IBM

Y20-0013-0

SALES and SYSTEMS GUIDE

Operating System/360 BTAM User's Guide

Terminal-Dependent Modifications

Preliminary Edition

This publication contains detailed information on the structure and operation of the Basic Telecommunications Access Method (BTAM) program support. It is designed to aid programmers who need to make terminal-dependent modifications to BTAM. An example is included showing the modifications necessary for an IBM 2740 terminal.

This manual is being released at the present level (corresponding to the BTAM level described in C28-6553-0) to benefit those who need early information on BTAM operation and modification. It does not replace IBM Operating System/360, Telecommunications: Preliminary Specifications (C28-6553), nor will it replace the yet to be released Telecommunications Program Logic Manual, both of which are to be referred to for final information on BTAM support.

RESTRICTED DISTRIBUTION

CONTENTS

Address comments concerning the contents of this publication to
IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

INTRODUCTION

Support for telecommunication systems is provided in the form of access methods under the Data Management portion of Operating System/360 Control Programs. Access methods are provided at both the basic (READ/WRITE) level and the queued (GET/PUT) level and are termed the Basic Telecommunications Access Method (BTAM) and the Queued Telecommunications Access Method (QTAM) respectively (see IBM Operating System/360, Telecommunications: Preliminary Specifications, C28-6553).

Terminal devices and telecommunications system control methods that are currently supported by QTAM and BTAM are listed in C28-6553 and in the Programming Section of the IBM Data Processing Sales Manual. There are, however, individual systems that will require modification of BTAM or QTAM because of a particular terminal device or method of operation. Candidates for this classification include Request for Price Quotation (RPQ) terminals, newly developed IBM terminals, non-IBM terminals, and those that are now considered insufficiently numerous to warrant official support. It is with these systems that this publication is concerned.

This manual is concerned only with BTAM. A later manual will contain comparable material on QTAM. It was decided to have two separate manuals in order to make the information available as it is developed and because of the prerequisite nature of BTAM relative to QTAM.

This manual is intended to be a modification guide. It is realized, however, that general procedures cannot be devised to handle all modifications that will become necessary. The intention is to provide the user with sufficient knowledge of the subject so that he can relate his system to the available programs and devise the necessary modifications. In large part, the value of this manual is in the education it provides on the contents and operation of BTAM. For those sections where a definite modification is needed, however, specific instructions are given.

To make use of this manual the user is expected to be thoroughly familiar with the requirements of the unsupported terminal device. These include such things as the line control procedure, transmission code, message formats, and hardware support. The user must also know the IBM System/360 Assembler Language. QTAM and BTAM consist of macro instructions and routines that make exclusive use of the assembler language, and it is with this language that modifications are to be made.

This manual consists of the following seven sections:

1. BTAM Operation — describes the operation of BTAM and its parts.

2. Preparation — describes the materials needed to modify BTAM.

3. Probable Modifications — details the new coding and revisions to the original that are most likely to be needed.

4. Possible Modifications — describes the areas most likely affected.

5. BTAM Listing Description — details the macro instructions and modules that make up the BTAM program.

6. Appendices — contain detailed information on control blocks, tables, and linkages. Appendix L shows the development of modifications for the IBM 2740 Communication Terminal*.

7. Bibliography

Emphasized again is the fact that this is a modification guide, not a stand-alone publication. The reader is assumed to be familiar with all the SRL manuals listed in the Bibliography.

*This does not constitute the official 2740 terminal support.

BTAM OPERATION

The Basic Telecommunications Access Method (BTAM) provides for reading from and writing to terminal devices attached to communications lines. Processing programs making use of BTAM may consider communications lines to be input/output devices that will be polled or addressed by the execution of READ or WRITE macro instructions.

BTAM relieves the user of the details of communication line control by providing a number of macro instructions to be used within his program. For instance, to receive a message from a terminal a READ macro can be written in the user program. This macro instruction will be expanded by the OS/360 assembler into a sequence of coding that will provide a linkage to the appropriate BTAM routine. Execution of this coding and the BTAM routine will establish contact with the terminal and take the message, if any, into a buffer area. Other BTAM macro instructions and their functions are listed below (see C28-6553 and a later section of this manual for more detail):

WRITE — contacts a terminal and sends a message to it from an output buffer area.

DCB — creates and initializes a data control block for a communications line group data set. This is the only data set essential to BTAM.

OPEN — prepares communications line groups defined by DCB for use.

WAIT — relinquishes control of the CPU to the OS/360 supervisor until a specific event, such as an I/O operation, has been completed.

CLOSE — removes communications line group data sets from use.

DFTRMLST — creates lists of terminal addresses that will define the order in which the terminals will be polled or addressed.

CHGNTRY — allows deactivating or reactivating a terminal in a polling/addressing list without redefining the list.

RESETPL — interrupts a polling sequence, or an ENABLE that has not completed (dial-up).

Figure 1 illustrates the aspects of a BTAM Read operation. From top to bottom these are:

System Generation

Assembly

Execution - Open

Execution - Read

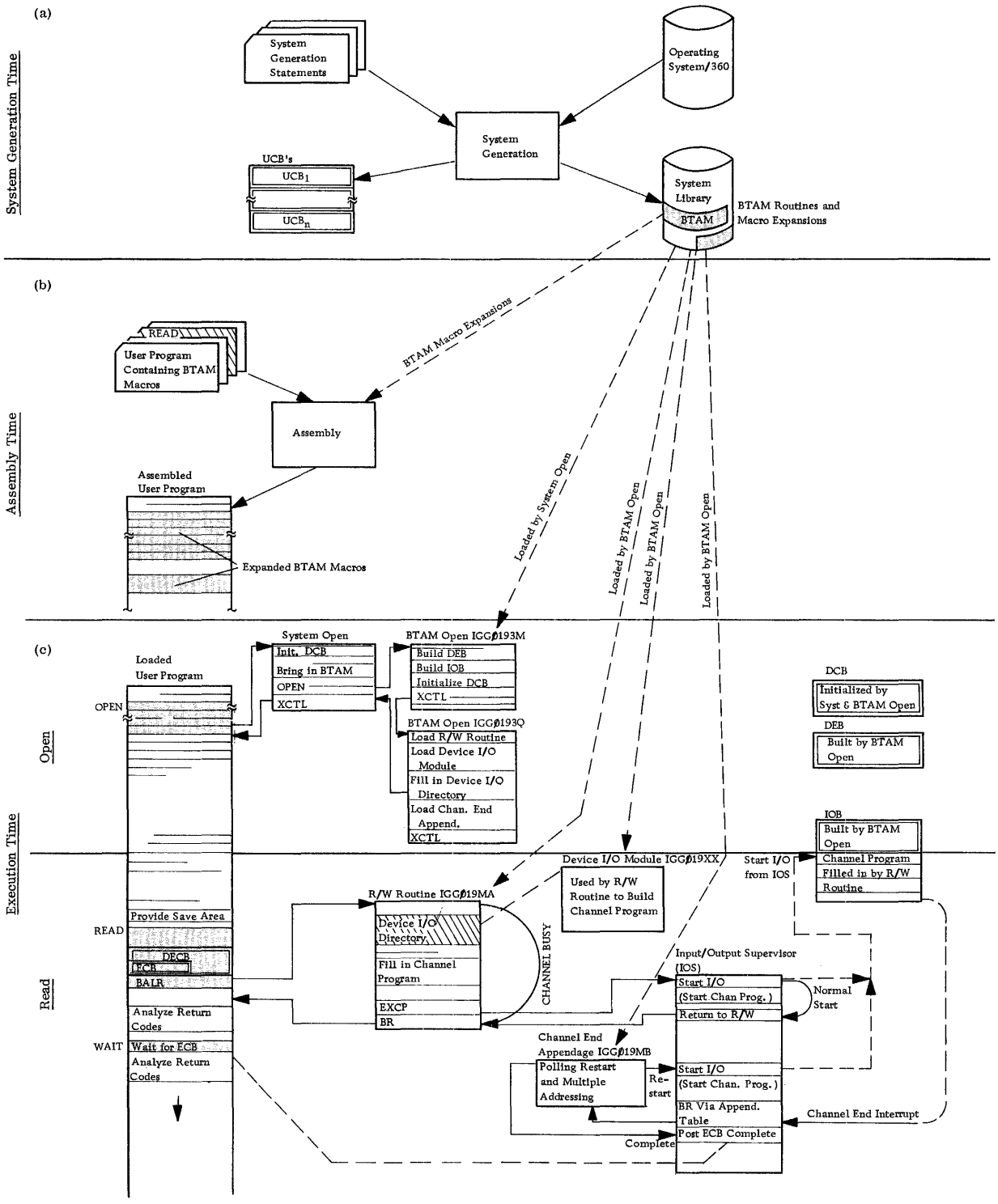


Figure 1. BTAM Read Operation

Not all BTAM macros needed for a Read operation are illustrated in Figure 1. Macros such as DCB, DFTRMLST, and CLOSE are necessary and would appear somewhere in the user program. They are excluded only to simplify the diagram. A BTAM Write operation could be illustrated in a similar diagram.

SYSTEM GENERATION

This aspect of BTAM operation is shown for background purposes and does not intend to describe the complete system generation procedure. It does show the necessity of describing pertinent details of the communication system hardware by use of system generation statements that result in the forming of unit control blocks (UCB — Appendix G). The UCB's so created will subsequently be used by the BTAM Open routine to determine the types of terminal devices involved.

Also as a result of system generation the BTAM routines and macro definitions are included in the system library.

ASSEMBLY

All BTAM macros are represented by macro definitions in the system library. This is true for all BTAM macros, whether they are purely BTAM macros (for example, DFTRMLST, CHGNTRY) or specialized parts of system macros used by BTAM (for example, READ, WRITE).

When the assembler language problem program that will be using BTAM is assembled, each macro instruction imbedded in the program is replaced by its appropriate macro expansion. The expansion may consist of assembler instruction statements (for example, DC, DS), symbolic machine instruction statements (executable machine instructions), or both.

EXECUTION

A BTAM Read operation, during execution of a user program, is shown in Figure 1 to consist of those functions performed during Open and those functions performed during Read.

Opening of the data set (the communications line group) is usually performed early in the user program. In any case, it is necessary that it be performed before that data set is referenced by a READ or WRITE macro instruction. Closing of the data set (CLOSE macro) is not shown in Figure 1. This macro would normally be placed at a point in the user program where it would be executed when no more references are to be made to the data set.

Defining the data control block (DCB — Appendix B) with the DCB macro and the terminal list with the DFTRMLST macro is normally done in the definition section of the program (along with DC and DS statements) and, of course, does not result in the generation of executable machine instructions.

Open

Opening, or making ready for use, the communications line group data set consists of sequentially proceeding through:

1. OPEN macro expansion coding

2. System Open Routine

3. BTAM Open Routine (Module 1)

4. BTAM Open Routine (Module 2)

The OPEN macro expansion coding is that system OPEN macro coding that is appropriate for a BTAM data set (communications line group). Execution of this expansion coding provides for identifying the specified DCB and issuing a supervisor call (SVC) for the system open routine. Parameters such as INPUT, OUTPUT, and INOUT are ignored by BTAM. (BTAM always OPENs for both input and output.)

The system open routine partially initializes the specified DCB and requests that BTAM Open Module 1 (IGG0193M) be brought into core storage and executed, via an XCTL macro instruction. (BTAM coding consists of a number of modules, or control sections, that are loaded and executed as separate but related units.) The two BTAM Open Modules are executed serially in the supervisor transient area. They are re-entrant and operate interrupt-enabled in supervisor mode.

BTAM Open Module 1 reserves storage for and initializes data extent blocks (DEB — Appendix C) and input/output control blocks (IOB — Appendix D). Information found in the UCB's created by system generation is used for this purpose. BTAM Open Module 2 (IGG0193Q) is loaded and given control via an XCTL macro instruction. This module loads (via the LOAD macro) into core storage the remaining BTAM modules needed. These include the Read-Write routine (IGG019MA), the Channel End Appendage (IGG019MB), and the Device I/O Modules (IGG019xx - where xx represents the code assigned to each module) for the particular terminal device involved. All of these modules are only loadable and cannot be transferred to via an XCTL macro. A directory of I/O modules is created within a section of the Read-Write module and, depending on the particular terminals, SAD and ENABLE commands are issued to the communications lines via the Input/Output Supervisor (IOS). To complete the Open process, control is now passed from IOS to the last load module of System Open and then to the user program at the point immediately following the OPEN macro expansion. The communication lines are now ready for operation.

Read

One READ macro expansion is shown in the user program in Figure 1. This is representative of operating a single communication line in the system. Many lines may be operated concurrently by issuing READs (or WRITEs) for each line before "waiting" for the completion of one or more.

At the point in the user program where a Read operation is desired the user must do the following in sequence:

1. Specify in Register 13 the address of a save area for storing the general register values when control is passed to the Read-Write routine.

2. Issue a READ macro instruction.

3. Analyze the codes returned by BTAM in Register 15, indicating whether the operation was initiated successfully (see Appendix K).

4. Issue a WAIT macro instruction at the point beyond which execution is to proceed only after the Read operation is complete.

5. Analyze the completion code in the event control block to determine whether the operation was completed successfully.

The expansion of a READ macro by the assembler results in initializing a data event control block (DECB — Appendix F) with the parameters specified with the READ macro. The DECB is generated or updated by the macro expansion and contains within it an event control block (ECB — Appendix E).

The DECB is the only direct communications link between the access method (BTAM) and the user program. The ECB within it is the entity upon which a WAIT is made (that is, a WAIT macro parameter) and in which completion is posted by the supervisor when the Read operation is completed.

The prime purpose of the Read-Write routine is to construct the channel program for the particular type of Read or Write desired, (that is, read initial, read repeat, etc.), making use of the device-dependent information contained in the appropriate device I/O module. Each device I/O module contains, along with special characters and a table of offsets, a number of model channel programs — one for each type of operation (Figures 12-14). Each model channel program consists of a number of model channel command words (CCW's) — there being a model CCW for every actual CCW in the channel program to be built.

Just as the DECB is considered the link between the user program and BTAM, the IOB can be considered the link between BTAM and the Input/Output Supervisor (IOS). The channel program built by the Read-Write routine in the IOB field reserved for it has its starting address indicated to IOS in a field in the IOB.

Once the channel program has been built and passed to IOS for execution, control will be returned to the user program, with return codes in Register 15 to indicate whether the initiation of the Read was successful (see Appendix K). The user program may then proceed with its execution up to a point in the program where a WAIT is specified for completion of the Read operation.

Execution of the channel program proceeds concurrently with the execution of the user program until a Channel End condition occurs, whereupon IOS regains control and enters the Channel End Appendage. This condition may have been caused by either completion of the Read operation or the need to update and restart the channel program, as would be the case for polling the next terminal in the terminal list after a negative polling response.

If the Read operation is completed, the Supervisor will post completion in the ECB specified by the READ macro. The WAIT macro referencing this ECB will then be satisfied, allowing the user program to continue its execution. At this point the user program should analyze the completion code returned by IOS in the ECB (see Appendix E). Further action in the user program should be based on the results of this analysis.

This completes the entire Read operation, after which the user program may continue with processing or handling of the message obtained from the terminal.

PREPARATION

This section describes the preparation and physical materials needed to modify BTAM.

Modifications to BTAM for operation of an unsupported terminal are considered as either probable modifications that are most likely needed for any unsupported terminal, or possible modifications that may or may not be needed for a particular unsupported terminal. Both situations are treated in succeeding sections.

Assuming at this point that the reader knows the operation of the unsupported terminal, OS/360 Assembler Language, and the general operation of the BTAM program, we are ready to proceed with the actual modifications. BTAM consists of a number of modules of OS/360 Assembler Language coding in the form of macro definitions, program routines and tables, and device-dependent modules for each terminal device. (BTAM modules are sections of coding modified by the Linkage Editor so that they can be brought in and executed as separate units.) In general, a source deck and listing of each module are needed and can be obtained from the OS/360 System Library.

MACRO DEFINITIONS

Macros such as DCB, OPEN, CLOSE, READ, and WRITE are actually system macros, some of which result in specialized expansions when specified for BTAM. Other BTAM macros such as DFTRMLST and RESETPL are exclusively for BTAM. Only those macros involved in the modification need be requested, since their functions are described in this document and their definitions contain a large amount of coding that is not useful for the problem at hand. BTAM and system macros of interest are DCB, OPEN, CLOSE, DFTRMLST, READ, WRITE, RESETPL, CHGNTRY, and WAIT.

PROGRAM MODULES

Name	Module Identification
BTAM Open Routine (Load 1)	IGG0193M
BTAM Open Routine (Load 2)	IGG0193N
BTAM Read-Write Routine	IGG019MA
BTAM Channel End Appendage	IGG019MB
BTAM Close Routine*	IGG0203M

*Not available at this writing.

DEVICE-DEPENDENT MODULES

Only the modules for the terminal devices of interest are needed; however, they should all be requested because they are small and can be used for ideas for constructing new modules. Other modules will be included as they are completed.

Name	Module Identification
IBM 1050, Device I/O Module	IGG019MD (Figure 12)
AT&T 83B3 Device I/O Module	IGG019ML (Figure 13)
WU 115A Device I/O Module	IGG019MN (Figure 14)

Each of these materials is described in detail in the section entitled "BTAM Listing Description (BLD)". The BLD explains the functions and operation of all BTAM modules and gives cross-references to the listing as well as references to the control block fields affected.

At this point it would be advisable to read the introduction to the BLD and briefly run through its contents. Subsequent sections will reference parts in the BLD while discussing modifications.

PROBABLE MODIFICATIONS

Modifications necessary for BTAM to support any terminal device not currently supported will be discussed here. (Modifications not necessary for every unsupported terminal will be treated in a subsequent section entitled "Possible Modifications".) The discussion to follow is written for any terminal device in general, using the IBM 1050 terminal as an example. Appendix L shows the procedure for modifying BTAM to handle a specific version of the IBM 2740 terminal. The necessary modifications involve:

1. Device I/O Modules (see BLD, Section C)

2. Device Type Analysis section of Open Module 1 (see BLD, Section B)

Development of a new Device I/O Module will require writing out a detailed line control procedure for the terminal to be supported. These line control procedures will be expanded into model channel programs, which will be collected into a Device I/O Module.

Modification for the Device Type Analysis section will involve slight program logic changes and an additional entry to each of two tables.

BUILDING A DEVICE I/O MODULE

Line Control Procedure Specification

At this point the user must concern himself with the details of operating his terminals. After making a thorough study of the desired operating procedure, he will know whether there will be polling, addressing, multisegment messages, error retransmissions, or inquiry replies. Each operation requires a certain sequence of characters transmitted on the line to effect the necessary control. A common way to illustrate this is to draw a chart as shown in Figure 2, identifying the sequences that perform certain operations as Read Initial, Read Repeat, Write Initial, etc.

While preparing such a chart it should be possible to write a narrative of each operation in terms of reading and writing characters onto the lines (from a CPU point of view), such as is shown below for an IBM 1050 terminal.

1. Read Initial (Polling) — This operation establishes contact with a terminal and allows it to send a message to the CPU. In detail, it does the following:

   a. Write EOT control character (C)

   b. Write terminal and component select (polling) characters

   c. Read response to poll

   d. Read message into a buffer

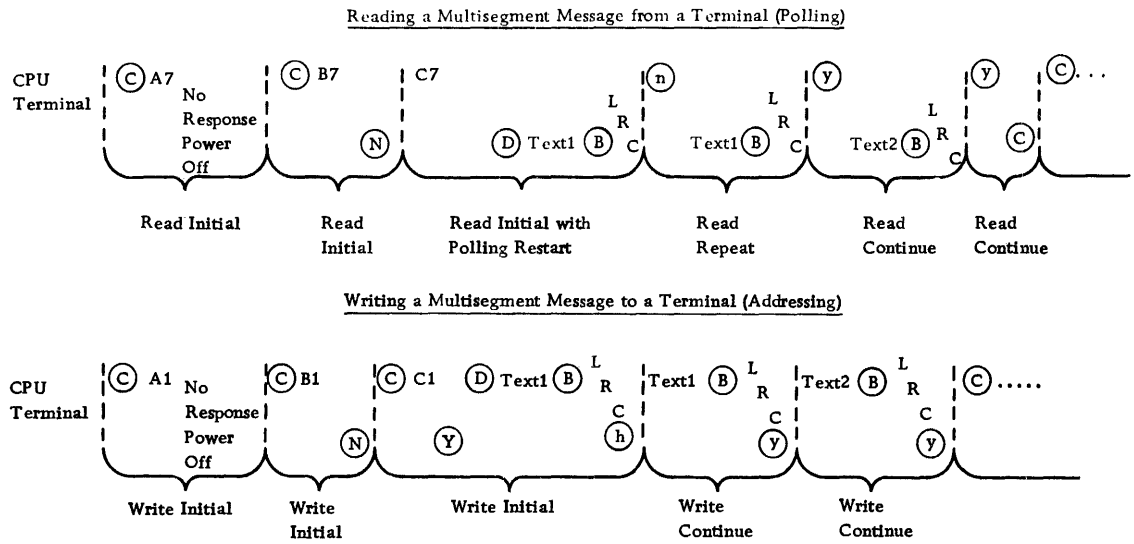


Figure 2. Examples of IBM 1050 line control

The read operation is terminated on receipt of the EOB-LRC check sequence in the message and control is returned to the BTAM user program. The program can examine the CSW field of the IOB and decide whether to perform a read continue or a read repeat operation.

2. Read Continue — This operation returns a positive answer (Y) to the LRC check and reads the next record.

   a. Write (Y)

   b. Read message into new buffer

3. Read Repeat — This operation returns a negative answer to the LRC check, telling the terminal to resend the last record.

   a. Write (N)

   b. Read message into same buffer

4. Write Initial (Addressing) — This operation establishes contact with a terminal (or group of terminals) and sends out a message.

   a. Write EOT control character (C)

   b. Write terminal and component select (addressing) characters

   c. Read response to addressing

   d. Write message from buffer

   e. Read answer to LRC check

   f. Write a (C) character to end the operation (only if write with reset is specified)

5. Write Continue — Write another record and read answer.

   a. Write message

   b. Read answer to LRC

   c. Write a (C) to end the operation (only if write with reset is specified)

When writing to a terminal, the message in core storage should have the appropriate line control characters such as (D) (EOA), CR/LF, and (B) (EOB) included in their proper places. These characters will also be embedded in messages received from a terminal, and should be allowed for.

Other special control characters such as (C), (n), and (y) for the 1050 are stored in a table in the Device I/O Module; polling and addressing codes are defined in terminal lists within the user program by the DFTRMLST macro.

Write out the line control procedure for the unsupported terminal in the same form as the IBM 1050 example above. Appendix L shows the same procedure followed for the IBM 2740 terminal.

Construction of Device I/O Module from Line Control Procedure

The remaining steps to building a Device I/O Module consist of the following:

1. Set up a table of special characters for line control

2. Write out detailed command sequences

3. Build model CCW's from line control commands and character table

4. Form the model channel programs and character table into a Device I/O Module preceded by an offset table

Each step will be explained in the above order. Completion of these steps will result in a Device I/O Module that may be assembled and incorporated in the system library as a module of BTAM.

1. Set up a table of special characters for line control. All of the line control characters or character sequences that are needed for operation of the unsupported terminal device are to be defined in a table, each one immediately preceded by its length. They may be listed in any order in the table. Figure 3 shows such tables for the IBM 1050 and AT&T 83B3 line control characters. Tables of code structures (S/360 internal byte representations for 1030, 1050, 1060, 1070, 83B2/83B3, 115A, World Trade, and TWX systems) may be found in the SRL publication IBM System/360 Component Description, IBM 2702 Transmission Control (A22-6846).

Name	Operation	Operand	Remarks		Value
TABLE1	DC	X'01'	LENGTH = 1		
	DC	X'1F'	CIRCLE C		0
(1050)	DC	X'01'	LENGTH = 1		
	DC	X'76'	CIRCLE Y		1
	DC	X'01'	LENGTH = 1		
	DC	X'40'	CIRCLE N		2
TABLE2	DC	X'30'	LENGTH = 3		
	DC	X'1B'	FIGS		0
(83B3)	DC	X'05'	H	EOT	
	DC	X'1F'	LTRS		
	DC	X'01'	LENGTH = 1		
	DC	X'1F'	LTRS		2

Figure 3. Line control characters

For later use when constructing model CCW's, "value" codes are to be developed from the table of line control characters just written. There will be one value code for each control character or character sequence, and it will be used to locate the particular entry in the table. The values are calculated as $\frac{D-1}{2}$, where

D is the offset in bytes of the first character of the sequence from the start of the table. For example, the three-character sequence FIGS H LTRS in the 83B3 table of Figure 3 has a value of 0, and the one-character sequence LTRS has a value of 2. Write the values for the control character sequence in the table alongside the table entries for later reference. Be sure that each value is an integral number, that is, that each sequence offset (D) is odd. Pad the table with zeros if necessary.

2. Write out detailed command sequences. Now express the line control procedure as a sequence of channel commands. It will help to study the CCW sequences in Figure 4 for an IBM 1050 terminal. Be sure to understand the reason for each item specified (refer to the BTAM program logic manual). Be sure to specify the following items:

   a. Command Code — the actual operation to be performed, for example, read, write, prepare

   b. Address — the location into/from which characters are read/written

   c. Flag — data chain, command chain, suppress incorrect length indicator, skip, program-controlled interrupt (as needed)

   d. TP Op Type — poll-restart is used with the read response command of Read Initial if the next terminal on the line is to be polled after a negative poll (see BLD, Section M.2.0.0)
      — multiaddressing is used with the read response command of Write Initial if more than one component or terminal is to be addressed (see BLD, Section M.3.0.0)
      — normal (all other operations)

   e. Count — the number of characters involved in the read/write

3. Build model CCW's from the verbal command sequences just written and the character table. Figure 5 shows the fields of a model CCW. There is to be one model CCW constructed for each actual CCW from the previous section. A more detailed format description of a model CCW can be found in Figure 11 in the BLD section on Device I/O Modules.

   a. Put the command code in the first byte:
      01 = Write
      02 = Read
      03 = Prepare

   b. Set needed flags in bits 8-12 — data chain, command chain, suppress length indication, skip, program-controlled interrupt

	Operation	Address	Flag	TP Op	Count
Read Initial	Write (C)	Table	CC, SLI	0	1
	Write poll chars.	List	CC, SLI	0	2
	Read response	Area	CD	Poll- restart	2
	Read data	Area+2	SLI	0	Length-2
Write Initial	Write (C)	Table	CC, SLI	0	1
	Write addr. chars.	List	CC, SLI	0	2
	Read response*	RESPN		Multiaddr.	1
	Write data	Area	CC, SLI	0	Length
	Read answer	RESPN+1		0	1
(with reset)	Write (C)	Table	SLI	0	1
Read Continue	Write (y)	Table	CC, SLI	0	1
	Read data	Area	SLI	0	Length
Write Continue	Write data	Area	CC, SLI	0	Length
	Read answer	RESPN+1		0	1
(with reset)	Write (C)	Table	SLI	0	1
Read Repeat	Write (n)	Table	CC, SLI	0	1
	Read data	Area	SLI	0	Length

*If multiaddressing is not desired, set CC flag on and TP Op Type to Zero

Figure 4. IBM 1050 line control command sequences

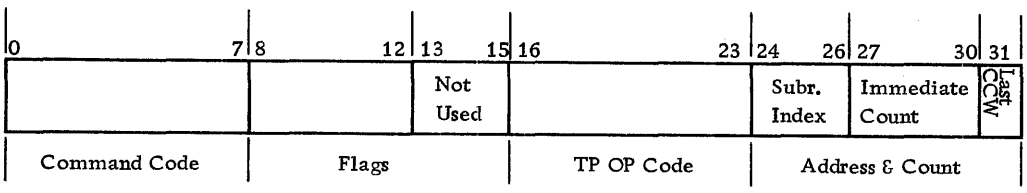


Figure 5. Model CCW fields

c. Insert TP Op Type in the third byte:
   00 = Normal no-restart
   04 = Poll-restart
   08 = Multiaddressing

d. Set subroutine index in bits 24-26. This is an index that will be used to identify the read-write subroutine that is appropriate for building the actual CCW from the current model CCW.

Index	Subroutine	Purpose
0	RTNE1	Read to / Write from a storage area (buffer) from / to a communications line
1	RTNE2	Write characters from a polling / an addressing list to a line
2	RTNE3	Read addressing / LRC response into RESPN area
3	RTNE4	Write special characters

e. Set immediate count in bits 27-30. The meaning of the immediate count field is dependent on the read-write subroutine specified in the subroutine index field.

Subroutine	Command Sequence	Immediate Count Value
RTNE1	Write data	0 - length will be gotten from DECB
	Read response	One character longer than a negative response
	Read data	0 - handled by routine
RTNE2		Number of polling/addressing characters
RTNE3		1 - if addressing response 5 - if LRC answer
RTNE4		Character table index "value" from step 1

f. Last CCW bit 31 — 1 in last model CCW of model channel programs, 0 in all others.

Upon completion of steps a to f, a sequence of model CCW's will have been specified in hexadecimal notation as shown in Figure 6.

Model CCW Content	Hexadecimal Specification
1. Write; CC, SLI; no op; RTNE4, (C), not last	01600060
2. Write; CC, SLI; no op; RTNE2, 2 char, not last	01600024
3. Read; CD; poll-restart; RTNE1, 2 char, not last	02800404
4. Read; SLI; no op; RTNE1, no count needed, last	02200001

Figure 6. Example of IBM 1050 Read Initial model CCW sequence

Actual coding in assembler language of the hexadecimal model CCW sequence should be as shown in Figure 7 to allow for comments.

Example: 1050 Read Initial

Name	Operation	Operand	Remarks
RDINIT1	DC	X'01'	WRITE CIRCLE C
	DC	X'60'	CC, SLI
	DC	X'00'	TP NO-OP
	DC	X'60'	CIRCLE C
RDINIT2	DC	X'01'	WRITE POLL CHARS
	.		
	.		
	.		
	.		
RDINIT4	DC	X'02'	READ DATA
	DC	X'20'	SLI
	DC	X'00'	TP NO-OP
	DC	X'01'	LAST

Figure 7. Coding on IBM 1050 Read Initial model CCW sequence

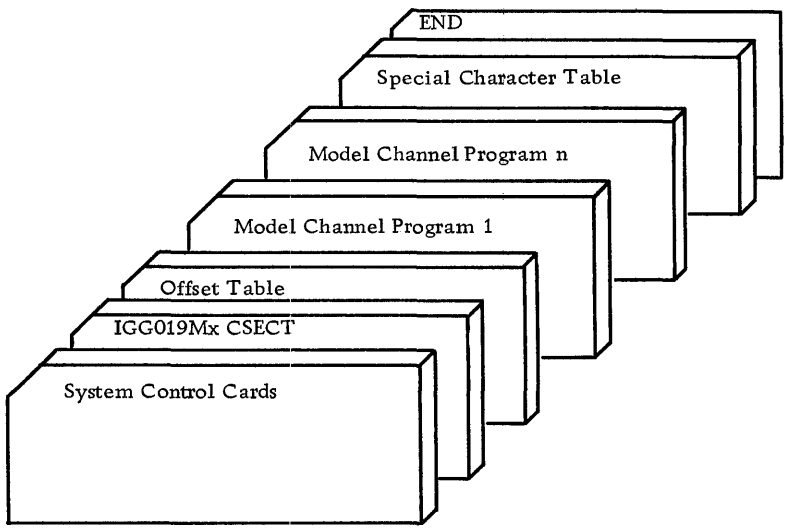
4. Form the model channel programs and character table into a Device I/O Module preceded by an offset table. The offset table is twelve bytes long and contains information needed to compute the addresses of the module's segments. Place the model channel programs in the desired order, followed by the special character table prepared in item 1 above. Determine the offsets (bytes, in hexadecimal) of the starting locations of each model channel program, relative to the first byte of the first model channel program. (See Figures 12-14 of the BLD section on Device I/O Modules.) Construct the offset table as shown in Figure 8, where the hexadecimal number in each DC instruction is the offset of the model channel program named in the comments field. Place FF in all unused offsets to indicate invalid specification.

```
DC X'xx'        Write Space
DC X'yy'        Read Initial
DC X'ww'        Write Initial
DC X'ss'        Read Continue
DC X'tt'        Write Continue
DC X'uu'        Read Conversational
DC X'vv'        Write Conversational
DC X'pp'        Read Repeat
DC X'FF'        Reserved
DC X'FF'        Reserved
DC X'FF'        Reserved
DC X'cc'        Special Character Table
```

Figure 8. Coding an offset table

All components of the Device I/O Module have now been written. Assign the module a unique symbolic identification. (Currently assigned identifications range from IGG019MD to IGG019MR — see Figure 9.)

Stack the card deck as shown below and the Device I/O Module is complete.



CODING CHANGES

Once all of the Device I/O Modules have been acquired for the terminals to be supported (those already supported plus new ones to be included), all that remains is to set up the tables and instructions that will inform BTAM and IOS of the presence and use of the new modules.

Remember that at system generation time a unit control block was constructed for each communication line. One word at relative location 16 in the UCB was filled with information on the physical and logical makeup of the terminals on the line. This word, the "device type word", is fully described in "UCB Device Codes" in Appendix H. During execution of the Device Type Analysis section of BTAM Open Module 1, five of its fields are examined and the code that was previously assigned to the terminal device and its options is chosen from the Terminal Code Table. This device code is used in determining the size of the IOB and in selecting the proper Device I/O Module.

The following paragraphs will discuss the section of coding that analyzes the device type word and the tables of IOB sizes and module identifications. Following this, a procedure will be detailed that shows how to write the device analysis and define the tables needed to support any group of terminal types.

Device Type Analysis

Analysis of the device word involves the testing of five of its fields and selecting a code from the Terminal Code Table. If any of the tests should fail or discover that an unsupported or illegal terminal has been specified, the task will be terminated through an abnormal end exit (SVC 13). An error code will be returned in Register 1 (see Appendix K). The following list describes the contents of each field (see Appendix H for details).

1. Device Class — This byte must contain the hexadecimal value 40, which indicates a communications device.
2. Control Unit Type — These four bits must be the hexadecimal numbers 1 or 2, representing the currently supported 2702 and 2701 control units respectively.
3. Adapter Type — These four bits identify the terminal adapter type assigned at system generation time.
4. Model Type — These four bits identify the model within its adapter class, for example, tests 3 and 4 identify the specific terminal in use.
5. Optional Features — These four bits further describe the selected terminal according to its optional features.

IOB Size Table

The Input/Output Block (IOB) provides the interface between BTAM and the I/O Supervisor (IOS). The Basic IOB, an area with a fixed size of 40 bytes, contains flags, pointers, addresses, and sense and status information (see Appendix D). Appended to this is an area of variable size in which the I/O channel programs are constructed. The size of this channel program area depends on the terminal and options selected. Its size in doublewords is the maximum number of channel command words in any one operation. For example, the longest operation for the regular 1050, write initial with reset, contains six CCW's; therefore, the size of the regular 1050 IOB channel program area is six doublewords or 48 bytes.

The channel program area sizes for each device are collected in a table in BTAM Open Module 1, starting at location SZTABLE (BLD, Section J.9.2.0). The device code generated by the device analysis section of Open Module 1 is used as an index on this list.

Module ID Table

During execution of BTAM Open Module 2, the needed Device I/O Modules are loaded into core storage. All of the modules are labeled IGG019xx, where xx represents two unique identifying characters. These characters are collected into a table in BTAM Open Module 2 (CHNIDTBL - BLD, Section K.7.7.0), and are used to access the desired module, being indexed by the device code.

Terminal Code Table

This table contains the hexadecimal integer codes assigned to every terminal/option combination. The device analysis section of BTAM Open Module 1 examines the unit control block for a line and decides which code is required. This code is stored in the data control block and is used to index the IOB Size Table and the Module ID Table later in the Open process.

Modification Procedure

By now a Device I/O Module is available or has been written for every terminal device to be supported. Set down a list of each terminal to be included in the system in some order, say, by terminal number, grouped by adapter type as shown in Figure 9.

1. List all terminal/option combinations. It would be wise to now include those contemplated, but not yet supported, to make future modifications more orderly (column 1 of Figure 9).

2. Examine every Device I/O Module and determine the number of model CCW's in the longest channel program of each one. Multiply these numbers by 8 and enter the answers beside the corresponding terminals (column 2).

3. All of the modules are labeled IGG019xx. List the last two characters of each ID beside the corresponding terminals (column 3).

4. Assign a hexadecimal integer between 00 and FF to each terminal/option combination (column 4). This code will be used as the terminal ID until the proper Device I/O Module is loaded by Open Module 2 at execution time.

5. List the adapter type and model codes for each unit (columns 5 and 6 respectively). These codes are arbitrarily chosen and assigned at system generation time. Refer to Appendix H for currently supported code assignments.

6. To construct the IOB Size Table in Open Module 1 (BLD, Section J.9.2.0), repeat column 2 as a series of DC HL1'no' statements, in the same order, where no is a decimal number from column 2. Label the first one SZTABLE.

7. To construct the Module ID Table in Open Module 2 (BLD, Section K.7.7.0), repeat column 3 as a series of DC C'xx' statements, in the same order, where xx is a two-character sequence from column 3. Insert a halfword zero for blanks. Follow each entry with a DC XL4'0' statement. These areas will later be loaded with the disk addresses and lengths of the module. Label the first entry CHNIDTBL.

8. To construct the Terminal Code Table in Open Module 1 (BLD, Section J.9.3.0), set down the entries of column 4 as a series of DC X'no' statements. Assign a label to the first entry for each adapter class.

9. Code a sequence of assembler language instructions that will make all five device type and option tests on the device word field of a UCB and access the proper entry in the table built by step 8.

Step 9 is a highly individualized sequence of coding; therefore, no general procedure can be specified for its writing. Refer to the original program (BLD, Section J.4.0.0) and to the sample shown in Appendix L for hints and techniques.

EXAMPLE: (Original BTAM support)

Column	1	2	3	4	5	6
	1050R	48	MD	00	1,3	1
	1050AP	32	ME	01	1,3	1
	1050A	48	MF	02	1,3	1
	1050AA	56	MG	03	1,3	1
	1050AC	64	MH	04	1,3	1
	1060R	48	MI	05	1	2
	1030R	40	MJ	06	2	1
	1030AP	16	MK	07	2	1
	83B3R	48	ML	08	4	1
	83B3TT	24	MM	09	4	1
	115AR	40	MN	0A	4	2
	115ATT	72	MO	0B	4	2
	TWXA	80	MP	0C	5	1
	TWXAA	88	MQ	0D	5	1
	TWXAC	16	MR	0E	5	1

R - regular
AP - auto poll
AA - auto answer
AC - auto call
A - both AA and AC
TT - terminal-to-terminal

Figure 9. Example of list of BTAM-supported terminals

POSSIBLE MODIFICATIONS

At a later date this section will discuss some possible modifications -- those that do not apply to all terminals.

BTAM LISTING DESCRIPTION (BLD)

This portion of the manual provides a detailed description of the BTAM programs to allow easy reference to their parts. The three sections of the BTAM Listing Description (BLD) cover BTAM's macro instructions, program modules, and device I/O modules.

Section A provides information on the system and special purpose macro instructions used by BTAM — purpose, format, parameters, and function. In the format section, items within brackets are optional; braces indicate that one of the enclosed items is to be chosen.

Section B contains a functional listing of the BTAM routines. It is a running narrative designed to be used with an assembly listing. The functional listing follows the program listing exactly, not necessarily the logic flow through the program. Each functional step has been assigned an index code that should not change appreciably with minor program changes. Where possible, the functions are keyed to symbolic locations in the program. As the program stabilizes toward official release time, more symbolic names may be included to facilitate working with the listings. Each step is also referenced to the control block field it affects. This allows the reader to quickly scan the BLD and find where a certain field was filled in and relate it to the program listing.

Section C contains a description and examples of the BTAM Device I/O Modules, which are device-dependent models of I/O channel programs.

SECTION A: MACRO INSTRUCTION EXPANSIONS

READ Macro Instruction

Purpose:

The READ macro instruction causes contact to be established or maintained with a terminal, and a message segment to be received by the CPU. The parameters of the macro specify the operation type, terminal, and read-in area.

Format:

Name	Operation	Operands
[name]	READ	decbname,{TI / TT / TV / TP}, dcbname, {area / 'S'}, {length / 'S'}, {termlist / 'S'}, rln [,MF={L / E}]

name — assigned to first location of expanded coding
decbname — symbolic address of DECB for this operation; also the ECB in which completion is to be posted
type — choice of

	Type codes set in DECB
TI (Read initial)	1
TT (Read continue)	3
TV (Read conversational)	5
TP (Read repeat)	7

dcbname — symbolic address of data control block for the line
area — symbolic address of input area
'S' — BTAM will provide buffer
length — number of bytes in input area
'S' — buffer length specified in DCB
termlist — symbolic address of next polling list entry
'S' — will repoll last terminal that sent message. This should only be used with wraparound polling lists.
rln — relative line number within line group
MF=L — creates parameter list, does not execute
MF=E — updates and uses previously defined parameter list, executes
MF blank — creates parameter list and executes
Note: Register form can be used with the MF blank forms for all macro parameters except decbname and type, and with MF=E for all but type.

Example: READ (1), TI, DCBNAME, (3), 100, (12), (13), MF=E where before execution the desired DECB address is put in Register 1, read-in area address is put in Register 3, terminal list address in Register 12, and relative line number on Register 13.

Function:

Assembly time. The MF=L form of the instruction generates a data event control block which provides communication with IOS at execution time. The coding generated inline is a series of assembler define constant (DC) instructions, which become the DECB (Appendix F). Figure 10 shows the block's format and placement of the macro operands. The parameter decbname is assigned to the first location of the block. The DECSDECB field is reserved for IOS use as the ECB in which completion of the operation will be posted. If the read operation type were specified with reset (TxR)* or inhibit (TxH), bit 0 or bit 1, respectively, of DECTYPE +1 will be set to one. The DECIOBPT field will be filled in during execution. DECRESPN is not used for reading. The MF blank form generates the same DECB as above, and adds instructions to branch-and-link to the BTAM Read-Write routine. The MF=E form does not generate a DECB. However, if any parameters are included in the macro statement, load and store instructions will be generated to update those parameters in a previously defined DECB. This form also produces the branch-and-link instructions.

*x=I, T, V, or P

Execution time. The MF=L form produced no executable instructions. The MF blank form will transfer control and pass its parameters in the DECB. The MF=E form will update some parameters in a DECB (defined in a previous READ macro of the MF=L form) and transfer control. If 'S' options were specified, appropriate BTAM routines will provide the parameters when needed.

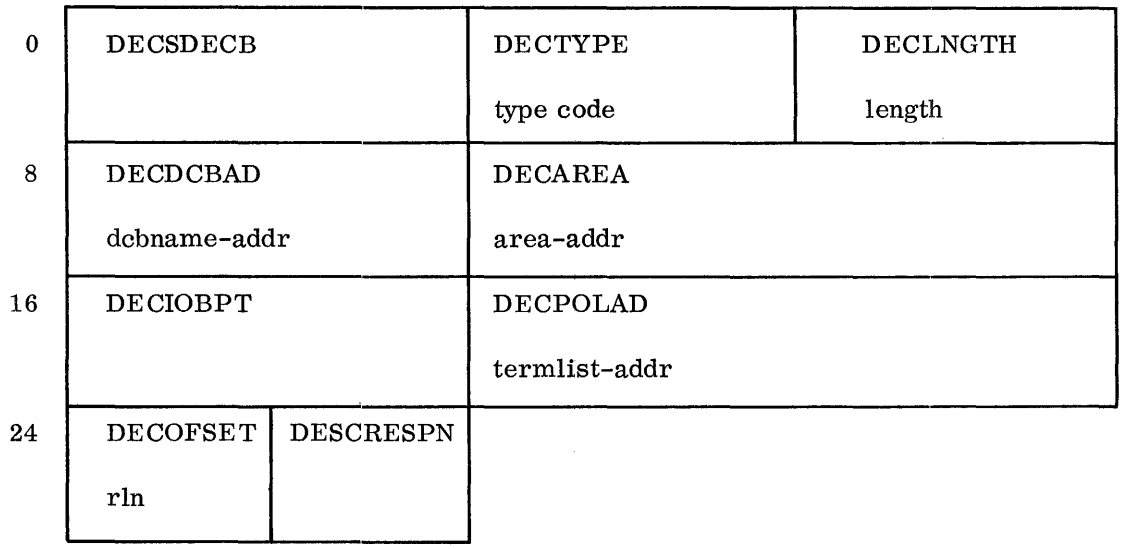


Figure 10. Line DECB

WRITE Macro Instruction

Purpose:

The WRITE macro instruction causes contact to be established or maintained with a terminal, and a message segment to be sent from the CPU to the terminal. The parameters of the macro specify the operation, type, terminal, and output buffer.

Format:

Name	Operation	Operands
[name]	WRITE	decbname, {TI / TT / TV / TB}, dcbname, area, length, termlist, rln [, MF= {L / E}]

name — assigned to the first location of expanded coding
decbname — address of data event control block for this WRITE

type —

	Type codes set in DECB
TI (Write initial)	2
TT (Write continue)	4
TV (Write conversational)	6
TB (break - write SPACE signal)	0

dcbname — address of data control block for line specified
area — address of output area, or of first buffer of a chain (dynamic buffering)
length — length of output area
termlist — address of next addressing list entry
rln — relative line number within line group
MF=L — creates parameter list, does not execute
MF=E — updates and uses previously defined parameter list, executes
MF blank — creates parameter list and executes

Function:

The WRITE macro operates in the same fashion as READ. It produces a DECB that is identical except for the type code. If the write operation type were specified with reset (TxR)* bit 0 of DECTYPE +1 will be set to 1. During Write operations, responses to addressing will be stored in the first byte of the halfword DECRESPN field; answers to LRC checks will be placed in the second byte.

OPEN Macro Instruction

Purpose:

The OPEN macro instruction causes communication line groups to be prepared for use. Its parameters are the addresses of DCB's to be opened and codes indicating file type and disposition.

Format:

Name	Operation	Operands
[name]	OPEN	$(dcb_1, , dcb_2, \ldots, dcb_n,) \left[, MF = \left\{ \begin{matrix} L \\ (E, \text{ listname}) \end{matrix} \right\} \right]$

name — assigned to first location of expanded coding or the list itself
$dcb_1, \ldots, dcb_n$ — addresses of all DCB's to be OPENed by this statement; may be in symbolic or register form, for example, DCB_1 or (DCBREG), where the user has previously loaded DCBREG with the DCB's address
MF=L — The generated coding creates a parameter list as specified, but does not execute the OPEN. The name in the name field of the OPEN is assigned to the parameter list.
MF=(E, listname) — The generated instructions will execute the OPEN for all parameters in the list specified by listname.
MF blank — The OPEN will be executed, using the parameters specified in the macro instruction.

*x = I, T, or V

Function:

Assembly. The MF=L form of the macro expansion creates a list of DCB addresses and option codes for System Open use. The parameter name is assigned to this list. The MF blank form generates the list and a supervisor call (SVC) instruction for System Open. The MF=(E, listname) form produces instructions to reference and modify an existing list and issue an SVC.

Execution. The MF=L form created no executable instructions. The other two forms transfer control to System Open through the SVC instruction. When the System Open routine finishes, control is passed to the BTAM Open Modules (described later in the BLD), which allocate storage for and/or initialize the various control blocks and load the BTAM routines and appendages.

CLOSE Macro Instruction

Purpose:

The CLOSE macro instruction causes communications line groups to be removed from use. Its parameters are the addresses of the DCB's to be closed.

Format:

Name	Operation	Operands
[name]	CLOSE	$(dcb_1,,dcb_2,,\ldots,,dcb_n)$ [{ MF=L / MF=(E, listname) }]

name — assigned to first location of expanded coding or the list itself if MF=L
$dcb_1 \ldots dcb_n$ — addresses of DCB's to be released (Note double commas — no options specified)
MF=L — creates parameter list called name, no execution
MF=(E, listname) — executes for parameter list at location listname
MF blank — creates parameter list and executes

Function:

Assembly time. The MF=L form creates a list of DCB addresses. The MF blank form generates the list and a supervisor call (SVC) to System Close. The MF=(E, listname) generates the SVC and instructions to pass listname as a parameter to System Close.

Execution time. The MF=L form created no executable instructions. The other two transfer control of the System Close routine. System Close will complete its work and pass control to the BTAM Close Module (to be described later).

DFTRMLST Macro Instruction

Purpose:

The DFTRMLST macro instruction creates a terminal list in the user program.

Format:

Name	Operation	Operands
entry	DFTRMLST	$\left\{\begin{array}{l}\left\{\begin{array}{l}\text{OPENLST}\\\text{WRAPLST}\end{array}\right\}, (\text{comp}_1, \text{comp}_2, \ldots, \text{comp}_n)\\ \text{DIALST}, \left\{\begin{array}{l}\text{nodig, dialdigs, } (\text{comp}_1, \text{comp}_2, \ldots, \text{comp}_n)\\ \text{0, } (\text{comp}_1, \text{comp}_2, \ldots, \text{comp}_n)\end{array}\right\}\\ \text{IDLST}, \left\{\begin{array}{l}\text{nodig, dialdigs, nochar, termid}\\ \text{0, nochar, termid}\end{array}\right\}\end{array}\right\}$

entry — symbolic name assigned to the list
OPENLST — indicates open list structure
WRAPLST — indicates wraparound list structure
DIALST — indicates dial or answer list where terminal ID need not be verified. (0 indicates answer list.)
IDLST — indicates dial or answer list where terminal ID must be verified, as TWX. (0 indicates answer list.)
comp_1 — polling/addressing characters (hexadecimal) in the terminal code's internal S/360 byte representation, for example, E215 (A0 for 1050)
nodig — number of dial digits (hexadecimal), for example, B. (0 indicates answering list.)
dialdigs — the dial digits, for example, 12173523818
nochar — number of terminal ID characters (hexadecimal), for example, C
termid — terminal ID, for example, CHICAGO=CH10

Function:

Assembly — creates within the user program the desired terminal list (see C28-6553 for the format of the created list)
Execution — no executable instructions

DCB Macro Instruction

Purpose:

The DCB macro instruction allocates storage within the user program for a data control block, describing the data set to the operating system.

Format:

Name	Operation	Operands
dcbname	DCB	keyword parameters

dcbname — symbolic name assigned to data control block
keyword parameters — up to 52 parameters that define all aspects of a data set. (These are explained in C28-6553 and C28-6541.)

Function:

Assembly — allocates storage and initializes values for those parameters the user has specified. Note: Some parameters have "assumed" values, that is, a certain value will be supplied by the macro expansion unless overridden.

Execution — no executable instructions

RESETPL Macro Instruction

Purpose:

The RESETPL macro instruction interrupts polling on a direct connection line following a Read Initial macro instruction or cancels an ENABLE command issued to a switched connection line as the result of a READ or WRITE operation with an "answer" type list.

Format:

Operation	Operands
RESETPL	$\left\{\begin{matrix}\text{decbname}\\(X)\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{POLLING}\\\text{ANSRING}\end{matrix}\right\}\right]$

decbname — name of the DECB in operation
X — register containing the address of the DECB involved
POLLING — indicates only the channel commands for the direct connection case need be included.
ANSRING — indicates only the channel commands for the switched connection case need be included.
(If neither POLLING nor ANSRING is specified, both will be provided for.)

Function:

Assembly — generates executable inline machine instructions.

Execution — Direct connection line. If a polling operation is currently in progress, and if it elicits a negative response, the polling list pointer will be incremented in the IOB (IOB POLPT), polling will be terminated, and the operation posted complete. If the polling operation elicits a positive response or a timeout, the polling list pointer will not be incremented, and the operation will proceed to its normal conclusion (normal conclusion for a timeout is to post it complete with error). If an operation other than polling is currently in progress (message reception, message transmission, addressing) it will proceed unaffected.

Switched connection line. If an ENABLE command is outstanding (a terminal has not dialed the computer since the ENABLE command was issued) a HALT I/O instruction will be issued to the enabled line. If the ENABLE command has completed (a call has been received from the terminal), the operation will proceed to its normal completion. A poll in progress will not be interrupted.

CHGNTRY Macro Instruction

Purpose:

The CHGNTRY macro instruction deletes or reactivates a terminal entry in a polling or addressing list without redefining the list, by manipulating the skip bit in the control byte of the entry.

Format:

Operation	Operands
CHGNTRY	{ entry, / (r) } type, { listntry, / (r) } { numchars, / (r) } { SKIP / ACTIVATE }

r — general register
entry — address of the beginning of the terminal list
type — list structure: OPENLST, WRAPLST, DIALST, IDLST
listntry — relative position of entry to be changed in the list
numchars — number of polling/addressing characters in each entry
SKIP — indicates to turn on the skip bit in the entry
ACTIVATE— indicates to turn off the skip bit

Function:

Assembly — generates executable machine instructions

Execution — terminal list skip bits are set on or off

SECTION B: PROGRAM MODULES

MODULE NAME BTAM OPEN (MODULE 1)

MODULE IDENTIFICATION IGG0193M

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	J.0.0.0.	DCB Dummy Section (Appendix B)	located at rear of module
	J.0.1.0.	IOB Dummy Section (Appendix D)	
	J.0.2.0.	DEB Dummy Section (Appendix C)	
	J.1.0.0.	Create and Initialize DEB (Partial)	
DDBLKLP	J.1.1.0.	Determine the size of the DEB from the number of UCB addresses in the task I/O table (TIOT) (Appendix I). There is one UCB (Appendix G) for each extent (communication line).	
	J.1.2.0.	Obtain core storage for the DEB via the GETMAIN macro.	DEB
CLEARL EXMODIFY	J.1.2.1.	Clear the DEB to zero.	
	J.1.3.0.	Fill in the DEB field (NMEXT) with the number of extents for this DEB.	DEBNMEXT
	J.1.4.0.	Fill in the DEB field (TCBAD) with the address of the task control block (TCB) for the current task.	DEBTCBAD
INITDEB1	J.1.5.0.	Fill in the DEB field (APPAD) with the address of the DEB. This is actually the address of the first byte of the appendage table attached to the front of the DEB.	DEBAPPAD
	J.1.6.0.	Fill in the DEB field (EXSCL) with 02 to indicate that this DEB is for non-direct access devices.	DEBEXSCL
	J.1.7.0.	Fill in the DEB field (DEBAD) with the address of the first DEB in the chain of DEBs.	DEBDEBAD

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	J.2.0.0.	Initialize DCB (Partial)	
	J.2.1.0.	Fill in the TCB field (DEB) with the address of the start of the basic section of the DEB. This is actually the address of the DEB field (NMSUB).	TCBDEB
	J.2.1.1.	Fill in the DCB field (DEBAD) with this same address.	DCBDEBAD
	J.2.2.0.	Fill in the DCB field (IFLGS) with 0C to indicate that the Input/Output Supervisor (IOS) error routine is not to be used.	DCBIFLGS
	J.3.0.0.	Create and Initialize DEB (Partial)	
	J.3.1.0.	Fill in the DEB field (DCBAD) with the address of the DCB.	DEBDCBAD
	J.3.2.0.	Fill in the DEB field (DEBID) with hexadecimal F to identify this block as a DEB to IOS.	DEBDEBID
	J.3.3.0.	Zero out the DEB field (PROTG) that will later contain the protection tag for this task.	DEBPROTG
	J.3.4.0.	Fill in the DEB field (LNGTH) with the length in doublewords of the DEB.	DEBLNGTH
DEBMOVE1	J.3.5.0.	Fill in the DEB field (UCBAD) with a table of addresses of the UCB's. There is one UCB for each communication line. One fullword is used for each address.	DEBUCBAD
	J.3.6.0.	Fill in each DEB appendage table field (EOEA, SIOA, PCIA, CEA, XCEA) initially with the "normal return" address for IOS, which would cause immediate returns from all appendage exits. When the appendages are actually loaded their addresses will replace the normal return address.	DEBEOEA DEBSIOA DEBPCIA DEBCEA DEBXCEA

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	J.4.0.0.	Device Type Analysis	
DEVTYANL	J.4.1.0.	Test for device class 40 (telecommunications) in byte 18 of the UCB (only one UCB will be analyzed since all UCB's of a given communication line group are the same). If the device class is not 40, exit via SVC 13 as an abnormal end of task with error code 00090000 in Register 1.	
	J.4.2.0.	Test for control unit types 1 or 2 (2701/2702) in the last four bits of byte 19 of the UCB. If types 1 (2702) or 2 (2701) are not found, exit via SVC 13 as an abnormal end of task with error code 00091000 in Register 1.	
	J.4.3.0.	Determine the device type code from the adapter code, model code, and optional features specified in bytes 16-19 of the UCB. If any invalid codes are detected, exit via SVC 13 with Register 1 error codes as follows: 00092000 — adapter code invalid 00093000 — device code invalid 00094000 — option code invalid	
IOBST1	J.5.0.0.	Initialize DCB (Partial)	
	J.5.1.0.	Determine the size of an IOB using the device type code to index the size table and store the number in the DCB field (EIOBX).	DCBEIOBX
	J.6.0.0.	Create and Initialize IOB (Partial)	
	J.6.1.0.	Determine the amount of core storage needed for all IOB's, one IOB per extent (line).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	J.6.2.0.	Obtain the core storage needed via the GETMAIN macro.	IOB
	J.7.0.0.	Initialize DCB (Partial)	
	J.7.1.0.	Fill in the DCB field (IOBAD) with the address of the first IOB less the length of an IOB.	DCBIOBAD
	J.7.2.0.	Fill in the DCB field (DEVTP) with the device type code determined in section J.4.3.0.	DCBDEVTP
	J.8.0.0.	Create and Initialize IOB (Partial)	
STARTLP	J.8.1.0.	Clear enough core storage for one IOB.	
	J.8.2.0.	Fill in byte 4 (5th byte) of the IOB field (CSW) with 0F to indicate channel program not busy.	IOBCSW
	J.8.3.0.	Set bits 0, 1, 6 in the IOB field (FLAG1) on to indicate command chaining, data chaining, and not-FIFO I/O requests.	IOBFLAG1
	J.8.4.0.	Fill in the IOB field (START) with the address of the first CCW of the channel program located at the end of the IOB.	IOBSTART
	J.8.5.0.	Fill in the IOB field (UCBX) with the relative line number to be used as an index to the UCB addresses in the DEB.	IOBUCBX
	J.8.6.0.	Fill in the IOB field (DCBPT) with the address of the associated DCB.	IOBDCBPT
	J.8.7.0.	Fill in the IOB field (WGHT) with 00 to indicate there is no channel loading factor specified for this IOB.	IOBWGHT
	J.8.8.0.	Repeat from section J.8.0.0. for all IOBs. There will be one IOB per extent (line).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	J.9.0.0.	Determine other line groups to be opened	
RELOOP	J.9.1.0.	Repeat from section J.1.0.0. for all communication line group data sets. The existence of these data sets is determined from the "where to go" table (WTG).	
XCTLRTNE	J.9.1.1.	When all data sets have been processed by this module, transfer control to BTAM Open Module 2 (IGG0193Q) via the XCTL macro instruction.	
SZTABLE	J.9.2.0.	A table of channel program sizes.	
CDIBM1	J.9.3.0.	A table of terminal/option device codes.	

MODULE NAME BTAM OPEN (MODULE 2)

MODULE IDENTIFICATION IGG0193Q

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	K.0.0.0.	DCB Dummy Section (Appendix B)	located at rear of module 2
	K.0.1.0.	IOB Dummy Section (Appendix D)	
	K.0.2.0.	DEB Dummy Section (Appendix C)	
	K.1.0.0.	Initialize Communication Line IOB's	
SADSTART	K.1.1.0.	Examine the control unit code in the UCB device type word. If a 2702 is specified, skip to SADRTNE (K.1.3.0.). If 2701, skip to ENABCTL (K.1.2.0.). If neither, exit via SVC 13 as an abnormal end of task with error code 00095000 in Register 1.	
ENABCTL	K.1.2.0.	If either auto-call or auto-answer is specified as an optional feature, skip to VDTBLST (K.2.0.0.).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	K.1.2.1.	Set ENABLE command in the channel program area, IOB field (CPA), and skip to EXCPSAD (K.1.3.3.).	IOBCPA
SADRTNE	K.1.3.0.	Get the number of extents (IOB's) from DEB field (NMEXT) and determine SAD type needed.	
SADLP	K.1.3.1.	Insert the SAD code in the IOB field (CPA). If either auto-call or auto-answer is specified, skip to EXCPSAD (K.1.3.3.).	IOBCPA
	K.1.3.2.	Insert the ENABLE command in the second CCW field of the IOB channel program area.	IOBCPA+8
EXCPSAD	K.1.3.3.	Fill in the IOB field (ECBPT) with the address of the ECB (Appendix E) associated with the SAD/ENABLE I/O request about to be issued.	IOBECBPT
	K.1.4.0.	Issue I/O request with the EXCP macro.	
	K.1.5.0.	Check for errors in the channel program execution of SAD/ENABLE and retry up to two times if necessary.	
SADLPCHK	K.1.6.0.	Repeat from section K.1.3.1. for all communication lines (IOB's).	
VDTBLST	K.2.0.0.	Load Read-Write Module IGG019MA	
	K.2.1.0.	Supply ID and relative track and record address (TTR) of Read-Write module to the LOADROUT routine (K.7.0.0.).	
	K.2.2.0.	Supply address of DEB field (SUBID) to the LOADROUT routine.	DEBSUBID
	K.2.3.0.	Load the Read-Write module via the LOADROUT routine (K.7.0.0.).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	K.2.4.0.	Fill in the DCB field (READ) with the address of the Read-Write module.	DCBREAD
	K.3.0.0.	Load Device I/O Module IGG019xx	
RWVD1	K.3.1.0.	Supply the ID and TTR of the desired device I/O module, as found by indexing CHNIDTBL with the contents of the DEB field (DEVTP), to the LOADROUT routine (K.7.0.0.).	
	K.3.2.0.	Load the desired device I/O module via the LOADROUT routine, if it is not already in core storage (K.7.0.0.).	
	K.4.0.0.	Build Device I/O Directory	
	K.4.1.0.	Fill in the first byte of the device I/O directory entry with the contents of the DCB field (DEVTP). The device I/O directory is located within the Read-Write routine, 12 bytes from the beginning (Appendix J).	
	K.4.2.0	Fill in the last 3 bytes of the device I/O directory entry with the address of the Device I/O Module.	
	K.4.3.0.	Change the DCB field (DEVTP) to contain the offset of this device I/O directory entry from the start of the directory.	DCBDEVTP
	K.5.0.0.	Load Channel End Appendage IGG019MB	
	K.5.1.0	Supply the ID of the channel end appendage module to the LOADROUT routine (K.7.0.0.).	
	K.5.2.0.	Load the channel end appendage via the LOADROUT routine (K.7.0.0.).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	K.5.3.0.	Fill in the DEB field (CEA) with the address of the channel end appendage.	DEBCEA
	K.6.0.0.	Terminating Procedure	
RELOOP	K.6.1.0.	Examine the WTG table for any other line group DCB entries that will need Open Module 2. Repeat from section K.1.0.0. for additional line group entries.	
	K.6.2.0.	Fill in the DEB field (NMSUB) with the count of the number of subroutines loaded by Open Module 2.	DEBNMSUB
LOADROUT	K.7.0.0.	Loading Routine	
	K.7.1.0.	Use the ID supplied by the requesting routine to calculate the address of the module to be loaded.	
	K.7.2.0.	Load the desired module into the calculated address via the SVC LOAD macro.	
	K.7.3.0.	Return the address of the loaded module to the requesting routine.	
	K.7.4.0.	Fill in the DEB field (SUBID) with the ID of the loaded module (last two characters of the module identification label).	DEBSUBID
	K.7.5.0.	Advance by two bytes the pointer (RUCB) to the DEB field (SUBID).	
	K.7.6.0.	Increment the count of subroutines loaded in Open Module 2.	
	K.7.6.1.	Return to calling routine.	
CHNIDTBL	K.7.7.0.	A table of BTAM module identifiers.	

MODULE NAME READ-WRITE

MODULE IDENTIFICATION IGG019MA

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	L.0.0.0.	DECB Dummy Section (Appendix F)	
	L.0.1.0.	IOB Dummy Section (Appendix D)	
	L.0.2.0.	DCB Dummy Section (Appendix B)	
IODIRECT	L.0.3.0.	Device I/O Directory	
	L.1.0.0.	Initialization	
	L.1.1.0.	Determine the address of the IOB for the line about to be read or written, from the appropriate entry in the DCB field (IOBAD).	
	L.1.2.0.	Test byte 4 (5th byte) of the IOB field (CSW) for channel program busy. If busy, set busy code 0C in Register 15 in the register save area and skip to section L.7.2.0.	Register 15
	L.1.3.0.	Use DCB field (DEVTP) to identify the needed device I/O directory entry (see Appendix J).	
	L.1.4.0.	From the device I/O directory entry and the contents of the DECB field (TYPE) obtain the offset of the needed model channel program in the device I/O module and test the offset for validity (Appendix J).	
		If invalid, set return code 08 in Register 15 and skip to section L.7.2.0.	Register 15
	L.1.5.0.	Use the offset to calculate the address of the needed model channel program within the device I/O module (figures 12-14). Retain this address in VECTOREG.	VECTOREG

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	L.2.0.0.	Linkages	
	L.2.1.0.	Fill in the DECB field (IOBPT) with the address of the IOB.	DECIOBPT
	L.2.2.0.	Fill in the IOB field (START) with the address of the area in which the channel program is to be built. This area is actually the IOB field (CPA).	IOBSTART
	L.2.3.0.	Fill in the IOB field (ECBPT) with the address of the ECB for this operation.	IOBECBPT
	L.2.4.0.	Clear the register (ACCUMREG) that will later be used for addressing terminal list entries.	ACCUMREG
	L.3.0.0.	Core Storage Data Address Specification	
	L.3.1.0.	Test first byte of the DECB field (TYPE) for an 'S' type address specification. Skip to L.4.0.0. if an 'S' type address is not specified.	
	L.3.2.0.	Get a buffer via the GETBUF macro if an 'S' type address is specified and fill in the DECB field (AREA) with its address.	DECAREA
	L.4.0.0.	Length Specification	
A1	L.4.1.0.	Test first byte of DECB field (TYPE) for an 'S' type length specification. Skip to section L.5.0.0. if an 'S' type length is not specified.	
	L.4.2.0.	Fill in the DECB field (LNGTH) with the length specified in the DCB field (BUFL).	DEBLNGTH
	L.5.0.0.	Buffering Type Specification	
	L.5.1.0.	Test the DCB field (BFTEK) for the type of buffering to be used.	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
		If dynamic buffering is specified skip to A3 (L.7.5.0.).	
	L.5.2.0.	Set the adjustment in AJUSTREG initially to zero. The adjustment will be used for dynamically building the channel program.	AJUSTREG
	L.6.0.0.	Build Channel Program	
	L.6.1.0.	Zero out bits 36-39 of the IOB field (CSW) to indicate channel program busy.	IOBCSW
A5	L.6.2.0.	Fill in bytes 4 and 5 of the next CCW with bytes 1 and 2 of the next model CCW. These bytes are the flags and "TP Op Type" bytes respectively (see Figure 11).	
	L.6.3.0.	Clear to zero the DECB field (RESPN).	DECRESPN
	L.6.4.0.	From byte 3 of the model CCW derive the index of the appropriate Read-Write subroutine (in WORKREG1) and the 4-bit immediate count field (in WORKREG2).	WORKREG1 WORKREG2
◄—[LEAVE]—	L.6.5.0.	Branch-and-link to the indicated subroutine (RTNE1, RTNE2, RTNE3, or RTNE4) after providing the return point (L.6.6.0.) in register LINKREG.	LINKREG
—[RETURN]►	L.6.6.0.	Fill in bytes 6 and 7 of the CCW with the count (or length) returned from the subroutine in WORKREG2.	IOBCPA
	L.6.7.0.	Fill in bytes 1, 2, and 3 of the CCW with the address of the area to be read into or written from as returned from the subroutine in WORKREG1.	IOBCPA

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	L.6.8.0.	Fill in byte 0 of the CCW with the command code specified in byte 0 of the model CCW.	IOBCPA
	L.6.9.0.	Tests	
	L.6.9.1.	Test for an inhibit specification in both the DECB field (TYPE) and in byte 3 of the model CCW. Fill in byte 0 of the CCW with the inhibit command code if specified in both places.	IOBCPA
A9	L.6.9.2.	Test for the last model CCW in the model channel program. If the present model CCW is not the last, start construction of the next CCW at section L.6.2.0. If the present model CCW is the last, continue.	
	L.6.9.3.	Test for a dynamic buffering specification in the DCB field (BFTEK). If dynamic buffering is specified skip to A7 (L.7.4.0.).	
	L.6.9.4.	Test for a reset specification in the DECB field (TYPE). If there is no reset specified skip to L.7.1.0.	
	L.6.9.5.	Set the CC (chain command) and SLI (suppress length indication) flags in byte 4 of the CCW.	IOBCPA
	L.6.9.6.	Fill in byte 5 of the CCW with a "Reset" TP Op Type code.	IOBCPA
	L.6.9.7.	Construct a disable CCW as the last CCW of the channel program.	IOBCPA
	L.7.0.0.	Program Linkage	
A8 ◄[LEAVE]—	L.7.1.0.	Pass control to the Input/Output Supervisor (IOS) via an EXCP SVC 0.	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
—[RETURN]►	L.7.2.0.	Restore registers of user routine that were saved in the user-specified save area.	
	L.7.3.0.	Pass control to the user routine via RETRNREG (Register 14).	RETRNREG
A7	L.7.4.0.	Skip to A8 (L.7.1.0.). (The dynamic buffering option is not yet supported.)	
A3	L.7.5.0.	Set the adjustment in AJUSTREG to 16. Branch to A5 (L.6.2.0.).	AJUSTREG
	L.8.0.0.0.	Subroutines	
	L.8.1.0.0.	Subroutine 1 - Data Address Determination	
RTNE1	L.8.1.1.1.	Increment the address specified in the DECB field (AREA) by the current adjustment value in AJUSTREG and retain the result in WORKREG1.	WORKREG1
	L.8.1.1.2.	Test the count specified in WORKREG2 for zero. If the count is zero skip to RTNE1A.	
	L.8.1.1.3.	Increment the current adjustment value by the specified count.	AJUSTREG
	L.8.1.1.4.	Return to L.6.6.0. via LINKREG.	
RTNE1A	L.8.1.2.1.	Decrement the length value taken from the DECB field (LNGTH) by the current adjustment value in AJUSTREG and retain the result in WORKREG2.	WORKREG2
	L.8.1.2.2.	Return to L.6.6.0. via LINKREG.	
	L.8.2.0.0.	Subroutine 2 - Terminal List Manipulation	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
RTNE2	L.8.2.1.1.	Test for Write Initial command code in byte 1 of the DECB field (TYPE). If Write Initial is not specified skip to L.8.2.1.3.	
	L.8.2.1.2.	Change the base address of the IOB so that the IOB field (POLPT) actually becomes the IOB field (ADRPT) and points to the addressing list entry.	IOBPOLPT
	L.8.2.1.3.	Test the first byte of the DECB field (TYPE) for an 'S' type terminal list specification. If an 'S' specification is not made skip to RTNE2A (L.8.2.2.1.).	
	L.8.2.1.4.	Test for all zeros in the IOB field (POLPT). If all zeros skip to RTNE2A, since this is an initial entry in the terminal list.	
	L.8.2.1.5.	Fill in the DECB field (POLAD) with the address of the current terminal list found in the IOB field (POLPT).	DECPOLAD
RTNE2A	L.8.2.2.1.	Calculate the address of the current entry in the terminal list from the current contents of ACCUMREG and the contents of the DECB field (POLAD).	
	L.8.2.2.2.	Test the count found in WORKREG2 for zero. If the count is zero skip to RTNE2B (L.8.2.8.1.).	
RTNE2E	L.8.2.3.1.	Test for the skip bit on in the current entry of the terminal list. If the skip bit is not on, skip to RTNE2D (L.8.2.7.1.).	
	L.8.2.3.2.	Test for the end of list bit on in the current entry of the terminal list. If the current entry is not the end of the list skip to RTNE2G (L.8.2.4.1.).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	L.8.2.3.3.	Set LINKREG (Register 14) to return control to the user routine (L.7.2.0.) rather than the Read-Write routine (L.6.6.0.).	LINKREG
	L.8.2.3.4.	Test for the format bit on in the current entry of the terminal list. If the format bit is on skip to RTNE2C (a wraparound terminal list if indicated). If the format bit is not on skip to RTNE2F.	
RTNE2G	L.8.2.4.1.	Test for the format bit on in the current entry of the terminal list. If the format bit is on skip to RTNE2C (a wraparound terminal list is indicated).	
	L.8.2.4.2.	Increment the address of the current terminal list entry in WORKREG1 to the address of the next entry.	WORKREG1
	L.8.2.4.3.	Skip to RTNE2F.	
RTNE2C	L.8.2.5.1.	Calculate the address of the start of the terminal list from the value given at the end of the terminal list and retain the result in WORKREG1.	WORKREG1
RTNE2F	L.8.2.6.1.	Test LINKREG for the type of return — to the user routine (L.7.2.0.) or to the Read-Write routine (L.6.6.0.). If the latter is specified, return to RTNE2E (L.8.2.3.1.).	
	L.8.2.6.2.	Set return code 04 in Register 15 to indicate a skip and end of list (EOL) condition.	Register 15
	L.8.2.6.3.	Set byte 4 of the IOB field (CSW) to indicate channel program not busy.	IOBCSW
RTNE2D	L.8.2.7.1.	Update the IOB field (POLPT) to the current entry of the terminal list and retain the address in WORKREG2.	IOBPOLPT WORKREG2

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	L.8.2.7.2.	Restore the base address of the IOB.	
	L.8.2.7.3.	Return via LINKREG.	
RTNE2B	L.8.2.8.1.	Obtain count value from current terminal list (DIALST) entry and retain in WORKREG2.	WORKREG2
	L.8.2.8.2.	Increment the address of the current terminal list entry in WORKREG1 past the count field to the dial digits.	WORKREG1
	L.8.2.8.3.	Update ACCUMREG to reference the component address portion of the current terminal list entry.	ACCUMREG
	L.8.2.8.4.	Return to RTNE2D.	
	L.8.3.0.0.	Subroutine 3 - Response Field Maintenance	WORKREG2
RTNE3	L.8.3.1.1.	Separate out the immediate count portion of byte 3 of the model CCW and retain its value in WORKREG2.	WORKREG2
	L.8.3.1.2.	Calculate the address of the proper byte within the DECB field index (byte 3) of the model CCW and the address of the first byte of the DECB field (RESPN). Retain the address in WORKREG1.	WORKREG1
	L.8.3.1.3.	Test for a response field index of zero. If the response field index is not zero return via LINKREG (LRC Response).	
	L.8.3.1.4.	Test for the end of list bit on in the current entry of the terminal list. If the end of list bit is not on skip to L.8.3.1.6.	
	L.8.3.1.5.	Set the CC flag on in the CCW and return via LINKREG.	IOBCPA

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	L.8.3.1.6.	Test for the skip bit on in the next entry of the terminal list. If the skip bit is not on return via LINK-REG.	
	L.8.3.1.7.	Test for the EOL bit on in the next entry of the terminal list. If the EOL bit is not on skip to L.8.3.1.6. If the EOL bit is on skip to L.8.3.1.5.	
	L.8.4.0.0.	Subroutine 4 - Special Characters	
RTNE4	L.8.4.1.1.	Calculate the address of the desired special character(s) from the EOT table index found in byte 3 of the model CCW. Retain the address in WORKREG1.	WORKREG1
	L.8.4.1.2.	Obtain the count of special characters from the EOT table entry specified and retain it in WORKREG2.	WORKREG2
	L.8.4.1.3.	Return via LINKREG.	

MODULE NAME CHANNEL END APPENDAGE

MODULE IDENTIFICATION IGG019MB

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	M.0.0.0.	IOB Dummy Section (Appendix D)	
	M.1.0.0.	Determine Operation Type	
	M.1.1.0.	Save registers as obtained from IOS. IOS will have the return address for channel end appendages in LNKRG2.	
	M.1.2.0.	Calculate the address of the current (trap) CCW from the address of the next CCW to be executed, whose address is contained in bytes 1-3 of the IOB field (CSW) (see Figure 19 in Appendix D).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	M.1.3.0.	Test the TP Op Type byte from the current CCW for validity. If the TP Op Type is invalid (>18) skip to M.5.1.0.	
OPTBASE	M.1.4.0.	Test the TP Op Type byte from the current CCW for code value. If the TP Op Type byte is 0 (no-op) skip to TPABNORM (M.5.0.0.). If the TP Op Type byte is 4 (poll-restart) skip to TPPOLL01 (M.2.0.0.). If the TP Op Type byte is 8 (multiaddressing) skip to TPADDR01 (M.3.0.0.). If the TP Op Type byte is 0C, 10, 14, 18 (all not yet supported) skip to TPABNORM (M.5.1.0.).	
TPPOLL01	M.2.0.0.	Polling - Restart	
	M.2.1.0.	Calculate the address of the control byte of the current entry of the terminal (polling) list from the current entry address (bytes 1-3 of the second CCW) and the count field (bytes 6-7 of the second CCW). Retain the address in WKREG2.	WKREG2
	M.2.2.0.	Test for a wrong length record flag (bit 41) on in the IOB field (CSW). If the wrong length record flag is not on skip to TPABNORM (M.5.1.0.).	
	M.2.3.0.	Test the control byte of the current entry of the terminal list.	
	M.2.3.1.	Test for EOL and format bits on in the current entry. Skip to TPPOLL03 (M.2.4.1.) if neither bit is on.	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	M.2.3.2.	Test for the format bit on. If the format bit is not on (then only the EOL bit must be on) skip to TPABNORM (M.5.1.0.).	
TPPOLL02	M.2.4.0.	This is the end of a wraparound list. Calculate the address of the first polling character at the start of the list from the two bytes at the end of the list and the contents of WKREG2.	
TPPOLL03	M.2.4.1.	Increment the address in WKREG2 to become the address of the next entry to be polled and fill in the IOB field (POLPT) with this value.	WKREG2 IOBPOLPT
	M.2.5.0.	Test for the EOL bit on in the trap entry of the terminal list. If the EOL bit is on skip to TPABNORM (M.5.1.0.).	
	M.2.5.1.	Test for end-of-list flag (bit 7) on in IOB field (FLAG2). This bit is set by the RESETPL macro. If on, skip to RESETFLG (M.4.5.0.).	
	M.2.6.0.	Calculate the address of the control byte of the next entry in the terminal list from the count in the second CCW and the contents of WKREG2. Retain the address in WKREG2.	WKREG2
	M.2.7.0.	Test the control byte of the next entry of the terminal list.	
	M.2.7.1.	Test for the skip bit on in the entry of the terminal list. If the skip bit is not on skip to TPPOLL10 (M.2.8.0.).	
	M.2.7.2.	Test for the format bit on in this entry of the terminal list. If the format bit is not on skip to TPPOLL03. If the format bit is on skip to TPPOLL02 (M.2.4.0.).	

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
TPPOLL10	M.2.8.0.	Calculate the address of the first character of this entry in the terminal list from the count value of the next entry and the contents of WKREG2. Retain the result in WKREG2.	WKREG2
	M.2.9.0.	Skip to TPEXIT (M.4.0.0.).	
TPADDR01	M.3.0.0.	Multiaddressing	
	M.3.1.0.	Calculate the address of the control byte of the current entry in the terminal (addressing) list from the current entry address (bytes 1-3 of the second CCW) and the count field (bytes 6-7 of the second CCW). Retain the address in WKREG2.	WKREG2
	M.3.2.0.	Test CSW	
	M.3.2.1.	Test for unit exception bit (bit 39) on in the IOB field (CSW). If the unit exception bit is on (1050 terminal) skip to TPABNORM (M.5.1.0.). (This was a negative response.)	
	M.3.2.2.	Test for wrong length record bit (bit 41) on in the IOB field (CSW). If the wrong length record bit is not on skip to TPABNORM (M.5.1.0.).	
	M.3.3.0.	Test Terminal List Entry	
	M.3.3.1.	Test EOL bit in the control byte of the current terminal list entry. If the EOL bit is on an error has occurred. Skip to TPABNORM (M.5.1.0.).	
TPADDR02	M.3.4.0.	Calculate the address of the next entry in the terminal list and retain this address in WKREG2.	WKREG2
	M.3.5.0.	Calculate the address of the control byte of the next entry in the terminal list and retain this address in WKREG3.	WKREG3

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
	M.3.6.0.	Test the control byte of the next entry of the terminal list.	
	M.3.6.1.	Test for the skip bit on in this entry of the terminal list. If the skip bit is not on skip to TPADDR03 (M.3.9.0.).	
	M.3.6.2.	Test for the EOL bit on in this entry of the terminal list. If the EOL bit is on skip to TPABNORM (M.5.1.0.).	
	M.3.7.0.	Increment the address in WKREG2 to become the address of the control byte of this entry.	WKREG2
	M.3.8.0.	Skip to TPADDR02 (M.3.4.0.).	
TPADDR03	M.3.9.0.	Test for the EOL bit on in this terminal list entry. If the EOL bit is on skip to TPADDR05 (M.3.9.4.).	
TPADDR04	M.3.9.1.	Calculate the address of the control byte of the next entry in the terminal list from the address in WKREG3 and the count field of the second CCW in the IOB field (CPA). Retain this address in WKREG3.	WKREG3
	M.3.9.2.	Test for the skip bit on in this terminal list entry. If the skip bit is not on skip to TPADDR06 (M.3.9.5.).	
	M.3.9.3.	Test for the EOL bit on in this entry of the terminal list. If the EOL bit is on skip to TPADDR05 (M.3.9.4.). If the EOL bit is not on skip to TPADDR04 (M.3.9.1.).	
TPADDR05	M.3.9.4.	Set the command chain flag in the second CCW in the channel program area in the IOB field (CPA).	IOBCPA
TPADDR06	M.3.9.5.	Fill in the IOB field (ADRPT) with the contents of WKREG2 so that it contains the address of the next entry in the terminal list to be addressed.	IOBADRPT

LOCATION	BLD INDEX	FUNCTION PERFORMED	FIELD AFFECTED
TPEXIT	M.4.0.0.	Exit to IOS (EXCP return)	
	M.4.1.0.	Fill in the IOB field (START) with the address of the second CCW in the channel program area in the IOB field (CPA).	IOBSTART
	M.4.2.0.	Update the data address of the second CCW to contain the address of the next terminal to be addressed (contents of WKREG2).	WKREG2
	M.4.3.0.	Restore registers for return to IOS.	
	M.4.4.0.	Return to IOS at: the contents of LNKRG2 + 8.	
RESETFLG	M.4.5.0.	Turn off bit 7 of IOB field (flag 2).	IOBFLAG2
TPABNORM	M.5.0.0.	Exit to IOS (no-op return)	
	M.5.1.0.	Restore registers for return to IOS.	
	M.5.2.0.	Return to IOS at: the contents of LNKRG2 + 0.	

SECTION C: DEVICE I/O MODULES

Device I/O Modules are quantities of device-dependent information located in protected core storage. There will be one module for each terminal/option combination (device type) supported.

Each module consists of three basic parts:

1. Offset table
2. Model channel programs
3. Control character list

The offset table of a module contains values that are used by the Read-Write routine to index the model channel programs in that module. Each offset value points to the beginning of a model channel program for a particular I/O operation such as read initial, read repeat, etc.

The model channel program section is made up of sequences of model CCW's (channel command words), each sequence specifying a particular I/O operation. Each model CCW corresponds to an actual CCW that will be needed in the channel program and contains the information necessary for its construction (see Figure 11). The control characters section consists of a series of fields containing the count or number of characters that make up the field, followed by the actual control character or sequence of characters for the particular device associated with this module.

Device I/O Modules for IBM 1050, AT&T 83B3, and WU115A devices are shown in Figures 12, 13, and 14 respectively.

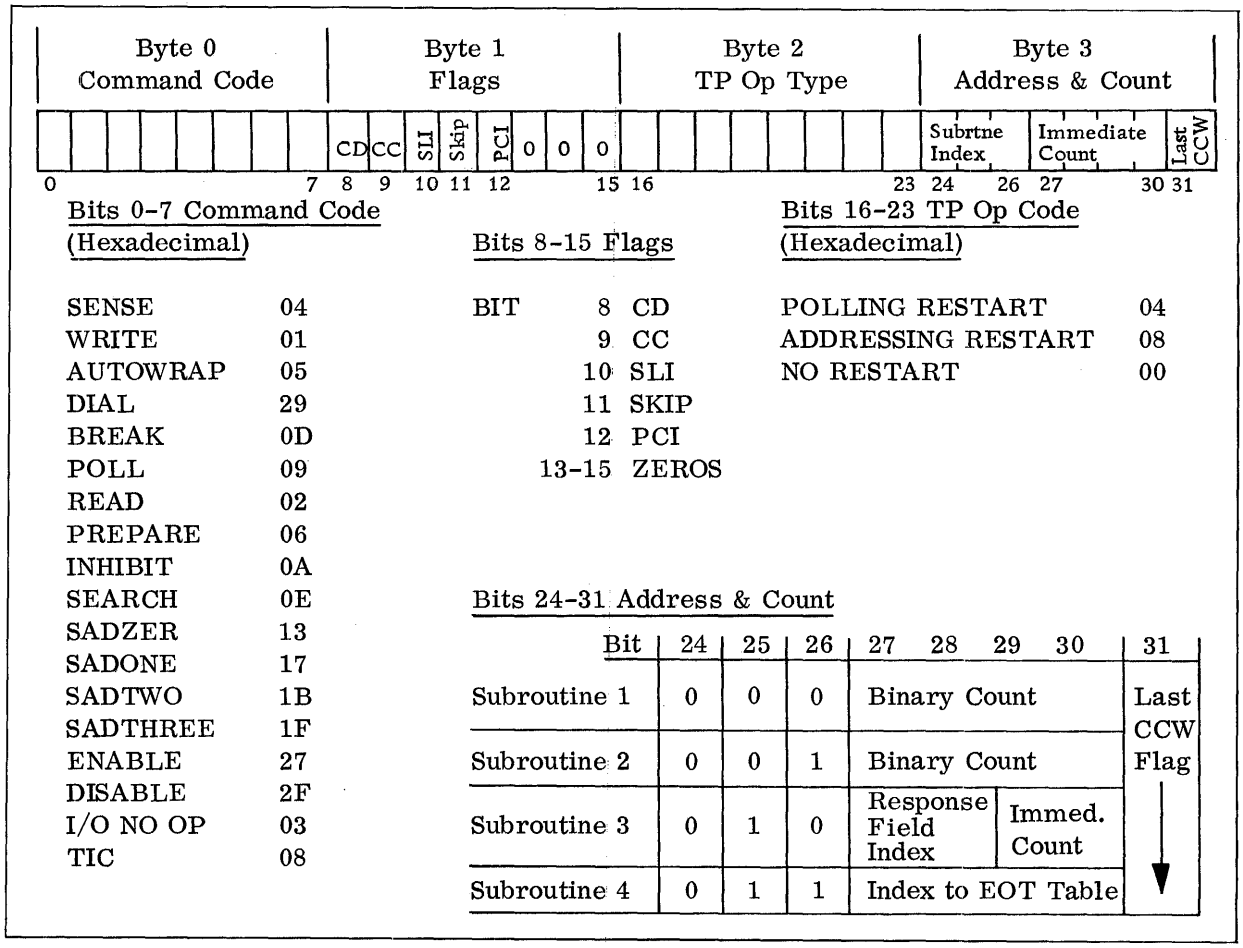


Figure 11. Model CCW

IGG019MD

	FF	00	10	28	OFFSET TABLE
	30	FF	FF	3C	
	FF	FF	FF	44	
+12	Write; CC, SLI; no-op; RTNE4, (C), not last CCW				READ INITIAL
	Write; CC, SLI; no-op; RTNE2, 2 char, not last				
	Read; CD; poll-restart; RTNE1, 2 char, not last				
	Read; SLI; no-op, RTNE1, no count needed, last				
+28	Write; CC, SLI; no-op; RTNE4, (C), not last				WRITE INITIAL
	Write; CC, SLI; no-op; RTNE2, 2 char, not last				
	Read;; multiaddr; RTNE3, 1 char, not last				
	Write; CC, SLI; no-op; RTNE1, no count, not last				
	Read; SLI; no-op; RTNE3, 1 char, last				
	Write; SLI; no-op; (C)				
+52	Write; CC, SLI; no-op; RTNE4, (Y), not last				READ CONTINUE
	Read; SLI; no-op; RTNE1, no count, last				
+60	Write; CC, SLI; no-op; RTNE1, no count, not last				WRITE CONTINUE
	Read;; no-op; RTNE3, 1 char, last				
	Write; SLI; no-op; (C)				
+72	Write; CC, SLI; no-op; RTNE4, (N), not last				READ REPEAT
	Read; SLI; no-op; RTNE1, no count, last				
+78	1	(C)	1	(Y)	CONTROL CHARACTERS
	1	(N)			

MODEL CHANNEL PROGRAMS

Figure 12. Device I/O Module for IBM 1050

IGG019ML

	FF	0	10	FF	OFFSET TABLE
	FF	FF	FF	FF	
	FF	FF	FF	28	
+12	Write; CC, SLI; no-op; RTNE4, FIGS H LTRS, not last				READ INITIAL
	Write; CC, SLI; no-op; RTNE2, 2 char, not last				
	Read; CD, poll-restart; RTNE1, 2 char, not last				
	Read; SLI; no-op; RTNE1, no count, last				
+28	Write; CC, SLI; no-op; RTNE4, FIGS H LTRS, not last				WRITE INITIAL
	Write; CC, SLI; no-op; RTNE2, 2 char, not last				
	Write; CC, SLI; no-op; RTNE4, LTRS, not last				
	Read;; multiaddr; RTNE3, 1 char, not last				
	Write; SLI; no-op; RTNE1, no count, last				
	Write; SLI; no-op; FIGS H LTRS				
+52	3	FIGS	H	LTRS	CONTROL CHARACTERS
	1	LTRS			

MODEL CHANNEL PROGRAMS

Figure 13. Device I/O Module for AT&T 83B3

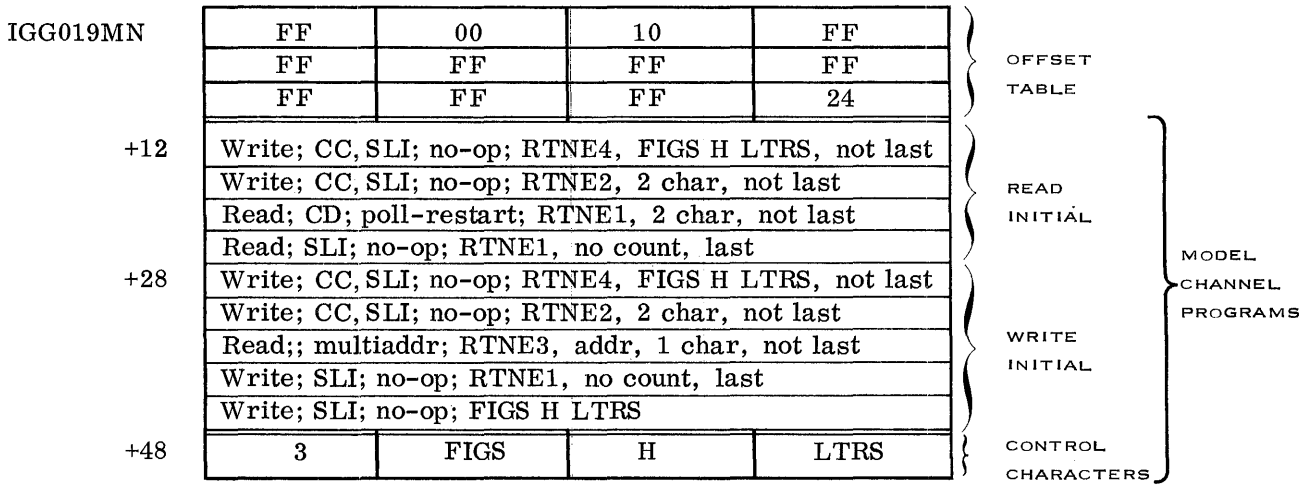


Figure 14. Device I/O Module for WU 115A

APPENDIX A: OPERATING SYSTEM/360 CONTROL BLOCK LINKAGES

Operating System/360 provides interfaces among programs by means of control blocks and tables. These blocks have standardized formats and contain numerous fields of information to be used and referenced by the program. Some of these fields are pointers to other blocks. Figure 15 shows the various blocks and their linkages of importance to BTAM.

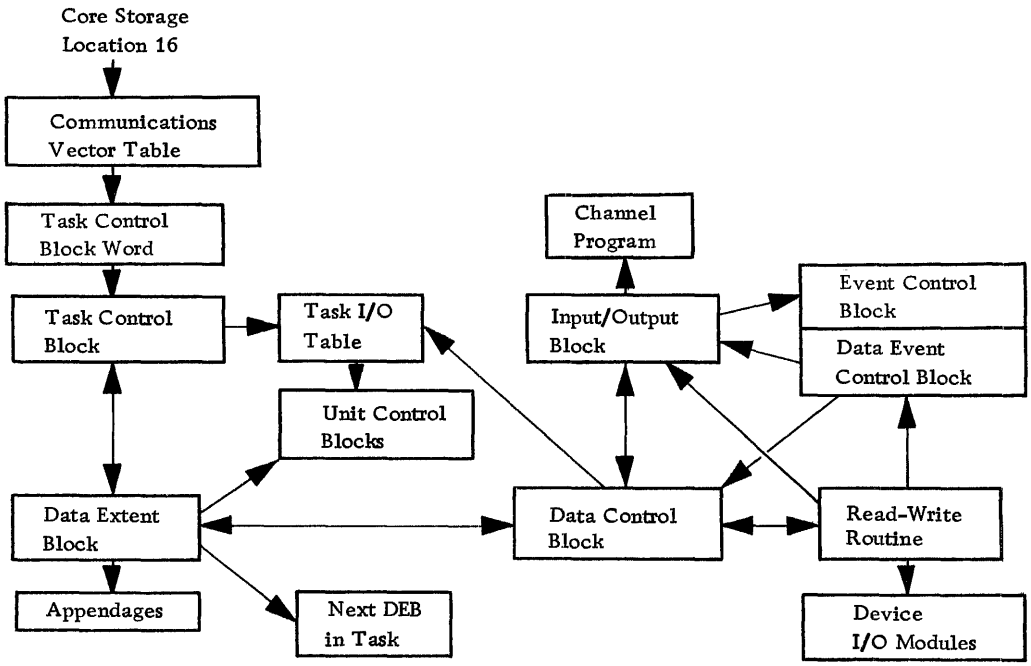


Figure 15. OS/360 control block linkages

APPENDIX B: DATA CONTROL BLOCK (DCB)

The data control block (DCB) provides information about its associated data set (in this case a communications line group). Each such data set will have a DCB. Expansion of the DCB macro instruction at assembly time reserves storage for the block and initializes the DCB with parameters from the macro defining the data set name, its organization, the macros used for input/output, and the exit list address.

Other parameters can be defined by the DD cards in the job stream, or dynamically by the program modules at any time before the DCB's data set is "opened". (There must be one DD card for each DCB in the program.)

Figure 16 shows a diagram of the DCB and Table 1 details the contents and possible sources of each field.

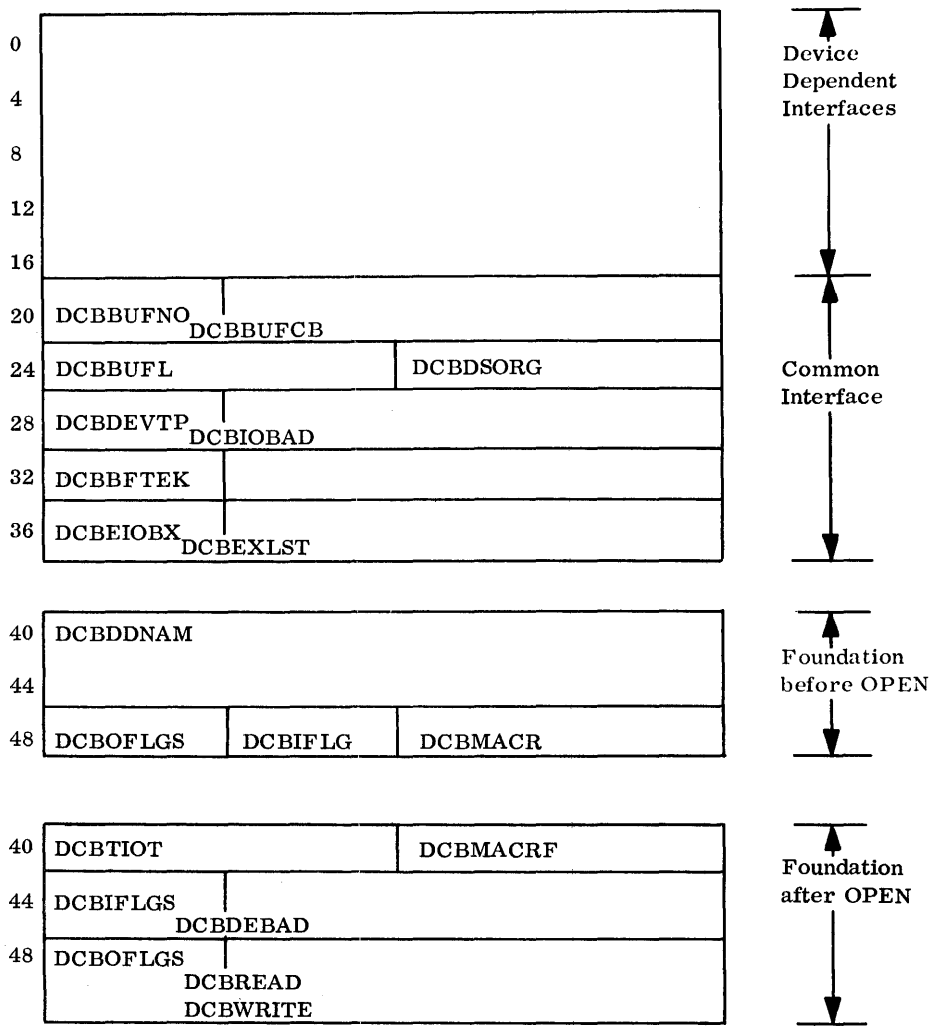


Figure 16. Data Control Block

Table 1. Data Control Block Fields

Section	Source*	Relative Location IHADCB+	Length (Bytes)	Name DCB+	Contents
Device Dependent Interfaces		0	20		Unknown
Common Interface	A, J, L	20	1	BUFNO	Number of buffers to be obtained by Open (0-255)
	D	20	4	BUFCB	Address of buffer control block
	A, J, L	24	2	BUFL	Buffer length: the length of buffers to be obtained by Open for a buffer pool, and/or the length to be used if the length parameter of a READ or WRITE macro is coded as 'S' (0-32764)
	J	26	2	DSORG	Data set organization: a communications line group specification of CX sets on bit 3.
	F, G	28	1	DEVTP	Index to device I/O directory
	F	28	4	IOBAD	IOB address: the address of the first IOB (the IOB for line 1) minus the length of an IOB (DCBEIOBX). The IOB address for any line associated with this DCB is equal to DCBIOBAD plus the product of the line number times DCBEIOBX.
	A, L, J	32	1	BFTEK	Buffering technique: Bit 4 - dynamic Buffer alignment: Bit 6 - doubleword boundaries
	F	36	1	EIOBX	Extended IOB index: size of IOB's associated with this DCB
	J	36	4	EXLST	Exit list: the address of a user-provided list which may contain an entry (control code and address) for a DCB exit
Foundation before Open	J	40	8	DDNAM	Data set name as used in data definition statement. Used by Open to locate job file control block (JFCB) address.
	D	48	1	OFLGS	Flags used by Open Bit 2 - EOVC: End of Volume sets this bit when it calls CLOSE for concatenation of data sets with unlike attributes. Bit 3 - OPEN: This bit set on when an OPEN has been successfully completed.

Table 1. (cont'd)

Section	Source	Relative Location IHADCB+	Length (Bytes)	Name DCB+	Contents
					Bit 4 - CONC: This bit is set on by a problem program to indicate a concatenation of unlike attributes.
					Bit 6 - LOCK: This bit is set on by an I/O support function if the DCB is to be processed by that function.
	D, F	49	1	IFLG	Used by IOS in communicating error conditions and in determining error procedures: 00xxxxxx Not in error procedure 01xxxxxx Error correction in process 11xxxxxx Permanent error condition xx10xxxx Channel 9 printer carriage xx01xxxx Channel 12 printer carriage xxxx00xx Always use IOS error routine xxxx01xx Test IOS mask (IMSK) for error procedure xxxx11xx **Never use IOS error routine xxxxxx11 Always use user error routine xxxxxx01 Test user mask (UMSK) for error procedure xxxxxx00 **Never use user error routine **BTAM Open always sets these two
	J	50	2	MACR	Macro instruction reference: specifies the major macros and various options associated with them. Used by Open to determine access method. Used by the access method executors in conjunction with other parameters to determine which load modules are required. Bit 2 - READ Bit 10 - WRITE } BTAM
Foundation after Open	D	40	2	TIOT	Point to DDNAME in Task I/O Table
	D	42	2	MACRF	Same as MACR
	F	44	1	IFLGS	Same as IFLG
	F	44	4	DEBAD	Address of the associated DEB
	D	48	1	OFLGS	Same as OFLGS above
	G	48	4	READ	Address of Read-Write module
	G	48	4	WRITE	Address of Read-Write module

*Source Codes

A Dynamic - any time before Open DCB exit
D IOS
E Read-Write
F Open Module 1
G Open Module 2
H Channel End Appendage
J Macro expansion in user program
L DD statement in job stream

APPENDIX C: DATA EXTENT BLOCK (DEB)

One data extent block (DEB) is created in protected core by Open Module 1 for each line group (data set). It contains tables of addresses of (1) the IOS appendages, (2) the unit control blocks for each line, and (3) other control blocks. There is also a list of the identifications of all BTAM modules needed to support the devices in this line group. Figure 17 is a diagram of the DEB, and Table 2 details the contents of each field, giving source codes where known.

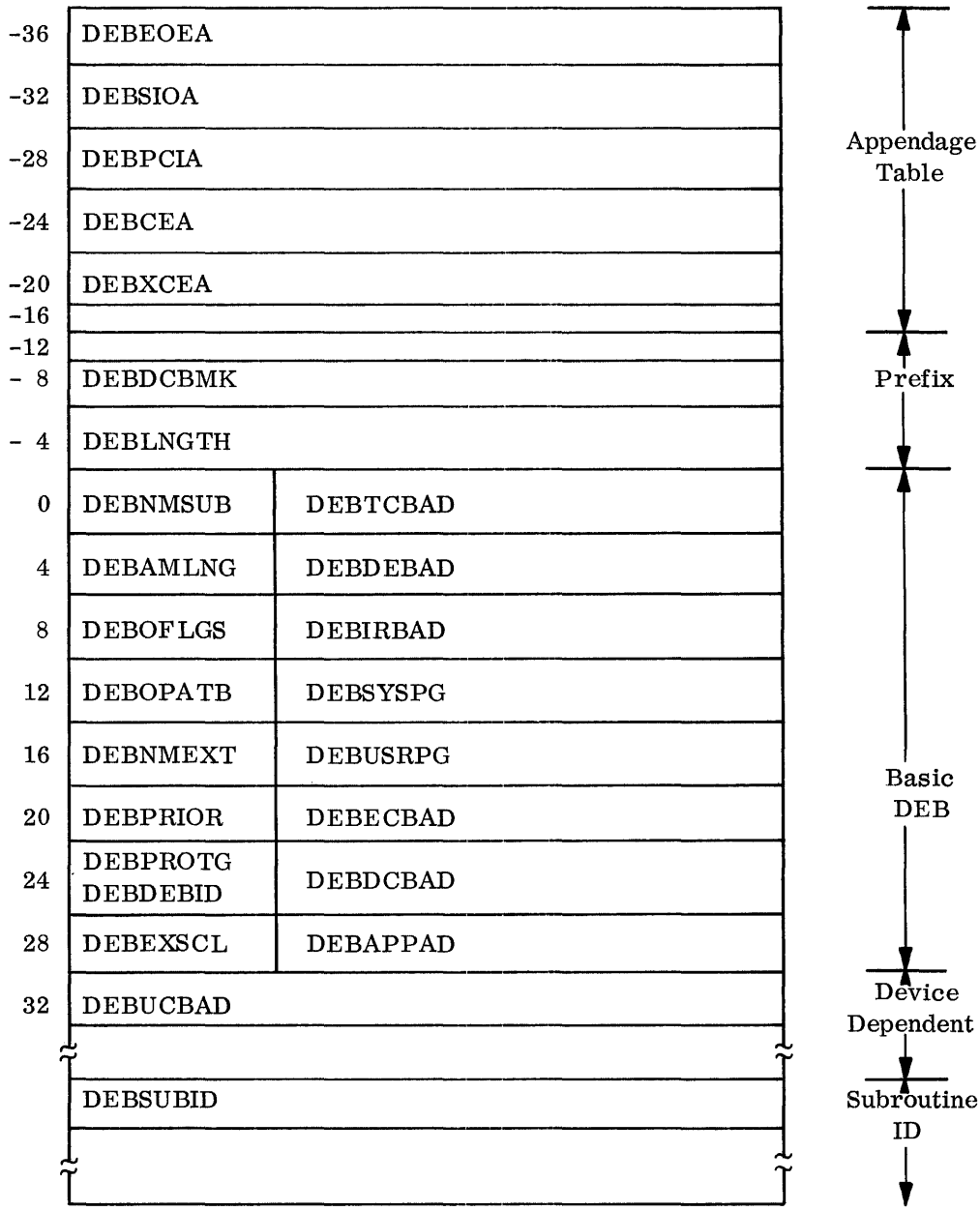


Figure 17. Data Extent Block

Table 2. BTAM DEB Fields

Section	Source	Relative Location IECTDEB+	Size in Bytes	Name DEB+	Contents
Appendage Table	F	-36	4	EOEA	Address of end of extent appendage branched to by IOS*
	F	-32	4	SIOA	Address of start I/O appendage branched to by IOS*
	F	-28	4	PCIA	Address of program controlled interrupt appendage branched to by IOS*
	G	24	4	CEA	Address of channel end appendage branched to by IOS*
	F	20	4	XCEA	Address of exceptional channel end appendage branched to by IOS*
Prefix	D	- 8	4	DCBMK	DCB modification mask used by I/O support
	F	- 4	1	LNGTH	Length of DEB in doublewords
Basic	G	0	1	NMSUB	Number of subroutines LOADed by Open Module 2
	F	1	3	TCBAD	TCB address of this DEB
		4	1	AMLNG	Number of bytes in access method section
		5	3	DEBAD	Address of next DEB in the same task
		8	1	OFLGS	Data set status flag: Bit 0 release unused external storage 1 end of volume or end of flag 2...7 reserved
		9	3	IRBAD	IRB address for error exit
		12	1	OPATB	Indicates file type: Bits 0 1 2 3 4 5 6 7 0 0 0 1 X X X X Reread 0 0 1 1 X X X X LEAVE 0 0 X X 0 0 0 0 INPUT 0 0 X X 1 1 1 1 OUTPUT 0 0 X X 0 0 1 1 INOUT 0 0 X X 0 1 1 1 OUTIN 0 0 X X 0 0 0 1 RDBACK 0 0 X X 0 1 0 0 UPDAT
		13	3	SYSPG	System purge chain Check IDLE
	F	16	1	NMEXT	Number of extents constructed (number of lines), specified in DSCB's
		17	3	USRPG	User purge chain
		20	1	PRIOR	Dispatching priority field from TCB, used by IOS for channel queuing of IOB's
		21	3	ECBAD	IOS internal ECB address

*Not supported by BTAM at present

Table 2. (cont'd)

Section	Source	Relative Location IECTDEB+	Size in Bytes	Name DEB+	Contents
	D	24	1/2	PROTG	Protection tag assigned to this task
	F	24 1/2	1/2	DEBID	Hexadecimal "F" identifies this block as a DEB
	F	25	3	DCBAD	Corresponding DCB address of this DEB
	F	28	1	EXSCL	Extent scale: = two for communication devices indicating four bytes per extent (used to determine size of Device Dependent section)
	F	29	3	APPAD	Address of I/O appendage table ahead of DEB
Device Dependent (Extents)	F	32	(a.)	UCBAD	Table of addresses of UCB's for each line (a.) Size = NMEXT shifted left logical EXSCL bits (in this case, four bytes/extent)
Subroutine ID	G		(b.)	SUBID	Two-character subroutine ID's - last two char of eight-byte name (b.) Size = 2 x NMSUB bytes

APPENDIX D: INPUT/OUTPUT BLOCK (IOB)

The input/output control block (IOB) provides communication between the user program and IOS. It is the sole parameter of the IOS execute channel program (EXCP) instruction. One IOB is created for each communications line by Open Module 1.

The basic IOB, 40 bytes in length, contains pointers to the channel program, the event control block, and the terminal lists, and provides areas for storing flags, sense bytes, the channel status word, and the start I/O condition code returned by IOS. Appended to each basic IOB is a variable-length area where the channel programs are constructed by the Read-Write routine.

Figure 18 is a diagram of the IOB and Figure 19 shows details of the channel status word (CSW) field. Table 3 contains descriptions and sources of the contents of the IOB fields.

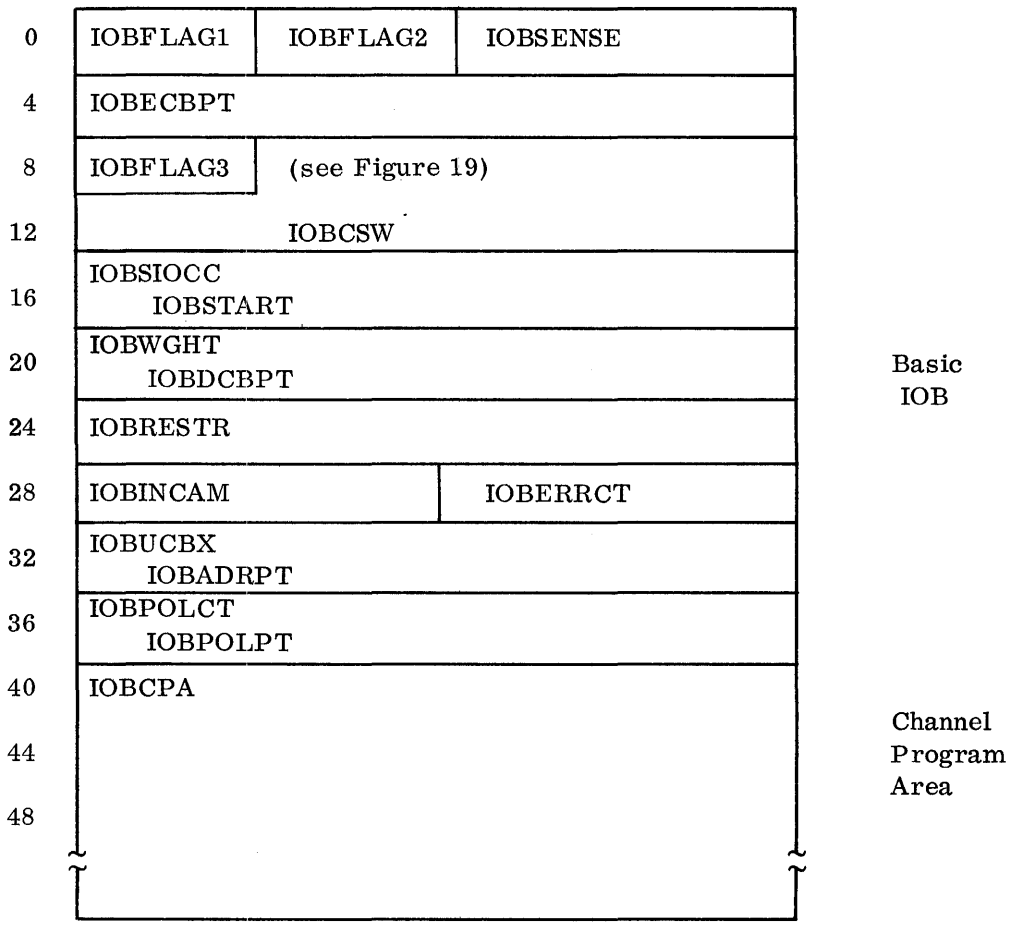
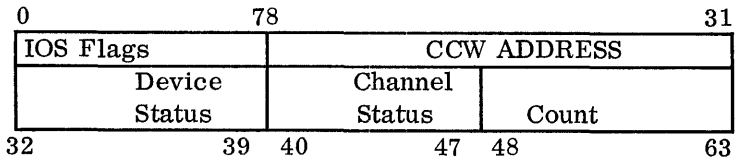


Figure 18. Input/output control block



From CAW	8-31	Address of Last CCW+8
Device Status	32	Attention Bit
	33	Status Modifier
	34	Control Unit End
	35	Busy
	36	Channel End
	37	Device End
	38	Unit Check
	39	Unit Exception
Channel Status	40	PCI
	41	Incorrect Length
	42-47	
Count	48-63	Count difference between number of characters sent or received and length specified.

Figure 19. Detail of IOB CSW field

Table 3. BTAM IOB Fields

Source	Relative Location	Length (Bytes)	Name IOB+	Contents
F	0	1	FLAG1	Flags: Bit 0 Data chaining in channel program 1 Command chaining in channel program 2 Sense flag 3, 4 Not used 5 IOB exception: used to flag an IOB in error 6 Unrelated: I/O requests need not be scheduled FIFO 7 Start/restart: IOS is to use channel program address IOBSTART (0) or IOBRESTR (1)
D	1	1	FLAG2	Flags: Bit 0 Error flag 1 . . 6 Internal IOS Flags 7 RESETPL

Table 3. BTAM IOB Fields (cont'd)

Source	Relative Location	Length (Bytes)	Name IOB+	Contents
D	2	2	SENSE	Two bytes of sense data stored here when an error occurs All zeros: successful completion Bit 0 Command reject 1 Intervention required 2 Parity error 3 Equipment check 4 Data check 5 Overrun 6 Receiving 7 Timeout
E, G	4	4	ECBPT	Address of event control block (ECB) associated with this I/O request
D	8	1	FLAG3	IOS Flags
D	9	7	CSW	The channel status word is stored here at channel end time
D	16	1	SIOCC	The condition code from execution of start I/O is stored here
E, H	16	4	START	Address of the first CCW at which to start I/O for normal conditions
	20	1	WGHT	Channel weight: user provides IOS with value of system loading imposed by this I/O request — not used by BTAM
F	20	4	DCBPT	Address of the DCB associated with this I/O request
F	24	4	RESTR	Address of CCW at which to start I/O for restart operations
	28	2	INCAM	Block count increment amount
	30	2	ERRCT	Error counter: not used by BTAM
F	32	1	UCBX	UCB index: line number — used as index to appropriate UCB address in the DEB
H	32	4	ADRPT	Pointer to addressing list
	36	1	POLCT	Poll count: the number of times polling has been consecutively initiated for the same terminal
E, H	36	4	POLPT	Poll pointer: address of currently active entry in polling list, or the last entry (EOL bit on), or the first entry (EOL and format bits on)
G, E	40		CPA	Channel program areas (length depends on terminal and options)

APPENDIX E: EVENT CONTROL BLOCK (ECB)

After initiating an I/O operation the task (user program) in control can continue processing until it needs the results of that operation. At this point is issues a WAIT instruction, which signals the supervisor that the task cannot proceed until completion of a specified event, for example, the I/O operation. The WAIT specifies an event control block (ECB), which is the first word of the data event control block formed by expansion of a READ or WRITE macro instruction. Figure 20 shows the ECB after the WAIT was issued. Bit 0 is the wait flag, and bit 1 is set on completion. Bits 8-31 specify the address of the task control block.

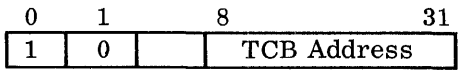
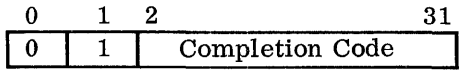


Figure 20. ECB after WAIT

The supervisor POSTs completion of the event by setting the wait flag off and the completion flag on. It inserts a completion code in bits 2-31. Figure 21 shows this.



Completion codes: ECB contents in hexadecimal
7F000000-completed without error (normal)
41000000-permanent error
48000000-I/O request purged or not started

Figure 21. ECB after POST

APPENDIX F: DATA EVENT CONTROL BLOCK (DECB)

The data event control block (DECB) is formed in the user program at assembly time by expansion of a READ or WRITE macro instruction with parameter MF=L or blank. It provides communication with the BTAM Read-Write module, specifying operation type, line group, line, and terminal list. Also included are areas for the standard ECB and responses to addressing and LRC checks. The format of the block is shown in Figure 22 and the detailed contents of its fields are in Table 4.

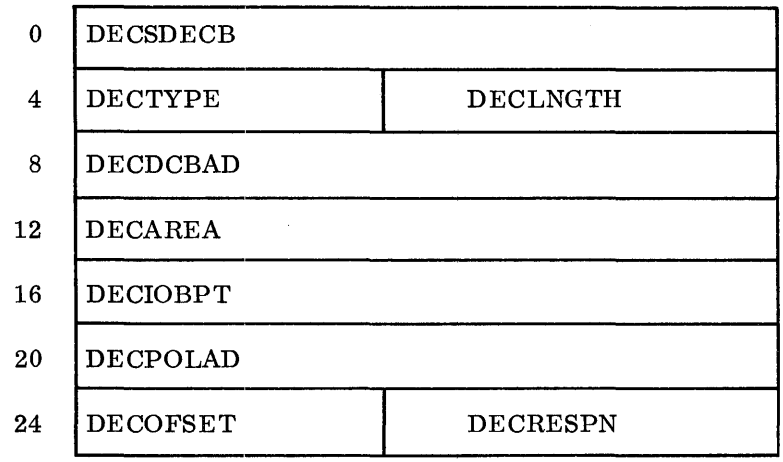


Figure 22. Data event control block

Table 4. DECB Fields

Source	Location IECTDECB+	Length	Name DEC+	Contents
D	0	4	SDECB	Event control block (ECB) for this I/O request (see Appendix E)
J	4	2	TYPE	Operation type 1st byte: Bit 5 - Terminal list coded as 'S' 6 - Area coded as 'S' 7 - Length coded as 'S' } in any combination 2nd byte: Bits 5, 6, 7

Value	Type Name	Type Code	Operation
0	Break	TB	Write
1	Initial	TI	Read
2	Initial	TI	Write
3	Continue	TT	Read
4	Continue	TT	Write
5	Conversation	TV	Read
6	Conversation	TV	Write
7	Repeat	TP	Read

Bit 0 - specifies Reset for type codes TIR, TTR, TVR, TPR
1 - specifies Inhibit for type codes TIH, TTH, TVH, TPH

Table 4. (cont'd)

Source	Location IECTDECB+	Length	Name DEC+	Contents
J, E	6	2	LNGTH	Buffer length
J	8	4	DCBAD	Address of associated DCB
J, E	12	4	AREA	Buffer address
G	16	4	IOBPT	Address of associated IOB
J, E	20	4	POLAD	Pointer to polling or addressing list
J	24	2	OFSET	Relative line number
D	26	2	RESPN	1st byte: address responses 2nd byte: LRC responses

APPENDIX G: UNIT CONTROL BLOCK (UCB)

A unit control block (UCB) is built for each line at system generation time and used by IOS during execution to determine physical locations. The only field requiring the attention of the BTAM user is the device type word, which gives details of the terminals on the line — control unit, adapter, model, and optional features. This word is fully described in Appendix H. Figure 23 shows the format of the UCB and Table 5 details its contents.

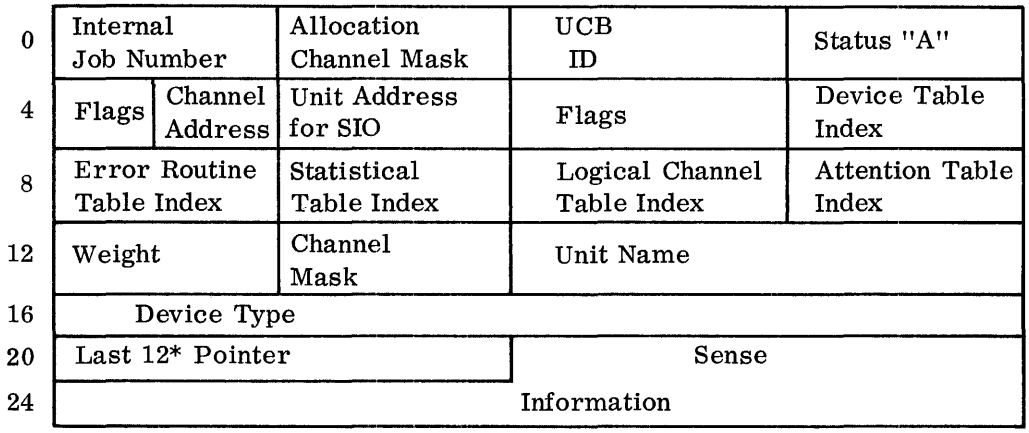


Figure 23. Unit Control block

Table 5. UCB Fields

Relative Location	Size	Name	Contents
0	1		Internal job number
1	1		Allocation channel mask
2	1		UCB identification
3	1		Status "A" flags
4	1		5 flag bits, 3-bit channel address
5	1		Unit address for SIO
6	1		Flags: Bit 0 UCB busy - set at SIO, reset at DE 1 UCB not ready - awaiting operator intervention 2 Post flag - waiting DE or error at DE will be passed at next SIO 3 UCB intercept 4 Control unit busy 5 Disk data transfer 6 Disk arm seeking 7 Status modifier
7	1	DEVTAB	Index on device table (one entry/device type)
8	1	ERRTAB	Index used to get error routine for this device
9	1	STATAB	Indexes statistical information table
10	1	LCHTAB	Indexes table of logical channel words (one per logical channel)
11	1	ANTAB	Indexes a table of attention routines to which IOS may pass control
12	1		Weight, used in computing overrun
13	1		Channel mask
14	1		Unit name: symbolic, used by allocator (in messages to operator, etc.)
16	1		Device type work (see Appendix H)
20	2		Last 12*: pointer to last or present active I/O request
22	2 or 6		Sense information

APPENDIX H: UCB DEVICE CODES

The device type word located at relative location 16 in the unit control block has the format shown in Figure 24.

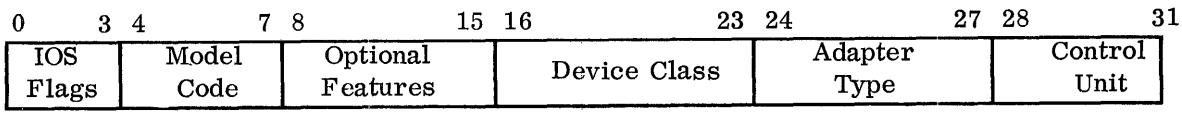


Figure 24. UCB device code word

Interpretation of the contents of the fields is shown below.

	Bit	Meaning	Hexadecimal Value
IOS Flags	0	Unassigned	
	1	Data chaining 1 = yes	
	2	Burst/byte 1 = burst	
	3	Overrunnable 1 = yes	
Device Class	16	Tape	80
	17	Communications equipment	40
	18	Direct access	20
	19	Display	10
	20	Unit record	08
	21	Character reader	04
	22	Spare	02
	23	Spare	01
Control Unit	28-31	Hexadecimal value: 1 - 2702 2 - 2701 3 - 7770 4 - 7772 5 : F } not assigned	
Adapter Type	24-27	Hexadecimal value: 1 - IBM Terminal Adapter Type I 2 - IBM Terminal Adapter Type II 3 - IBM Telegraph Adapter 4 - Telegraph Adapter Type I 5 - Telegraph Adapter Type II 6 - World Trade Telegraph Adapter 7 - Synchronous Adapter Type I 8 - IBM Terminal Adapter Type III 9 : F } not assigned	
Model Code	4-7	Hexadecimal value: (see Table 6 below)	

Table 6. Current BTAM-Supported Terminals

Adapter Code	Model Code		
	1	2	3
1	1050	1060	1070
2	1030		
3	1050		
4	83B3	115A	
5	TWX		

	Bit	Meaning
Optional Features	8	Automatic Call
	9	Automatic Poll
	10	Terminal-to-Terminal Transmission
	11	Automatic Answer
	12-15	Hexadecimal value: 0 - SADZER 1 - SADONE 2 - SADTWO 3 - SADTHREE

APPENDIX I: BTAM OPEN POINTERS AND TABLES

Figure 25 is included for reference to show the path followed by BTAM to access UCB's while a line group is being opened.

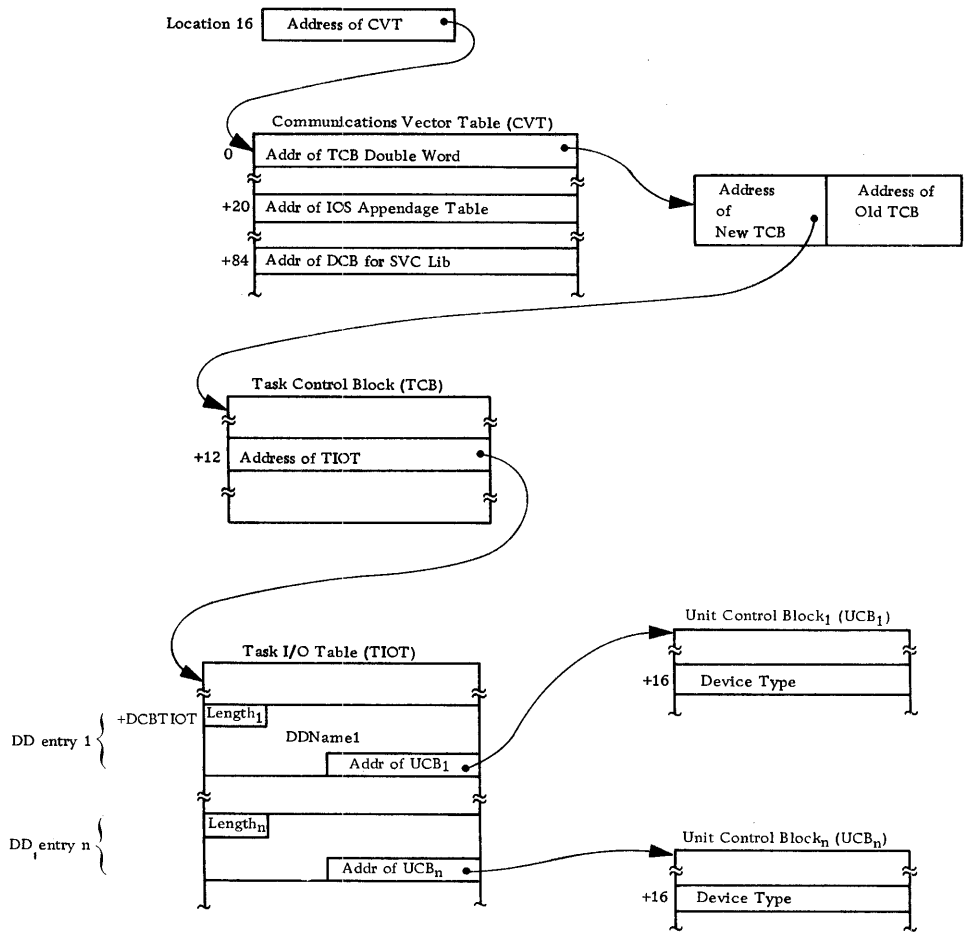


Figure 25. Open pointers and tables

APPENDIX J: BTAM READ-WRITE POINTERS AND TABLES

When a READ or WRITE command is issued, control is passed to the Read-Write routine whose address is stored in the DCB specified in the macro. Using the device type code in the DCB to index its Device I/O Directory, Read-Write accesses the proper Device I/O Module. The operation type code in the DECB indexes the offset table in the module to calculate the address of the desired model channel program. Read-Write expands the model, using information in the DECB, and stores the complete channel program in the IOB. Control is passed to IOS, which executes the channel program. Figure 26 details the flow.

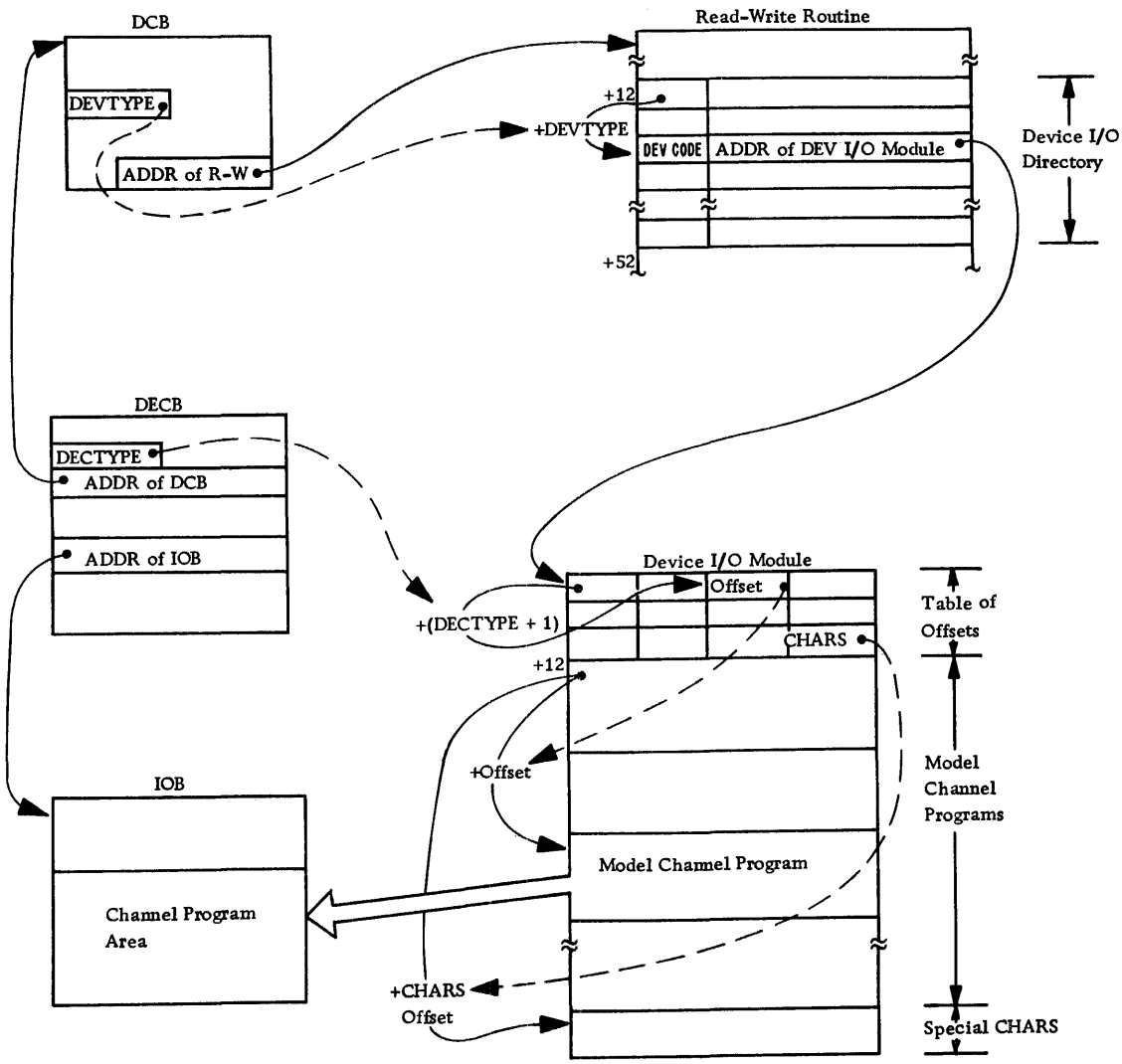


Figure 26. Read-Write pointers and tables

APPENDIX K: ABEND, RETURN, AND COMPLETION CODES

If an error is discovered while the Open executor is processing the information found in UCB's, an abnormal end-of-task exit (SVC 13) is taken. A code indicating the error type is left in general register 1. The codes and their meanings follow:

(load 1)	00090000	Device class 40 (Telecommunications) not specified
	00091000	2701 or 2702 not specified
	00092000	Terminal adapter type incorrect
	00093000	Device type incorrect for adapter
	00094000	Special features incorrect for device
(load 2)	00095000	2701 or 2702 not specified

The Read-Write routine returns I/O start codes to the problem program via register 15. The codes used there are:

00	Normal
04	Skip and end-of-list both set in polling/addressing list
08	Invalid I/O command
0C	Channel busy

I/O completion codes are returned in the event control block (see Appendix E for those codes).

APPENDIX L: BTAM MODIFICATION FOR IBM 2740 TERMINAL

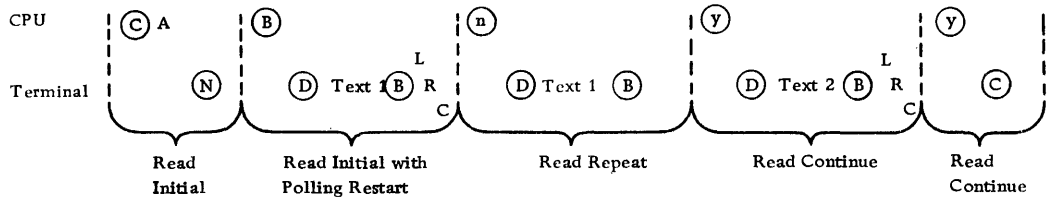
The following is an example of the use of the BTAM modification guide to provide BTAM operation of an IBM 2740 terminal with VRC, LRC checking, station control options. This is untested coding and is not official programming support for the 2740 terminal. It is included here only for educational purposes.

Building a 2740 Device I/O Module

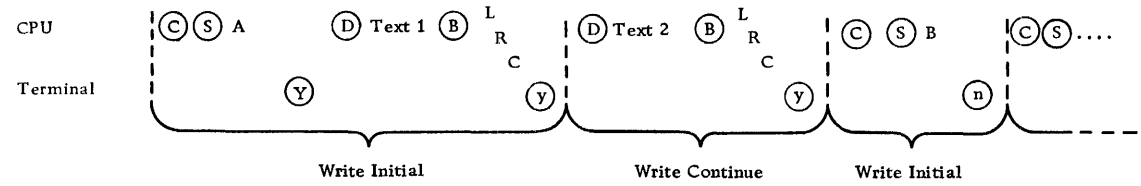
With the options specified, the 2740 can operate in a multidrop leased/private line environment and is a "regular" terminal in that dial-up and answering are not implemented. The operations to be supported are polling and addressing (read and write initial), read and write continue, and read repeat.

As discussed under "Building a Device I/O Module" in the section entitled "Probable Modifications", the first step is to define the line control sequence needed to perform the desired operation.

Reading a Multisegment Message from a 2740 Terminal



Writing a Multisegment Message to a 2740 Terminal



We are now in a position to write a narrative for each operation as follows:

Read Initial
1. Write (C) character ((C) indicates polling)
2. Write terminal ID character
3. Read response to polling
4. Read data

Write Initial
1. Write (C) (S) characters ((C) (S) indicates addressing)
2. Write terminal ID character
3. Read response to addressing
4. Write data
5. Read answer to LRC

Read Continue
1. Write (Y) positive response
2. Read data

Write Continue
1. Write data
2. Read answer to LRC

Read Repeat
1. Write (N) negative response
2. Read data

1. Set up a table of special characters for 2740 line control. The character table will need (C), (CS), (Y), and (N). ((D), (B) will be in the messages.)

			Value
TABLE	DC X'01'		
	DC X'1F'	(C)	0
	DC X'02'		
	DC X'1F'	(C)	1
	DC X'37'	(S)	
	DC X'00'	(Padding)	
	DC X'01'		
	DC X'76'	(Y)	3
	DC X'01'		
	DC X'40'	(N)	4

2. Write out detailed 2740 command sequences.

	Command	Address	Flags	TP Op Type	Count
Read Initial:	Write (C)	Table	CC, SLI	0	1
	Write poll char	List	CC, SLI	0	2
	Read resp	Area	CD	Poll-restart	2
	Read data	Area + 2	SLI	0	Length-2
Write Initial:	Write (CS)	Table	CC, SLI	0	2
	Write Addr char	List	CC, SLI	0	2
	Read resp	RESPN	CC	0	1
	Write data	Area	CC, SLI	0	Length
	Read answer	RESPN+1		0	1
Read Continue:	Write (Y)	Table	CC, SLI	0	1
	Read data	Area	SLI	0	Length
Write Continue:	Write data	Area	SLI	0	Length
	Read answer	RESPN+1		0	1
Read Repeat:	Write (N)	Table	CC, SLI	0	1
	Read data	Area	SLI	0	Length

3. Build model CCW's for 2740 support.
We can now construct the model CCW's ---------- in hexadecimal

Read Initial:

Write; CC, SLI; normal; RTNE4, (C), not last CCW	01600060
Write; CC, SLI; normal; RTNE2, 2 char, not last	01600024
Read; CD; poll-restart; RTNE1, 2 char, not last	02800404
Read; SLI; normal; RTNE1, no count needed, last	02200001

Write Initial:

Write; CC, SLI; normal; RTNE4, (C)(S), not last	01600062
Write; CC, SLI; normal; RTNE2, 2 char, not last	01600024
Read; CC, SLI; normal; RTNE3, addr, not last	02600042
Write; CC, SLI; normal; RTNE1, no count needed, not last	01600000
Read; SLI; normal; RTNE3, LRC, last	0220004B

Read Continue:

Write; CC, SLI; normal; RTNE4, (Y), not last	01600066
Read; SLI; normal; RTNE1, no count needed, last	02200001

Write Continue:

Write; CC, SLI; normal; RTNE1, no count needed, not last	01600000
Read; SLI; normal; RTNE3, LRC, last	0220004B

Read Repeat:

Write; CC, SLI; normal; RTNE4, (N), not last	01600068
Read; SLI; normal; RTNE1, no count needed, last	02200001

Writing the 2740 model CCW's in assembler language and determining the "offset" values of each model channel program:

```
offset (hexadecimal)
  0                RDINIT1     DC X'01'
                               DC X'60'
                               DC X'00'
                               DC X'60'

                   RDINIT2     DC X'01'
                               DC X'60'
                               DC X'00'
                               DC X'24'
```

```
                RDINIT3     DC X'02'
                            DC X'80'
                            DC X'04'
                            DC X'04'

                RDINIT4     DC X'02'
                            DC X'20'
                            DC X'00'
                            DC X'01'

                RITEINT1    DC X'01'
                            DC X'60'
                            DC X'00'
                            DC X'62'

                RITEINT2    DC X'01'
                            DC X'60'
                            DC X'00'
                            DC X'24'

                RITEINT3    DC X'02'
                            DC X'60'
                            DC X'00'
                            DC X'42'

                RITEINT4    DC X'01'
                            DC X'60'
                            DC X'00'
                            DC X'00'

                RITEINT5    DC X'02'
                            DC X'20'
                            DC X'00'
                            DC X'4B'

24              RDCONT1     DC X'01'
                            DC X'60'
                            DC X'00'
                            DC X'66'

                RDCONT2     DC X'02'
                            DC X'20'
                            DC X'00'
                            DC X'01'
```

```
2C              RITECNT1    DC X'01'
                            DC X'60'
                            DC X'00'
                            DC X'00'

                RITECNT2    DC X'02'
                            DC X'20'
                            DC X'00'
                            DC X'4B'

34              RDRPETE1    DC X'01'
                            DC X'60'
                            DC X'68'

                RDRPETE2    DC X'02'
                            DC X'20'
                            DC X'00'
                            DC X'01'
```

Now we can code the "offset" values in assembler language.

```
IGG019MZ    CSECT
            DC X'FF'
            DC X'00'        Read Initial
            DC X'10'        Write Initial
            DC X'24'        Read Continue
            DC X'2C'        Write Continue
            DC X'FF'
            DC X'FF'
            DC X'34'        Read Repeat
            DC X'FF'
            DC X'FF'
            DC X'FF'
            DC X'3C'        Special Characters
```

To form the Device I/O Module we now combine the three parts:

1. Offset table
2. Model channel programs
3. Character table

This is the Device I/O Module designed for the specified 2740 terminal support.

Device Type Analysis. With the Device I/O Module specified, it is now necessary to run through the steps listed under "Coding Changes" in the section entitled "Probable Modifications". Columns 1-6 in the table below will contain, respectively, all terminal and option combinations (current BTAM support plus 2740), the sizes of IOB channel program areas, the Device I/O Module ID characters, a hexadecimal integer, the adapter code(s), and model code.

		Column			
1	2	3	4	5	6
1050 R	48	MD	00	1, 3	1
1050 AP	32	ME	01	1, 3	1
1050 A	48	MF	02	1, 3	1
1050 AA	56	MG	03	1, 3	1
1050 AC	64	MH	04	1, 3	1
1060 R	40	MI	05	1	2
2740 R	40	MZ	06	1	4
1030 R	40	MJ	07	2	1
1030 AP	16	MK	08	2	1
83B3 R	48	ML	09	4	1
83B3 TT	24	MM	0A	4	1
115A R	40	MN	0B	4	2
115A TT	72	MO	0C	4	2
TWX A	80	MP	0D	5	1
TWX AA	88	MQ	0E	5	1
TWX AC	16	MR	0F	5	1

R - regular
AP - auto poll
AA - auto answer
AC - auto call
A - both AA and AC
TT - terminal-to-terminal

Now we can build the IOB channel program area size table (SZTABLE) from the digits in column 2. This table will be substituted for the present SZTABLE coding in BTAM Open Module 1 (IGG0193M).

Name	Operation	Operand	Remarks
SZTABLE	DC	HL1'48'	1050R
	DC	HL1'32'	1050AP
	DC	HL1'48'	1050A
	DC	HL1'56'	1050AA
	DC	HL1'64'	1050AC
	DC	HL1'40'	1060R
	DC	HL1'40'	2740R
	DC	HL1'40'	1030R
	DC	HL1'16'	1030AP
	DC	HL1'48'	83B3R
	DC	HL1'24'	83B3TT
	DC	HL1'40'	115AR
	DC	HL1'72'	115ATT
	DC	HL1'80'	TWXA
	DC	HL1'88'	TWXAA
	DC	HL1'16'	TWXAC

From the digits in column 3 we can build the Module ID table (CHNIDTBL). This table will replace the CHNIDTBL coding in Open Module 2 (IGG0193Q).

Name	Operation	Operand	Remarks
CHNIDTBL	DC	C'MD'	1050R
	DC	XL4'0'	TTRL FIELD
	DC	C'ME'	1050AP
	DC	XL4'0'	
	DC	C'MF'	1050A
	DC	XL4'0'	
	DC	C'MG'	1050AA
	DC	XL4'0'	
	DC	C'MH'	1050AC
	DC	XL4'0'	
	DC	C'MI'	1060R
	DC	XL4'0'	
	DC	C'MZ'	2740R
	DC	XL4'0'	
	DC	C'MJ'	1030R
	DC	XL4'0'	
	DC	C'MK'	1030AP
	DC	XL4'0'	
	DC	C'ML'	83B3R
	DC	XL4'0'	
	DC	C'MM'	83B3TT
	DC	XL4'0'	
	DC	C'MN'	115AR
	DC	XL4'0'	
	DC	C'MO'	115ATT
	DC	XL4'0'	
	DC	C'MP'	TWXA
	DC	XL4'0'	
	DC	C'MQ'	TWXAA
	DC	XL4'0'	
	DC	C'MR'	TWXAC
	DC	XL4'0'	

From column 4 we can build the Terminal Code Table for Open Module 1.

Name	Operation	Operand	Remarks
CDIBM1	DC	X'00'	1050R
	DC	X'01'	1050AP
	DC	X'02'	1050A
	DC	X'03'	1050AA
	DC	X'04'	1050AC
	DC	X'05'	1060R
	DC	X'06'	2740R

```
Name        Operation        Operand          Remarks

CDIBM2        DC             X'07'            1030R
              DC             X'08'            1030AP

CDTEL1        DC             X'09'            83B3R
              DC             X'0A'            83B3TT
              DC             X'0B'            115AR
              DC             X'0C'            115ATT

CDTEL2        DC             X'0D'            TWXA
              DC             X'0E'            TWXAA
              DC             X'0F'            TWXAC
```

Device Type Analysis Coding (step 9 of the "Modification Procedure"). Possibly the best way to modify this section of BTAM Open Module 1 is to rewrite it. Its purpose is to check for the terminals supported by the system in use. For a particular system with specific terminals there is no need to include tests for all terminal/option combinations. Sections of coding can thereby be eliminated by a customized version of the device type analysis.

The sample coding that follows represents an extreme case in that the 2740 is being added to all of the current BTAM support — nothing is being deleted. The method used to do the analysis here differs only slightly from that of the original coding, mostly near the beginning. No superiority is claimed — it merely shows an alternate way to test for invalid terminals and to perform tests 3 and 4. Narrative and coding will be interspersed from here on.

The device type analysis begins at symbolic location DEVTYANL (BLD, Section J.4.0.0.). The three general registers to be used are cleared and the integer value 1 is loaded into a fourth for use as a "bumper" later in the program. The address of the UCB device word is computed and put in register DEVTYREG. Tests 1 and 2 are performed on the device class and control unit type, which must be communications and 2701 or 2702, respectively. Location ABENDT contains the supervisor call for abnormal end-of-task.

```
Name        Operation    Operand                 Remarks

DEVTYANL      SR         UNITYREG, UNITYREG
              SR         DEVCDREG, DEVCDREG
              SR         DEVSTREG, DEVSTREG
              LA         DEVBUMPR, 1(0, 0)
              L          DEVTYREG, DEBUCBAD      UCB ADDRESS
              AH         DEVTYREG, SIXTEEN       DEV TYPE WORK ADDR
              CLI        2(DEVTYREG), X'40'      DEVICE CLASS - MUST BE
              BNE        ABENDT                  40 = COMMUNICATIONS
              TM         3(DEVTYREG), X'03'      CONTROL UNIT CODE MUST
              BC         9, ABENDT               BE 1 (2702) or 2 (2701)
```

Next the terminal adapter code is extracted, placed in UNITYREG, and checked to see if it exceeds the highest adapter code supported, in this case 5. Subtracting 1 makes it an index, and a left shift one position makes it usable on a table of halfwords.

Name	Operation	Operand	Remarks
	IC	UNITYREG, 3(DEVTYREG)	GET FOURTH BYTE
	SRL	UNITYREG, 4	ADAPTER CODE IN REG
	CH	UNITYREG, HIGHAD	CHECK AGAINST HIGHEST
	BH	ABENDT	SUPPORTED ADAPTER CODE
	SH	UNITYREG, HWC1	MAKE INDEX ON HALFWORD
	SLL	UNITYREG, 1	BOUNDARY

The model code is now extracted, left in DEVCDREG, and tested to see if it exceeds the highest model code within its adapter class. Subtracting 1 and left shifting 2 make it a fullword index. The adapter index in UNITYREG is also shifted to fullword.

Name	Operation	Operand	Remarks
	IC	DEVCDREG, 0(DEVTYREG)	GET FIRST BYTE MODEL CODE IN REG
	CH	DEVCDREG, HIGHMOD (UNITYREG)	CHECK IF MODEL CODE
	BH	ABENDT	GREATER THAN SUPPORTED
	SH	DEVCDREG, HWC1	MAKE INDEX ON FULLWORD
	SLL	DEVCDREG, 2	BOUNDARY
	SLL	UNITYREG, 1	PUT ON FULLWORD BOUNDARY

The next instruction is a branch to location MODELRT indexed by UNITYREG (adapter code), that is, to one of the five branch instructions at or following MODELRT. This completes test 3 and the terminal has been specified as far as the adapter type.

Name	Operation	Operand
	B	MODELRT(UNITYREG)

The next branch (one of the five below) is indexed by DEVCDREG (model code) and will be to a location within one of the lists for the adapter types.

Name	Operation	Operand
MODELRT	B	TYPE1(DEVCDREG)
	B	TYPE2(DEVCDREG)
	B	IBMTEL(DEVCDREG)
	B	TELTYPE1(DEVCDREG)
	B	TELTYPE2(DEVCDREG)

The final branch will be to one of the terminal sections and marks the completion of test 4 — the device type is known.

```
Name        Operation   Operand     Remarks

TYPE1           B       D1050
                B       D1060
                B       ABENDT      1070 NOT SUPPORTED
                B       D2740

TYPE2           B       D1030

IBMTEL          B       D1050

TELTYPE1        B       D83B3
                B       D115A

TELTYPE2        B       DTWX
```

The previous coding could have been condensed somewhat, considering that three of the adapter types only go with one terminal type each. Thus the branch to TYPE2 could have been replaced with the one to D1030, effectively eliminating one indexed branch instruction. However, the form shown is more easily expanded.

Before proceeding to the option tests, it would be appropriate to include the constants used by the program so far. HIGHAD is the highest adapter code supported, currently 5. The list starting at HIGHMOD contains the highest model codes for each adapter type.

```
Name        Operation   Operand     Remarks

SIXTEEN         DC      H'16'

HWC1            DC      H'1'

HIGHAD          DC      H'5'        HIGHEST ADAPTER CODE

HIGHMOD         DC      H'4'        IBM1 — HIGHEST MODEL CODES
                DC      H'1'        IBM2
                DC      H'1'        IBMTEL
                DC      H'2'        TELTYPE1
                DC      H'1'        TELTYPE2
```

The next sections of coding make the terminal option tests. The general procedure is to check for each option in the order listed earlier, and to generate an offset for skipping down the terminal code table to pick up the proper code.

The terminal code byte is inserted into the last eight bits of register DEVCDREG. It will be used later as an index on the IOB size table and on the I/O Module ID table.

Name	Operation	Operand	Remarks
D1050	TM	1(DEVTYREG), X'F0'	ANY OPTIONS
	BZ	IBM1	NO-REGULAR
	AR	DEVSTREG, DEVBUMPR	YES
	TM	1(DEVTYREG), X'40'	AP OPTION
	BO	IBM1	YES
	AR	DEVSTREG, DEVBUMPR	NO
	TM	1(DEVTYREG), X'90'	A OPTION-BOTH AA AND AC
	BO	IBM1	YES
	AR	DEVSTREG, DEVBUMPR	NO
	TM	1(DEVTYREG), X'10'	AA OPTION
	BO	IBM1	YES
	AR	DEVSTREG, DEVBUMPR	NO
	TM	1(DEVTYREG), X'80'	AC OPTION
	BO	IBM1	YES
	B	ABENDT	NO-ANY OTHER IN ERROR
D2740	LA	DEVSTREG, 1(0, 0)	
D1060	LA	DEVSTREG, 5(0, DEVSTREG)	
	TM	1(DEVTYREG), X'F0'	ANY OPTIONS
	BZ	IBM1	NO
	B	ABENDT	ANY = ERROR
IBM1	IC	DEVCDREG, CDIBM1 (DEVSTREG)	
	B	IOBST1	
D115A	LA	DEVSTREG, 2(0, 0)	
D83B3	TM	1(DEVTYREG), X'F0'	ANY OPTIONS
	BZ	TELTY1	NO
	AR	DEVSTREG, DEVBUMPR	YES
	TM	1(DEVTYREG), X'20'	TT OPTION
	BO	TELTY1	YES
	B	ABENDT	ANY OTHERS IN ERROR
TELTY1	IC	DEVCDREG, CDTEL1 (DEVSTREG)	
	B	IOBST1	

Name	Operation	Operand	Remarks
D1030	TM	1(DEVTYREG), X'F0'	ANY OPTION
	BZ	IBM2	NO-REGULAR
	AR	DEVSTREG, DEVBUMPR	YES
	TM	1(DEVTYREG), X'40'	AP OPTION
	BO	IBM2	YES
	B	ABENDT	NO-ANY OTHER IN ERROR
IBM2	IC	DEVCDREG, CDIBM2 (DEVSTREG)	
	B	IOBST1	
DTWX	TM	1(DEVTYREG), X'90'	A OPTION
	BO	TELTY2	YES
	AR	DEVSTREG, DEVBUMPR	NO
	TM	1(DEVTYREG), X'10'	AA OPTION
	BO	TELTY2	YES
	AR	DEVSTREG, DEVBUMPR	NO
	TM	1(DEVTYREG), X'80'	AC OPTION
	BO	TELTY2	YES
	B	ABENDT	NO-ANY OTHERS IN ERROR
TELTY2	IC	DEVCDREG, CDTEL2 (DEVSTREG)	
	B	IOBST1	

The branch to symbolic location IOBST1 marks the return to the regular coding following the device type analysis. Remember that the IOB Size Table, I/O Module ID Table, and Terminal Code Tables replace those in the original BTAM Open Modules.

BIBLIOGRAPHY

IBM OPERATING SYSTEM/360

IBM Operating System/360, Telecommunications: Preliminary Specifications, Systems Reference Library (C28-6553).

IBM Operating System/360, Control Program Services, Systems Reference Library (C28-6541).

IBM System/360 Operating System, Data Management, Systems Reference Library (C28-6537).

IBM Operating System/360, Assembler Language, Systems Reference Library (C28-6514).

IBM Operating System/360, System Programmer's Guide, Systems Reference Library (C28-6550).

IBM SYSTEM/360

IBM System/360, Principles of Operations, Systems Reference Library (A22-6821).

IBM 2701 Data Adapter Unit, Principles of Operation, Systems Reference Library (A22-6864).

IBM System/360 Component Description, IBM 2702 Transmission Control, Systems Reference Library (A22-6846).

IBM System/360 Component Description, IBM 2703 Transmission Control, Systems Reference Library (A22-2703).

IBM 2740 Communications Terminal, Systems Reference Library (A24-3403).

IBM 1050 Reference Digest, Systems Reference Library (A24-3020).

Printed in U.S.A. Y20-0013-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601