORE REPORTING data-name

Sort Description Entry

♦ SD sort-file-name

VALUE OF FILE-ID is external-name

[RECORD CONTAINS [integer-1 TO]

integer-2 CHARACTERS]

Sort Description Entry *(Cont'd)*

```
[DATA {RECORD IS  } record-name . . .]
      {RECORDS ARE}
```

Sort Procedure Division Considerations

```
◆ SORT sort-file-name {DESCENDING} data-name-1 . . .
                      {ASCENDING }

  [ {DESCENDING} data-name-2 . . . ]
  [ {ASCENDING } ]

  {USING                  } section-name-1 }
  {INPUT PROCEDURE        }

  { OUTPUT PROCEDURE section-name-2 }
  { GIVING file-name-2              }

  RELEASE record-name

  RETURN sort-file-name [AT END imperative-statement . . .]
```

Random Processing

```
◆ FILE ENTRIES

      SA saved-area-name

  PROCEDURE DIVISION

      PROCESS section-name

      HOLD [ALL          ]
           [section-name . . .]
```

Copy/Include Program Library Facility

◆ Prewritten source program entries can be included in a COBOL program at compile time. Thus, an installation can utilize standard file descriptions, record descriptions, or procedures without having to repeat programming them. These entries and procedures are contained in a user-created library. They are included in a source program by means of a COPY clause or an INCLUDE statement.

COPY Clause

◆ The COPY clause permits the programmer to include prewritten Data Division entries or Environment Division clauses in his source program. The COPY clause is written in the Data Division in one of the following forms:

```
Option 1
(Within a Saved Area Description entry)
SA saved-area-name COPY library-name FROM LIBRARY
Option 2
(Within a Sort Description entry)
SD file-name COPY library-name [FROM LIBRARY].
Option 3
(Within a Report Description Entry)
RD data-name CODE non-numeric-literal
COPY library-name FROM LIBRARY
Option 4
(With a Report Group Description entry)
01 data-name COPY library-name FROM LIBRARY
```

Library-name is contained in the programmer's library and identifies the entries to be copied. It is an external name and must follow the rules for external-name formation.

SORT/MERGE GENERATOR

FUNCTIONAL DESCRIPTION

♦ A Sort/Merge Generator is provided as part of the Spectra 70 Tape Operating System. This routine enables a programmer to generate a program that sorts files of random records or merges files of sequenced records into one sequential file. From one to twelve control fields may be specified, with the output in ascending or descending order. Control statements are used to tailor the program to the programmer's needs.

The Sort/Merge system has two distinct phases. The *Generation* phase interprets parameters and creates the desired program. The *Object* phase structures memory, and performs the actual sort or merge.

The Object phase, after memory is structured, is subdivided into four parts. The *Internal Sort* reads the input data and, using a replacement selection technique, generates strings onto work tapes. The *External Sort* performs successive reverse-reading polyphase merges, decreasing the number of strings until no tape contains more than one string. The *Final External Sort* merges the data into one sequenced string. The *File Merge* performs a forward-reading merge of input files when the merge function has been specified or when the volume limit has been exceeded.

A summary of the features included in the Sort/Merge are listed below:

1. Up to a 16-way sort or merge is provided.
2. Standard Spectra 70 label processing is provided, as well as provision for programmer checking of non-standard labels.
3. Exits are provided for user own-coding which will permit inserting, deleting, altering records, etc.
4. Checkpoints are taken before each External Sort pass and at the end of each cycle to allow for restarts.
5. Tape alternation may be specified for both input and output files. Certain input-output work tape duplication is also permitted.

INPUT-OUTPUT DESCRIPTION

♦ Input records may be fixed or variable in size, blocked or unblocked. Record formats and labels must adhere to Spectra 70 standards.

The maximum size of the input file is determined by the way of the sort times the number of records that can be written to a reel (based on the internal blocking factor).

EQUIPMENT REQUIREMENTS

♦ 1 Processor (65K bytes)
1 Card Reader
1 Console Typewriter
3 to 17 Tape Devices (7 or 9 track) plus Operating System tapes.

A minimum of one selector channel is required. At least two selector channels are required for maximum efficiency.

MEMORY REQUIREMENTS

♦ The Sort/Merge normally uses all of memory. However, the programmer has the option of specifying the upper limit. A minimum of 12K must be available to the program.

RELATED PROGRAMMING COMPONENTS

♦ The Sort/Merge is part of the Spectra 70 Tape-Oriented System and functions under the control of the Tape Operating System.

ACCURACY CONTROL

♦ The Sort/Merge validates input parameters and observes the standard set of error messages. Control is given to the programmer at a designated exit point if input/output errors cannot be corrected by the program.

TIMING

♦ See Sort/Merge Timing Formula section of the Spectra 70 Sales/System Guide (#99-70-001) for timing information.

CARD OR PAPER TAPE TO PUNCH AND PRINTER

FUNCTIONAL DESCRIPTION

♦ The Card or Paper Tape to Punch and Printer routine produces card output files in EBCDIC format, and printed output in display format, from 80-column card input files or 80-character paper tape blocks of EBCDIC format. The input cards may be field-selected, packed, unpacked, sequence checked, and sequence numbered before being transferred to the card punch and printer.

The display format provides a visual picture of the input and/or output card file in character mode. Each input card starts a new print line with positions reserved to indicate the card number.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of a card file or paper-tape file punched in EBCDIC format.

Output

♦ Output from this routine consists of a punched-card file in EBCDIC format and a printer listing in 80-column character mode.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader or Paper Tape Reader/Punch
1 Card Punch
1 Printer
1 Magnetic Tape Device
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine performs validation of input parameter records and produces a standard set of error messages.

TIMING

♦ (To be supplied)

CARD OR PAPER TAPE TO TAPE

FUNCTIONAL DESCRIPTION

♦ The Card or Paper Tape to Tape routine provides a convenient method of transcribing data from card files or paper tape to magnetic tape. Optional control parameters direct this routine to perform one or more of the following functions:

1. Tape positioning prior to conversion
2. Blocking of output records on tape
3. Sequence checking of input records
4. Editing (field selection and rearranging of fields within the record)
5. Data conversion (packing and unpacking of numeric fields)

INPUT-OUTPUT DESCRIPTION

Input

♦ This routine accepts 80-character card records on cards or paper tape (punched in EBCDIC format) as input.

Output

♦ Output from this routine may consist of a labeled, single-reel, magnetic tape file containing unblocked records. Optional control parameters provide for the production of multireel files, and blocked fixed-length records.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader or Paper Tape Reader/Punch
2 Magnetic Tape Devices
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages. In addition, the routine checks the expiration date on the volume label of the output file prior to transcribing the input file.

When the routine terminates, card count and tape block counts are displayed on the console typewriter. At the programmer's option, the routine may be directed to perform a sequence check on input.

TIMING

♦ (To be supplied)

CARD OR PAPER TAPE TO PUNCH

FUNCTIONAL DESCRIPTION

♦ The Card or Paper Tape to Punch routine provides a facility to reproduce 80-column card or paper-tape files. Via optional control parameters, the routine can also be directed to perform one or more of the following functions:

1. Field selection (rearrangement of one or more fields)
2. Editing (packing or unpacking of numeric fields)
3. Sequence checking of input records
4. Sequence numbering of output records

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of parameter cards (if required) and the programmer's card input deck or paper-tape reel.

Output

♦ Output consists of a card file punched as directed by the control parameters. If no control parameters are submitted, the output file is identical to the input file.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader or Paper Tape Reader/Punch
1 Card Punch
1 Magnetic Tape Device
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages. Card counts of the input and output records are displayed at the termination of the routine. Sequence checking of input records and sequence numbering of output records are provided as programmer options.

TIMING

♦ (To be supplied)

CARD OR PAPER TAPE TO PRINTER

FUNCTIONAL DESCRIPTION

♦ The Card or Paper Tape to Printer routine prints information contained in an 80-column card or paper tape file (punched in EBCDIC format) under the direction of programmer-supplied control parameters. One of two output formats may be specified: Display or List.

The Display format provides a visual picture of the input file. Each input card starts on a new output line with positions reserved to indicate the card number. In this format data may be displayed in either a hexadecimal or character mode.

The List format provides a simple edited listing of the input file. In this format each logical record forms one line of printed output, with data being listed in either a hexadecimal or character mode.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of a card or paper-tape file that is punched in EBCDIC format.

Output

♦ Output from the routine consists of a printed copy of the input file in the format specified by the user.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader or Paper Tape Reader/Punch
1 Printer
1 Magnetic Tape Device
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages. At the termination of this routine a card count and a page count are displayed on the console typewriter. The programmer may also specify a sequence check on the input records.

TIMING

♦ (To be supplied)

TAPE TO CARD

FUNCTIONAL DESCRIPTION

♦ The Tape-to-Card routine transcribes fixed-length records stored on magnetic tape (7 or 9 level) to 80-column punched cards. The input tape records may be blocked or unblocked; output cards are punched in standard EBCDIC format.

By means of parameters the following options are also provided:

1. Multireel input
2. Positioning of input tape
3. Label handling
4. Unblocking of input records
5. Field selection (rearrangement of input fields to card output fields)
6. Packing or unpacking of input field data before transfer to the output card field
7. Sequence numbering of output cards

The numbers of input blocks processed and output cards generated are displayed on the console typewriter at the end of the routine.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input records are fixed-length, blocked or unblocked. When the field selection option is used, the portion of the input record transferred to the output card cannot exceed 80 characters.

Output

♦ Each output record contains the contents of one logical input record, punched in EBCDIC format.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader
2 Magnetic Tape Devices
1 Card Punch
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine performs validation of input parameter records and produces a standard set of error messages.

TIMING

♦ (To be supplied)

TAPE EDIT

FUNCTIONAL DESCRIPTION

♦ The Tape Edit routine displays the contents of a magnetic tape, which has been recorded in EBCDIC format, on the on-line printer. The edited output is displayed in both alphanumeric and hexadecimal format.

Optional control parameters are provided to allow for the following functions:

1. Positioning the input tape before or after printing.
2. Printing a specified number of blocks or files.
3. Printing pre-edited tapes (output from Memory Print and Trace Routines).
4. Provision for programmer-specified header lines.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of a 7- or 9-level magnetic tape that has been recorded in EBCDIC format. Input blocks may be fixed or variable in length, blocked or unblocked. Pre-edited tapes produced by the Memory Print and Trace routines are also accommodated.

Output

♦ Standard output from this routine is an on-line printer listing showing both the alphanumeric and hexadecimal equivalents of each eight-bit input character. The programmer has the option to select only alphanumeric or hexadecimal format.

When pre-edited tapes are used as input, the input data is transcribed to the printer without modification.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader
1 Printer
2 Magnetic Tape Devices
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages.

TIMING

♦ (To be supplied)

TAPE TO PRINTER

FUNCTIONAL DESCRIPTION

♦ The Tape-to-Printer routine displays the contents of a magnetic tape file on the on-line printer. Two print formats are provided: Display and List.

The Display format presents a visual representation of the data file in hexadecimal format. The output line contain descriptive information about the file, such as block size, block number, and record number.

The List format provides a simple alphanumeric edited listing of data files that contain fixed-length records. With this format each logical record forms one print line. In addition, the programmer has the facility to select various editing options in transferring fields from the input record to the print record. These options include field selection, pack, unpack, and hexadecimal representation.

This routine includes input tape positioning and label checking as standard functions. Optional functions include multireel input, positioning to the first logical record to be printed, fixed- or variable-length blocked input records, page headings, suppression of page numbering, and double or triple spacing between lines.

INPUT-OUTPUT DESCRIPTION

Input

♦ This routine accepts magnetic tape (7 or 9 level), single or multi-volume. Records may be fixed or variable in length, blocked or unblocked.

Output

♦ Output of this routine is a listing of the tape file on the on-line printer.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader
2 Magnetic Tape Devices
1 Printer
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages.

Tape block counts and page counts are displayed on the console typewriter at the termination of the routine.

TIMING

♦ (To be supplied)

TAPE TO TAPE

FUNCTIONAL DESCRIPTION

♦ The Tape-to-Tape routine transcribes all, or selected portions, of an input magnetic tape to an output magnetic tape. The input file records can be blocked or unblocked; fixed, variable, or undefined in size.

By means of parameter control cards programmer can exercise the following options during the transcription process:

1. Multireel input or output
2. Tape positioning, both input and output
3. Tape label handling
4. Unblocking of input records
5. Blocking of output records
6. Field selection* (rearrangement of input fields to output fields)
7. Packing or unpacking of input fields*

*For fixed-length records only.

INPUT-OUTPUT DESCRIPTION

Input

♦ This routine accepts single or multivolume magnetic tape (7- or 9-level).

Output

♦ Output of this routine is a single or multivolume magnetic tape (7- or 9-level).

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)
1 Card Reader
3 Magnetic Tape Devices
1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ (Not applicable)

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages.

Input and output block counts are displayed on the console typewriter at the termination of the routine.

TIMING

♦ (To be supplied)

RANDOM ACCESS DATA TRANSCRIPTION ROUTINES

FUNCTIONAL DESCRIPTION

♦ The Random Access Data Transcription routines are a group of utility routines that transcribe data to or from a random access device. In transcribing data to an output random access device, the input may appear on cards, magnetic tape, another random access device, or the same random access unit as the output device. Transcription from a random access device can be directed to cards, magnetic tape, or the on-line printer.

By means of control cards, these routines can be directed to perform one or more of the following functions:

1. *Copy* - This function transcribes data from the input device to the output device without change.
2. *Reblock* - This function is used when data transcribed to or from magnetic tape is to be blocked or unblocked.
3. *Field select* - This function is used when input data is to be reformatted before being transcribed to the output device.
4. *Reblock and field select* - This function is used when a combination of the Reblock and Field Select options is desired.
5. *List* - This function is used when transcribing data from a random access device to the printer. Output is restricted to one print line per record and is listed in hexadecimal format. Page headings and page numbering may be specified.
6. *Display* - This function is used when transcribing data from a random access device to the printer. Output is displayed in both alphanumeric and hexadecimal format, although either format can be suppressed. When using this function, an input record may extend over several print lines. Page headings and page numbering can also be specified.
7. *Display and Field Select* - This function is a combination of the Display and Field Select options and is used when an edited print-out of random access records is desired.

INPUT-OUTPUT DESCRIPTION

Input

♦ When data is transcribed to a random access device, input to this routine can include all, or selected portions of, a magnetic tape, a card file, another random access device, or the same random access device.

Output

♦ When data is transcribed from a random access device, all or selected portions of the random access area can be transcribed to magnetic tape, to an on-line printer, to a card file, to another random access device, or to the same random access device.

EQUIPMENT REQUIREMENTS

Minimum Equipment

- 1 Processor (65K bytes)

  1 Console Typewriter

  1 Random Access Device

  1 Magnetic Tape Device

  1 Card Reader

  Magnetic Tape Devices, Card Reader, Card Punch, Printer (as required for input-output devices)

MEMORY REQUIREMENTS

- (To be supplied.)

RELATED PROGRAMMING COMPONENTS

- Not applicable.

ACCURACY CONTROL

- This routine validates input parameters and produces a standard set of error messages.

TIMING

- Timing is variable, depending on the input and output device used and extent of data manipulation (blocking, field select, etc.).

AUTOMATIC INTEGRATED DEBUGGING SYSTEM

FUNCTIONAL DESCRIPTION

♦ The Automatic Integrated Debugging System (AIDS) is designed to minimize the time taken for program testing, thereby maximizing computer efficiency. This system provides programmers with a simple method of collecting their programs and test data for off-site (remote) program testing.

AIDS is a two-phase routine. In the first phase, the program(s) and associated test data are collected onto a single input test tape, together with control information for memory prints and tape edits.

At program test time, phase two is executed by the console operator. This phase distributes the test data on the input tape to those input tape devices specified by the programmer in phase one. The object program is then loaded and executed. At the conclusion of each test, a memory print and an edit of the output tapes are optional.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of the program(s) and test data, which can be stored on cards or magnetic tape. In addition, parameter cards are included that identify the programs and test data sets to be collected onto the test tape, as well as the diagnostic routines desired.

Output

♦ Output of this routine consists of a single output test tape in loadable format for program testing.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

1 Card Reader

4 Magnetic Tape Devices (9 level)

1 Console Typewriter

Optional

♦ Additional magnetic tape devices may be used as input to phase one.

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ Not applicable.

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages.

TIMING

♦ Not applicable.

CONSOLE ROUTINES

FUNCTIONAL DESCRIPTION

♦ To limit intervention by the operator as much as possible, a number of diagnostic routines available in the system can be called in by means of the console typewriter. Functions that normally require extensive manual console manipulations can be initiated by a minimum of parameters entered at the typewriter. The console routines available to the operator are as follows:

APPLY PATCHES	DISPLAY GENERAL REGISTERS
OPEN DIAGNOSTIC DEVICE	DISPLAY FLOATING-POINT REGISTERS
CLOSE DIAGNOSTIC DEVICE	DISPLAY HIGH-SPEED MEMORY
TAPE EDIT	DISPLAY DEVICE RETURN TABLE
MEMORY PRINT	DISPLAY PROGRAM START TABLE
SNAPSHOT	DRUM PRINT
DISPLAY ENVIRONMENT	DRUM INDEX EDIT

1. *Apply Patches* - This routine provides the ability to apply object-time patches from the card reader, console typewriter, paper tape or magnetic tape.

2. *Open Diagnostic Device* - This routine specifies the device that is to receive the output of certain diagnostic routines. All debugging routines requested use this output device.

3. *Close Diagnostic Device* - This routine inhibits the use of a particular output device by those diagnostic routines that previously recognized the device as *open*.

4. *Tape Edit* - This routine permits the operator to print the entire contents of a magnetic tape or selected files on the tape.

5. *Memory Print* - This routine provides a printer listing of all or parts of high-speed memory including the console area and register settings.

6. *Snapshot* - By means of this routine the programmer may specify selected portions of memory to be dumped at program intervals of his choice.

7. *Display Environment* - This routine displays, via the console typewriter, tables used by the control system that are associated with the object program.

8. *Display General Registers* - This routine transcribes to the console typewriter the general register contents at the time of the request.

FUNCTIONAL DESCRIPTION (Cont'd)

9. *Display Floating-Point Registers* - This routine transcribes to the console typewriter the contents of the floating-point registers at the time of request.

10. *Display High-Speed Memory* - This routine displays to the console typewriter, in hexadecimal format, the contents of specified memory areas.

11. *Display Device Return Table* - This routine prints on the console typewriter all I/O devices associated with a particular program.

12. *Display Program Start Table* - This routine indicates the program name, starting address, and priority of the program or program segment.

13. *Drum Print* - This routine provides a printer listing of parameter specified drum areas.

14. *Drum Index Edit* - This routine edits the control information of the Drum Index Table and transcribes it to the console typewriter in hexadecimal format.

INPUT-OUTPUT DESCRIPTION

Input

♦ The input to the console routines is by means of parameters inserted by the operator using the console typewriter.

Output

♦ See routine description.

EQUIPMENT REQUIREMENTS

♦ Processor (65K bytes)

Console Typewriter

2 Magnetic Tape Devices (9 level)

1 Random Access Device

NOTE: Additional equipment requirements depend upon the individual routines.

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ Not applicable.

ACCURACY CONTROL

♦ These routines validate input parameters and produce a standard set of error messages.

TIMING

♦ (To be supplied.)

MEMORY PRINT

FUNCTIONAL DESCRIPTION

♦ The Memory Print routine is a program testing aid used to display the general registers, channel status indicators, and designated core areas during the execution of the program. This data is normally displayed on-line, but a magnetic tape may be used for off-line printing.

The programmer selects those points at which memory prints are to be taken by inserting parameters at appropriate places in his assembly source program. These parameters are then assembled as part of the program.

The description of the areas to be printed may be supplied in the assembled source program or at execution time by means of console type-ins or parameter control cards.

The Memory Print routine can be incorporated in the program at assembly time, or this routine can be bound to the program through the Linkage Editor routine.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of a program and parameter information describing the core areas to be printed.

Output

♦ Output consists of an edited printer listing of the contents of the general registers, channel status indicators, and the user-selected area of memory. This listing can be displayed on the on-line printer or written to magnetic tape for subsequent printing.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

1 Card Reader

1 Printer

1 Magnetic Tape Device

1 Console Typewriter

Optional

♦ A magnetic tape device may be substituted in lieu of the on-line printer for off-line listings.

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ This routine operates in conjunction with a programmer problem program. It may be assembled with the problem program or linked to it by means of the Linkage Editor.

When the output from Memory Print routine is written to magnetic tape, the Tape Edit routine is required for subsequent printing.

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of accuracy controls.

TIMING

♦ (To be supplied.)

SELF-LOADING MEMORY PRINT

FUNCTIONAL DESCRIPTION

♦ The Self-Loading Memory Print routine is a program testing aid used to display the contents of core when a program under test has terminated abnormally.

The printed output is displayed in hexadecimal and graphic format. This data is normally displayed on-line, but a magnetic tape may be used for off-line printing.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of the starting and ending addresses of the memory area to be printed, supplied at object running time.

Output

♦ Output from this routine is a printed listing of the contents of the channel status indicators, the general registers, and the memory area defined by the operator. At the operator's discretion the output may be printed directly to the on-line printer, or written to magnetic tape for printing at a later time.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

1 Card Reader

1 Printer

1 Magnetic Tape Device

1 Console Typewriter

Optional

♦ A magnetic tape device may be substituted in lieu of the on-line printer for off-line listings.

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ When the output from this routine is written to magnetic tape, the Tape Edit routine is required for printing the output tape.

ACCURACY CONTROL

♦ Not Applicable.

TIMING

♦ (To be supplied.)

TAPE COMPARE

FUNCTIONAL DESCRIPTION

♦ The Tape Compare is a diagnostic routine that enables the programmer to compare the contents of two magnetic tapes. During the comparison process any discrepancies that exist between the two input tapes are displayed on the on-line printer.

The programmer can elect to compare the entire contents of the input tapes, or by optional parameter cards, specify certain portions only.

INPUT-OUTPUT DESCRIPTION

Input

♦ The input to this routine consists of two volumes of magnetic tapes, plus parameter cards if positioning of input tapes is required for selective comparison.

Output

♦ The output of this routine is a printer listing of the differences existing between the two tapes, displayed in hexadecimal format. If the contents of the tapes are identical, a print-out to this effect is given at the end of the routine.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

3 Magnetic Tape Devices

1 Printer

1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ Not applicable.

ACCURACY CONTROL

♦ This routine validates the input parameters and produces a standard set of error messages.

TIMING

♦ (To be supplied.)

TEST DATA GENERATOR

FUNCTIONAL DESCRIPTION

♦ The Test Data Generator routine automatically creates program test data from programmer-supplied parameter descriptions and transcribes this data to cards or magnetic tape. Multifile volumes and multivolume files can be generated, with record size, blocking factor, and the number of fields to appear in the output records determined by the programmer.

Provision is made for the generation of alphanumeric or numeric data in the output record fields, with the base value for these fields supplied via input parameters. In addition, the programmer may include an increment value for any field that will be added to the designated field each time a new output record is generated.

INPUT-OUTPUT DESCRIPTION

Input

♦ The input to the Test Data Generator routine consists of programmer supplied parameter control cards that specify the type of data generation desired.

Output

♦ The output of this routine is a card or magnetic tape file containing the generated test data.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

2 Magnetic Tape Devices

1 Card Reader

1 Console Typewriter

1 Card Punch or Magnetic Tape Station

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ Not applicable.

ACCURACY CONTROL

♦ This routine validates input parameters and produces a standard set of error messages.

TIMING

♦ Not applicable.

TRACE

FUNCTIONAL DESCRIPTION

♦ The Trace routine is a program testing aid that provides diagnostic information regarding the instructions executed in a program and the status of registers after the instructions have been executed. This routine is normally used when the programmer is unable to isolate a problem using the standard testing routines such as memory dumps and snapshot prints.

When this routine is used, the programmer supplies the addresses of the first and last instructions to be traced. Whenever control is transferred to any instruction within the defined area, the routine displays the instruction being executed, its core location, the contents of the general registers utilized by the instruction, and the status of the condition code after the instruction has been executed. The programmer is thus provided with a detailed picture of the sequence of conditions that occur during the running of his program.

By using control parameters, the programmer can direct the routine to defer tracing until the instructions within the area to be traced have been executed a specified number of times. In addition, the output from this routine can be specified to be written to the on-line printer or to a magnetic tape for printing at a later time.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of a program and parameter information designating the program area to be traced.

Output

♦ Output from this routine is an on-line edited listing of the executed program instructions and the contents of selected registers associated with each instruction, printed in hexadecimal or graphic format. An off-line listing can be produced at the programmer's option.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor

1 Card Reader

1 Printer

1 Magnetic Tape Device

1 Console Typewriter

Optional

♦ A magnetic tape device may be substituted in lieu of the on-line printer for off-line listings.

MEMORY REQUIREMENTS

♦ (To be supplied)

RELATED PROGRAMMING COMPONENTS

♦ The Trace routine is assembled independently of the program and combined with the program by means of the Linkage Editor routine. The Trace routine must reside in memory at the time that the program is executed.

If the Trace output is written to tape, the Tape Edit routine is used to obtain the printed listings.

ACCURACY CONTROL

♦ This routine validates input parameter records and produces a standard set of error messages.

TIMING

♦ (To be supplied)

PROGRAM SECTION LIBRARY MAINTENANCE ROUTINE

FUNCTIONAL DESCRIPTION

♦ The Program Section Library Maintenance routine is a service routine that provides the programmer with a means for inserting, replacing or deleting program control sections from a Program Section Library tape.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to this routine consists of:

1. Control cards defining the program sections to be inserted, replaced or deleted, followed by an End-of-Input card.
2. A Program Section Library tape.
3. A Program Section File on magnetic tape that contains the program sections that are to replace corresponding program sections on the Program Section Library and any new sections that are to be inserted into the Program Section Library.

Output

♦ Output from this routine is an updated Program Section Library tape.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

1 Card Reader (or a 9-level magnetic tape station)

3 Magnetic Tape Devices (9 level)

1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ The Program Section File used by this routine is the standard output generated by the Assembly, Report Program Generator, FORTRAN or COBOL systems.

ACCURACY CONTROL

♦ This routine produces a standard set of error messages.

TIMING

♦ (To be supplied.)

LINKAGE EDITOR

FUNCTIONAL DESCRIPTION

♦ The Linkage Editor routine is a service routine that is used for the maintenance and creation of programmer Load Library tapes. This routine accepts program sections from an input magnetic tape and binds these sections into loadable programs on the programmer's Load Library tape. The facility is also provided for selecting and updating programs from an existing Load Library, as directed by input control cards.

INPUT-OUTPUT DESCRIPTION

Input

♦ Input to the Linkage Editor consists of:

1. Control cards that specify the Linkage Editor functions and the load structures of the programs to be bound, followed by an End-of-Input card.
2. A Program Section Library tape containing the program sections to be bound into loadable programs on the output tape.
3. A programmer Load Library tape.

Output

♦ Output from this routine is a new programmer Load Library tape.

EQUIPMENT REQUIREMENTS

Minimum Equipment

♦ 1 Processor (65K bytes)

1 Card Reader

5 Magnetic Tape Devices

1 Console Typewriter

MEMORY REQUIREMENTS

♦ (To be supplied.)

RELATED PROGRAMMING COMPONENTS

♦ The output Load Library tape is in a format acceptable for loading and execution by the Tape Executive routine.

ACCURACY CONTROL

♦ This routine validates input parameters and produces a standard set of error messages. In addition, this routine displays the names of any program sections designated for linkage, addition, selection or deletion that were not found on the specified source tape.

TIMING

♦ (To be supplied.)