SENTRY

ASSEMBLY LANGUAGE PROGRAMMING UNDER TOPSY

FAIRCHILD

SYSTEMS TECHNOLOGY

A DIVISION OF FAIRCHILD CAMERA AND INSTRUMENT CORPORATION

ASSEMBLY LANGUAGE PROGRAMMING UNDER TOPSY

Part Number: **57500015**
Date Released: May 1977

FAIRCHILD
SYSTEMS TECHNOLOGY
A DIVISION OF FAIRCHILD CAMERA AND INSTRUMENT CORPORATION

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

PREFACE

Assembly language programs may be implemented by the user for execution under the testor oriented operating system (TOPSY). TOPSY files provide such capabilities as data processing, plot generation, and local memory utility functions not easily and/or efficiently handled by a FACTOR test program.

This manual describes procedures, rules, and most efficient implementation for assembly language programs for revision 10.4 and 11.0 of the standard system software. It presupposes an understanding of asembly language programming and the FST Assembly Language.

For more detailed information and cross reference refer to the following manuals.

DESCRIPTION	PUBLICATION NUMBER
FACTOR Programming Language Reference Manual	67095738
Sentry VII User Reference Manual	67095733
Pattern Processor (PPM) User and Programming Reference Manual	67095583
FST-2 Computer Manual	67095701
FST-1/2 Subroutine Library Reference Manual	67095091

INDEX

SECTION 1

INTRODUCTION

Two types of user-written assembly language programs or files may be executed under TOPSY.

The first type of assembly language program is one which is called by a FACTOR test program. These are called Assembly Language Linkage files or ALLINK files. A PPM microprogram is a type of ALLINK file and is described under ALLINK files.

The second type of assembly language program is one which is called by an operator command while no test station is active. These files are called user overlays. These are not to be confused with the system overlays such as DATALOG, CPMAIN, and ANALYSIS which are called into memory as the system demands without the user's need to be aware of them. System overlays are required to be on the disc for TOPSY to execute. User overlays provide optional features.

SECTION 2

ALLINK FILES

ALLINK and PPM microprograms are discussed in this section. The following topics are covered:

a. Calling ALLINK Files

b. ALLINK File Programming Rules & Techniques

    1. Entry and Exit Conditions
    2. Accessing System Constants
    3. Accessing Parameters from the FACTOR Test Program
    4. Accessing System Subroutines
    5. Using the FST-2 ALLINK Storage Buffer
    6. Creating the Coreimage of the ALLINK File
    7. Miscellaneous

c. Optimizing Memory Management to Reduce Test Execution Time

2.1 CALLING ALLINK FILES

An ALLINK file is called via the FACTOR statement, EXEC. A PPM microprogram is called via the FACTOR statement, REXEC. The differences between two statements is that the EXEC statement may have 63 parameters and the REXEC statement may have only 13, and at execution time the REXEC statement causes terminal error 77 if the Pattern Processor Module (PPM) is not part of the system.

EXEC

General Forms:

EXEC file name;

EXEC filename (parameter 1, parameter 2,...parameter n);

'filename' must be an identifier which follows the rules for FACTOR identifiers and is also the name of a coreimage file on disc. The value for n may not exceed 63, i.e. up to 63 parameters may be passed from the FACTOR test program to the ALLINK program.

When EXEC is executed a check is made to see if the ALLINK file is already in memory because of a previous execution. Up to six ALLINK files may be resident simultaneously if they do not overlap. If the file is resident, it is executed immediately. If not, a check is made to see if the file's address on disc is known. If it is, no disc search is necessary. Otherwise, a disc search takes place to find the file on disc under the current job. If not found in the current job, the system job is searched. If not found there, terminal error 60 is issued. Once the file is located on disc it's size and load points are checked. If the ALLINK file will load on top of the operating system, system variable and global storage area, or the working stack, terminal error 61 is issued. If six ALLINK files are already resident or if this ALLINK file will overlap any currently resident ALLINK file, all memory of the previously loaded ALLINK file is forgotten and the new ALLINK file becomes the only resident file. Also, if an FST-1 is use the test program and the ALLINK file must share the same area in memory, so whenever an ALLINK file must be loaded from disc, the pointers to the test program buffer must be re-established and a disc load will occur for the test program when the next FACTOR statement is executed. These error checks and disc accesses require time which slows down testing. Techniques to reduce execution time of ALLINK files is described under the Memory Management section.

Once the ALLINK file is found to be in memory or has been loaded into memory, control is passed to the entry point of the program indicated by the PROC statement.

The parameters passed to an ALLINK file may be global variable, user variable, arrays, array elements, functions, formal parameters, or arithmetic expressions. However, arrays must not be passed to PPM microprograms.

REXEC

General Forms:

REXEC filename;

REXEC filename (parameter 1, parameter 2, ...parameter n), The value of n may not exceed 13, i.e. up to 13 parameters may be passed from the FACTOR test program to the ALLINK FILE. For a description of the parameters passed to a microprogram, see the PPM User's Manual and Programming Reference.

When a PPM microprogram is assembled (RASM command) DMA code which may be executed by the control RAM is generated for each module. An object file of an FST assembly language file is then placed in working storage followed by the DMA code generated by RASM. This object file is actually an ALLINK file which provides the link between the FACTOR test program and the microprogram. The coreimage file which the user creates at the end of the assembly microprogram contains this ALLINK file as well as all the modules. At the REXEC statement, the file is treated as described under EXEC. When the file is loaded each module is also in memory. When control passes to the ALLINK file, it processes the parameters passed to it, determines if the module to be executed is already in the control RAM, and if not loads the control RAM from memory. It then starts the control RAM execution and waits for completion at which time the FACTOR test program continues execution.

2.2 ALLINK FILE PROGRAMMING RULES & TECHNIQUES

2.2.1 Entry & Exit Condition

The entry point of an ALLINK file is at the PROC statement. Rev. 10.4 and later software revisions do not require the PROC to be at the first locations of the program, however, a PROC or data cell or statement which generates assembly language code must precede a BSS statement because storage space fine by a BSS at the start of a file is not actually part of a coreimage file. The load point follows any initial storage defined by BSS. This feature provides faster load time for files under DOPSY when other files will not be loaded below a resident file. Under TOPSY, since it is not know that this area belongs to the ALLINK file the test program or another ALLINK file could overlap the storage area.

The name of the file executed is the name of the coreimage file on disc and not the name on the PROC statement. A normal sequence would be:

```
ENTRY     PROC    Ø
          BRU     START
           '
           '
          data cells and storage
           '
           '
START     EQU     *
           '
           '
          program body
           '
           '
          BRU*    ENTRY
```

The normal return is at CALL + 1.

An error exit is provided to allow the ALLINK file to cause a terminal error and stop the test execution. The ALLINK files may display an error message before exist for more clarification of the terminal error. The terminal error number out put is the contents of the A register plus 100, if the A register is not negative on exist. If it is, the terminal error is 100. This error exit is coded as follows:

```
LDAERRORNUMB
          AOM     ENTRY
          BRU*    ENTRY
```

On entry to the ALLINK file the following conditions are set:

X3	Contains the number of parameters passed to the ALLINK file.
X4	Points to the addresses of the parameters passed to the ALLINK file.
A	Contains O. This is to differentiate from an entry to a user overlay where the A register equals 2.

Any index registers or state switches may be used by the ALLINK file. The operating system saves and restores what is requires. From one entry to another no switch or register is saved for the ALLINK file by the system. This must be done by the ALLINK file. In general, however, an ALLINK file does not known when it will be overlayed in memory so it usually is written so that no data is expected to be there from a previous execution. Also, since the file is not necessarily reloaded at each execution anything expected to be zero but set by a previous execution must be cleared.

If necessary, an ALLINK file can determine if it has been reloaded by defining a flag word as zero and setting it to a one at exit. On entry, if the flag is one the file has not been reloaded; if zero it has been reloaded. If the ALLINK file requires extensive clears or set up procedures which might be avoided by such a flag, then execution time would be reduced.

2.2.2 Accessing System Constants

Space is reserved in GLOBAL for the commonly used constants. These constants may be used by the ALLINK file to reduce the size of the ALLINK file. In other words, there is no need to duplicate the definitions. These constants and their absolute locations are:

```
00740   00000000   D0       DATA   0
00741   00000001   D1       DATA   1
00742   00000002   D2       DATA   2
00743   00000003   D3       DATA   3
00744   00000004   D4       DATA   4
00745   00000005   D5       DATA   5
00746   00000006   D6       DATA   6
00747   00000007   D7       DATA   7
00750   00000010   010      DATA   10B
00751   00000017   017      DATA   17B
00752   00000020   020      DATA   20B
00753   00000060   D48      DATA   48
00754   00000077   077      DATA   77B
00755   00000100   0100     DATA   100B
00756   00000200   0200     DATA   200B
00757   00000377   0377     DATA   377B
00760   00000400   0400     DATA   400B
00761   00001000   01T      DATA   1000B
00762   00001777   01777    DATA   1777B
00763   00002000   02T      DATA   2000B
00764   00004000   04T      DATA   4000B
00765   00010000   010T     DATA   10000B
00766   00020000   020T     DATA   20000B
00767   00037777   037777   DATA   37777B
00770   00040000   040T     DATA   40000B
00771   00070000   070T     DATA   70000B
00772   00100000   0100T    DATA   100000B
00773   00200000   0200T    DATA   200000B
00774   00400000   0400T    DATA   400000B
00775   00600000   0600T    DATA   600000B
00776   01000000   01M      DATA   1000000B
```

```
00777   02000000    02M     DATA   2000000B
01000   04000000    04M     DATA   4000000B
01001   10000000    010M    DATA   10000000B
01002   20000000    020M    DATA   20000000B
01003   40000000    040M    DATA   40000000B
01004   60000000    060M    DATA   60000000B
01005   77777777    0M1     DATA   77777777B
```

To use the system contants the following technique should be used. The example shows how to load the A register with 7.

```
D7      EQU     747B
         '
         '
         '
        LDA     D7
```

2.2.3 Accessing Parameters from the FACTOR Test Program

The address of each parameter passed to an ALLINK file is stored on the working stack in memory. Index register 4 points in front of the first parameter location. Index register 3 contains the number of parameters passed.

Whether the parameters are variable, arrays, or expressions, the working stack contains just one word for each. If the parameter is a variable, the word stack contains the address of the variable. If the parameter is an expression, the value of the expression at EXEC time is what is referenced. The expression is actually analyzed before the EXEC transfer control to the ALLINK file and the result is stored at a location and then that location's address is placed on the work stack.

If the parameter is an array, the work stack contains the address of the location of the array and bit 23 is set to indicate the parameter is an array. Therefore, the test program may pass either an array or a variable for a given parameter. Word 0 of the array is the array size, word 1 is element 1, etc.

The following code will access parameters:

Parameters 1 and 2 are always known to be variables:

```
LDA*    1,X4    GET VALUE OF PARAMETER 1
STA*    2,X4    STORE VALUE IN PARAMETER 2
```

Parameter 3 is always to be an array:

```
LDA*    3,X4    GET ARRAY ADDRESS
LXA     X5      X5 WILL POINT TO ARRAY
LDA     0,X5    GET ARRAY SIZE
LDA     9,X5    GET VALUE OF ELEMENT 9 OF ARRAY
STA     11,X5   STORE VALUE IN ELEMENT 11 OF ARRAY
```

Note that the above examples will not work correctly if the calling FACTOR program passes an array when a variable is expected or vice versa.

Parameter 2 may be an array or a variable:

```
          LDA    2,X4    GET ADDRESS
          BN     ARRAY   BIT 23 IS SET, SO IT'S AN ARRAY
          LDA*   2,X4    GET VALUE OF PARAMETER 1
          BRU    *+4
    ARRAY LDA*   2,X4
          LXA    X5
          LDA    1,X5    GET VALUE OF ELEMENT 1 OF ARRAY
```

2.2.4 Accessing System Subroutines

Input and output routines, the floating point routines, and other utilities may be used by ALLINK files. However, the input/output routines (or any routine with an interrupt address) must not be called directly. Other routines may be called directly, however, if they are already resident as part of TOPSY the ALLINK file can be kept smaller by calling routines indirectly.

The address in memory of each resident routine is stored in low memory locations in GLOBAL. To access these routines indirectly, the user programs BSM* to the address of the memory location of the routine.

The calling sequence which may be used is shown below. The address of TTRIO is always stored at 520B.

```
    TTRIO  EQU    520B
    DCB    DATA   n, test-address, text
            '
            '
            '
           BSM*   TTRIO
           DATA   1
           DATA   DCB
```

The remainder of the calling sequence is identical to that of a direct call.

The I/O driver, floating point, and other miscellaneous system routines and their memory address locations are shown below:

SYSTEM ROUTINE	ADDRESS OF MEMORY LOCATION	
I/O DRIVERS		
CLIO	1232B	(rev. 11 and later)
CRIO	553B	
DISCIO	554B	
LPIO	543B	
MTIO	552B	
TAPIO	541B	
TTPIO	512B	
TTRIO	520B	

FLOATING POINT ROUTINES

FADD	540B
FDIV	521B
FFIX	542B
FFIXS	514B
FFLTS	515B

OTHER SYSTEM ROUTINES

ADRXLATE	1236B
ALPCLR	1244B
BINDEC	513B
CILOAD	510B
CLOSE	537B
ENTRFN	1235B
FIND	511B
GET	517B
GFREC	1205B
INREC	551B
OPEN	547B
PFREC	1204B
PUT	516B
PUTW	1210B
READ	545B
SRCH	1233B
WRITDS	1234B
WRITE	544B

Most of these routines are described in the Subroutine Manual and will not be discussed here. The floating point routines are described in a FACTOR Manual Appendix. Calls to CLIO are discussed in the revision 11 Com Link Manual. Routines not documented elsewhere are described below.

TAPIO is an interface between the user and MTIO. It is only for use in TOPSY. The call is identical to MTIO except that for read and write operations the A and E register must contain the name of the block to be written or read. For write operations a block header record is output before the block to be written. The header record contains this block name and length. For read operations the block header record is read. If it does not contain the same name as in the A and E registers on entry to TAPIO, the following block is skipped and the next block header is read. This process continues until the desired block is found or the end of tape.

If the proper block is not found an error occurs. Thus TAPIO gives block identification and block search features not provided in MTIO.

TAPIO also does more error checking on the tape unit and the DCB. If an error is found there is no return to the calling program and a terminal error is output describing the error found.

ALPCLR is a TOPSY routine provided to allow the ALLINK files to request that the currently resident ALLINK files not be retained so that the space they use is returned to the test program. This will cause a reload of the test program in the full area available. If execution is on an FST-2 CPU, there is usually no advantage in clearing out the ALLINK files unless later in the execution the stack will extend into the ALLINK file and cause terminal error 54. However, this will only eliminate terminal error 54 if the ALLINK file is not called again or if the stack is smaller when the ALLINK file is called again. A subsequent execution of this ALLINK file will force a disc access to reload the file. However, a disc search is not required if no other ALLINK file has been executed on the same station.

The call to ALPCLR is as follows:

```
ALPCLR      EQU     1244B
            BSM*    ALPCLR
```

The A register is destroyed by this call. Return is always at CALL+1.

CILOAD IS USED TO LOAD COREIMAGE files. The calling sequence, if the file is also to be found on disc, is to load the A and E register with the name of the file. CALL+1 is the error return, the file could not be found on disc. CALL+2 is the normal return.

```
CILOAD      EQU     510B
            LDA     FILE
            BSM*    CILOAD
            BRU     ERROR
```

If the location of the file on disc is known, the disc search can be avoided. The A register must be zero, the E register contains the disc address, and index 7 points to an image of the disc directory entry for the files.

```
CILOAD      EQU     510B
            LDA     DO
            LDE     DISC-ADDRESS
            LDX     X7,DIRECTORY-ENTRY
            BSM*    CILOAD
            BRU     ERROR
```

Index 7 points to the image of the disc directory image after a call to FIND or SRCH.

2.2.5 Using the FST-2 ALLINK Storage Buffer

When operating with the FST-2 CPU a buffer is reserved in memory for use by ALLINK files. Assembly language code should not be put in this area. However, it is useful for storage of large amounts of data.

An area of 4K words is always reserved for the ALLINK storage buffer. If a test program fits in memory in less space than reserved for it, the ALLINK storage buffer is expanded to use all the available left-over area. See Figure 2-1.

If the ALLINK storage buffer is not to be used or if less space is required to be reserved, the buffer size may be reduced by a patch to $TOPSY. It may also be expanded if desired. The size must be patched from DOPSY to take effect. Cell 1230B of $TOPSY (called TPALLOC in GLOBAL) contains the constant 4000 or 7640B. If this cell is altered so that less than 48 words remain for the test program, terminal error 54 will result.

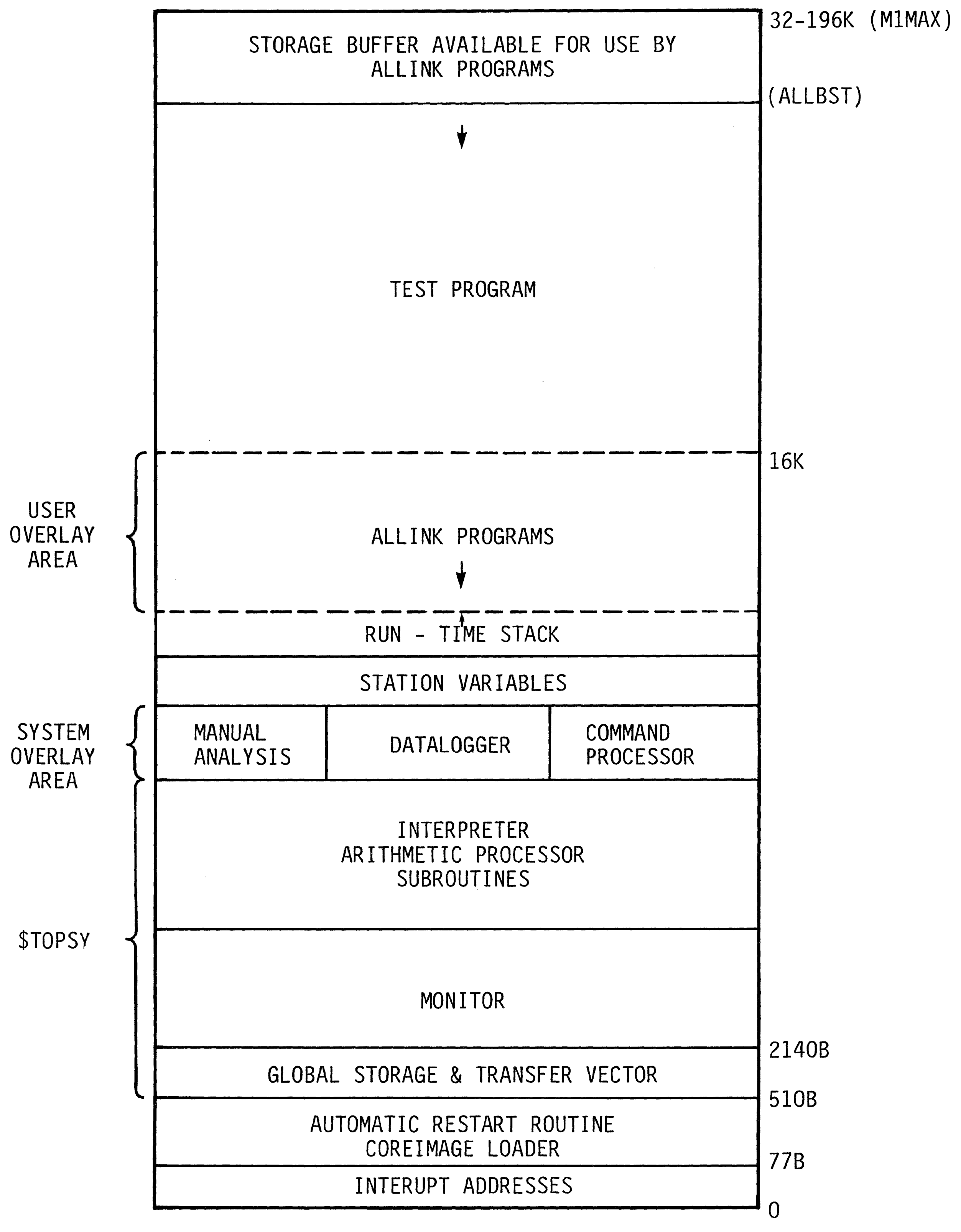


Figure 2-1. FST-2 Memory Allocation in TOPSY

The buffer start address is in ALLBST in GLOBAL, cell 1227B. This cell contains the start of the buffer for that test program execution and will not change as long as only that test program is run. The end of the buffer is the end of the memory and is called M1MAX and is stored at 117B. These should not be altered by the ALLINK file.

To store data or retrieve data that is in the high memory buffer the interrupts must be disabled and the FST-2 CPU placed in the FST-2 mode. When in this mode index registers may be loaded with 18 bit values, thereby allowing the access of data, through an index register, at any memory location. When a desired word has been retrieved or stored, the FST-2 mode must be reset and the interrupts re-enabled. This is the same technique used by the operating system to execute a test program in the upper memory (memory above 16K) while itself operating in lower memory almost entirely in FST-1 mode. While in the FST-1 mode, the memory above 16K may not be accessed and index registers may be loaded with 14 bit may not be accessed and index registers may be loaded with 14 bit values only. If a register contains an 18 bit value can be stored in a data cell in FST-2 mode, the full 18 bit value can be stored in a data cell in FST-1 mode but cannot be retrieved except in FST-2 mode.

The following shows a method of storing data in the ALLINK storage buffer:

```
ALLBST     EQU     1227B
M1MAX      EQU     117B
POINTER    BSS     1
X7         EQU     7
X6         EQU     6
            '
            '
            '
           LDA     DATA
           BSM     STORE
           BRU     DONE          BUFFER IS FULL
            '                    WORD IS STORED
            '
            '
STORE      PZE     0
           IDA
           DATA    07000612B     SET FST-2 MODE
           LDX*    X7,POINTER
           LDX*    X6,M1MAX
           ATX     X7,1          AT END OF BUFFER?
           BG      *+4           YES
           STA     0,X7          NO, STORE DATA
           STX     X7,POINTER SAVE X7
           AOM     STORE         INCREMENT RETURN ADDRESS
           DATA    07000611B     SET BACK TO FST-1 MODE
           IEN
           BRU*    STORE
```

The data to be stored is in the A register when the subroutine is called. The interrupts are disabled and then the CPU is set into the FST-2 mode so the memory above 16K may be accessed. This instruction may be coded with the data cell as shown or an opcode may be defined so a mnemonic may be used. Index registers are loaded with the buffer start and end points and a check is made to see that the buffer is not exhausted. If it is, exit is at CALL + 1. Otherwise the data is stored in memory, the index register saved, and the return address set to CALL + 1. Otherwise the data is stored in memory, the index register saved, and the return address set to CALL + 2. The CPU is set back to FST-1 mode and the interrupts enabled. This subroutine could be used to retrieve data from the buffer by changing the 'STA' statement to a 'LDA' statement. It is up to the user to determine that the ALLINK files called by a test program do not conflict in their use of the storage buffer.

2.2.6 Creating the Coreimage of the ALLINK File

After an object file has been generated for an ALLINK file, it is necessary to create a coreimage type file. Care should be taken in selecting the origin point for the ALLINK file since up to six ALLINK files may be resident in memory at the same time if none overlap. It is good practice not to set the origin point in the ALLINK file. Either have no 'ORG' statement or ORG at zero. The origin point can then be established when the coreimage file is created. It is possible to origin an assembly language program higher than the ORG point in the program but not lower, i.e. the CREATE COREIMAGE command will not accept a negative number.

To determine that the ALLINK files called by a test program do not overlap, the coreimage map generated when the coreimage file is created must be examined. The number specified in the CREATE command indicates the load point of the program, the first cell to be used by the program. The number printed on a line by itself directly after the CREATE command and any control cards used is the next available word in memory after the coreimage file. In other words, it is the end point + 1 of the file and is the location at which another ALLINK file may be origined if ALLINK files are to be packed closely together. If it is the only or highest loading ALLINK file, this number indicates how much space is between the top of the ALLINK file and the end of memory available for ALLINK files, 37776B. Since this is wasted space which in the FST-1 CPU is needed by the test program or in the FST-2 CPU may be needed by the stack, it is best to origin the file so that there is as little wasted space as possible.

Below is a typical example of a coreimage create command and the coreimage map created:

```
// CREATE "IBUS'   OVLY COREIM   '=BUS'        35200B
37775
IBUS        N   35200
IBIO            37461
IBUS        IS ENTRY PNT
```

37775 is the address of the next available cell above 'IBUS'.

No definite low point may be specified for ALLINK file since the size of the operating system varies with each revision, and with rev. 11 varies with the configuration selected (MTIO, DEBUG, CLIO, none). Also the number of stations initialized changes the space available as 73 words are reserved for each initialized station.

2.2.7 Miscellaneous

If an ALLINK file does I/O to the VKT, the user may wish to ensure that the prompting character will be output again at the end of the test sequence. To do this the cell CPBZSW, at location 1142B should be cleared.

```
CPBZSW     EQU     1142B
            '
            '
            '
           LDA     DO
           STA     CPBZSW
```

If FACTOR I/O or datalogging to the VKT is done by the test program the operating system clears this flag.

A system constant called RPLEN in GLOBAL at location 1243B is set to 4096. Any ALLINK file larger than this constant is automatically not retained in memory after its execution and the area used is returned to the test program. The next time the ALLINK file is executed, it will have to be reloaded from disc, although a disc search will not take place if no other ALLINK file has been executed on the same station. If the user wishes some other constant to be the maximum size of retained ALLINK files, cell 1243B may be patched in DOPSY or altered from TOPSY with DEBUG. Note that any change to the global storage area during the first entry to TOPSY following an INIT will cause the change to become a permanent change to $TOPSY on disc.

2.3 OPTIMIZING MEMORY MANAGEMENT TO REDUCE TEST EXECUTION TIME

Test execution time can be reduced by careful memory management which reduces the number of disc accesses required. Disc accesses occur when a test program does not fit in memory and must be paged and when an ALLINK file must be found on disc or loaded into memory.

In the FST-1 CPU, the memory above the operating system, and station variables is shared by the test program the ALLINK files, and the working stack. See Figure 2-2. The working stack is the area where the system stores the user variables arrays, parameters passed to ALLINK files, and subroutine return points. It is built above the operating system and system globals as the test program is executed and it's size depends on the test program. Each variable used or declared, each array element declared, and each nested subroutine call or block increases the stack's size, reducing the area available for the test program and ALLINK file.

When a test program is larger than the memory available all of the test program cannot reside in memory at once. It is then necessary to bring the test program into memory in segments. Once piece of the test program is brought into memory and executed and then another segment is brought in on top of the previous piece. This technique is called paging. When an ALLINK file is executed, it too reduces the area available for the test program. As the area for the test program is reduced the number of disc accesses required to page the test program increases.

With the FST-2 CPU, the test program is stored in the upper memory, memory above 16K, and the stack and ALLINK file do not cause disc accesses on the test program. If the test program fits in memory, no disc accesses for the test program occur after it is first loaded into memory. If it does not fit the test program is paged in the maximum area available between 16K and, generally, 4K from the end of memory.

For either FST-1 or FST-2 CPUs, when an ALLINK file is executed the first time it must be found on the disc. This requires a search of the disc directory which itself may take several disc accesses, in addition to the access for the load from disc to memory. Once an ALLINK file is loaded into memory, however, it will be retained and no disc access will be caused by it's execution until the ALLINK file is removed from memory. The following occurrences remove an ALLINK file from memory. (This does not mean that memory is actually cleared or that the ALLINK file is written back to disc. It simply means the area where the file was is now used by some other program).

1) Executing a test plan with a different name.
2) Executing a user overlay even if it is not loaded on top of a resident ALLINK file or test program.
3) Going to DOPSY.
4) Executing more than six ALLINK files by one test program. The seventh file becomes the only resident file.
5) Executing an ALLINK file that overlays some portion of a resident ALLINK file. The lastest file becomes the only resident file.
6) Executing an ALLINK file which is larger than the system constant, RPLEN. See III G.

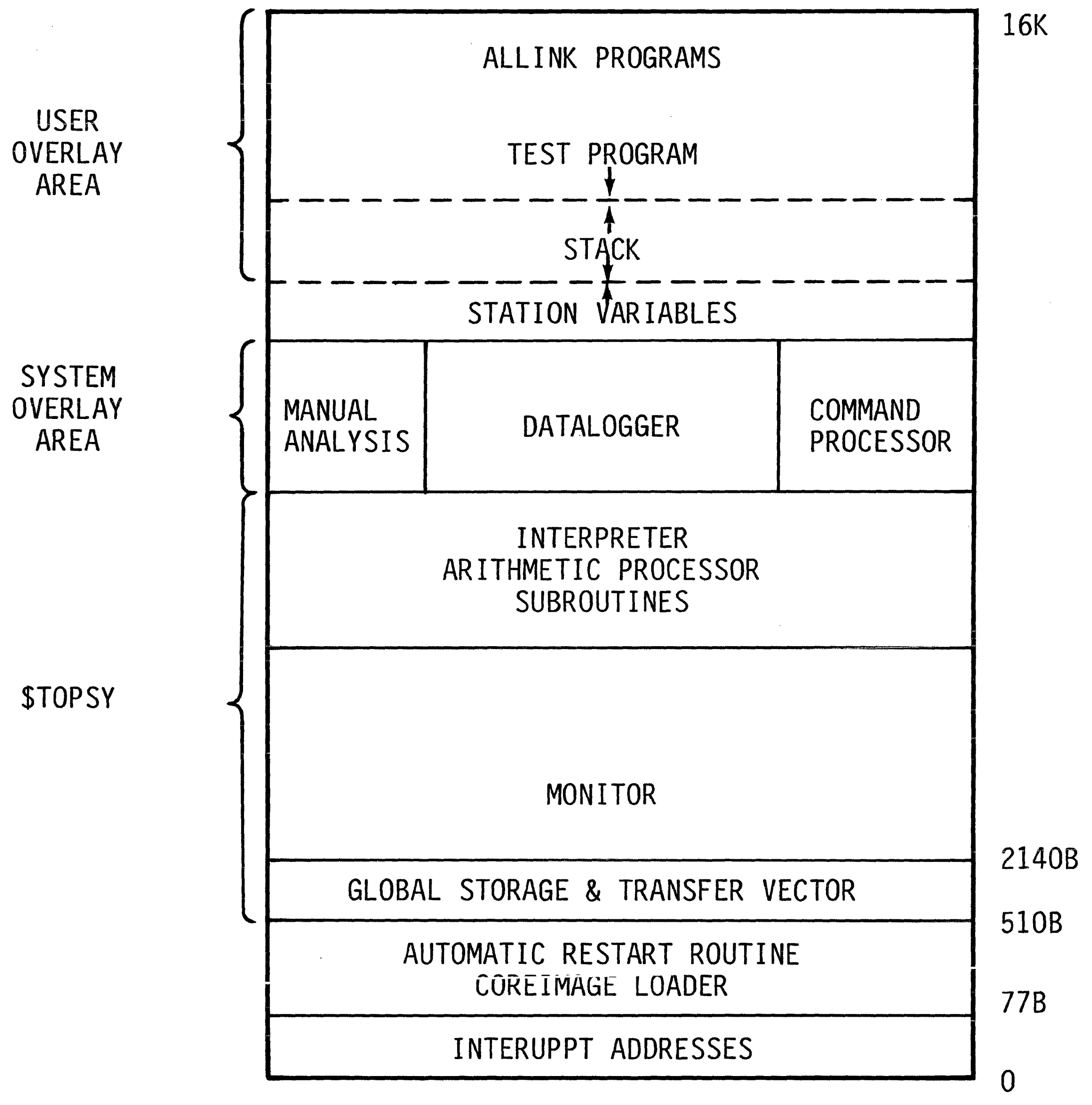


Figure 2-2. FST-1 Memory Allocation in TOPSY

7) Executing an ALLINK file which calls ALPCLR to force its own removal from memory.

Items 1 and 3 cause a disc access for the test program for both the FST-1 and FST-2 CPU. The other items cause a disc access on the test program is the FST-1 CPU is in use so that the test program will use all the available memory.

Item 6 and 7 describe times when the user can chose to have the ALLINK files removed to gain more memory for the test program on the FST-1 CPU.

Once files have been "removed" from memory they will have to be reloaded from the disc at the next EXEC statement. Only the address on disc of the last file executed is saved so once a file is removed a disc search is required for all but the last file.

Given this background information these points highlight the way to decrease the number of disc accesses and increase test rate on the FST-1 CPU.

1) Origin the ALLINK files as high in memory as possible to keep the maximum space for the test program.

2) Execute the same test program on all stations because the resident ALLINK files will not have to be loaded.

3) Minimize the number of executions of user overlays and returns to DOPSY since the next test program execution requires the test program to be reloaded and ALLINK files to be found on disc and reloaded.

4) Minimize the number of ALLINK files executed by a test program since the first execution of each requires a disc search. Where possible combine the ALLINK files needed.

5) Reduce the disc search time when a search is required by storing the ALLINK file in the current job and as near the beginning of the disc as possible. To do this, see Application Note 34, Minimize Access Time for Assembly Language Overlays.

6) If more than one ALLINK file is to be executed repeatedly by a test program, origin the files so that they do not overlap.

7) Determine the most efficient origin point for each ALLINK file called if more than one is to be resident ALLINK file and not overlapping any other file, the boundaries of the test program buffer are not changed so no disc access occurs. Therefore, it may be most efficient to origin the first ALLINK file executed at the lowest point. However, if the other ALLINK files are executed much later in the program the space may have been removed from the test program long enough to cause disc accesses that would not have been necessary.

8) Determine that the disc access time required for the test plan to page in a small area is not longer than the time required if the ALLINK files are removed from memory and must be loaded (and searched for if more than one is executing) each time they are loaded. If test time may be enhanced by returning the area to the test program the ALLINK file may call ALPCLR (See II D), or RPLEN (at 1243B) may be reduced (See III G).

Similar points to reduce execution time may be followed if an FST-2 CPU is used, except that the key factor is not maintaining the test program area but reducing ALLINK search and load times. These points are important for execution speed on FST-2.

1) Execute the same test program on all stations because the resident ALLINK files and the test program (if it fits in memory) will not have to be reloaded.

2) Minimize the number of executions of user overlays and returns to DOPSY since the next test program execution requires the test program to be reloaded and ALLINK files to be found on disc and reloaded.

3) Minimize the number of ALLINK files executed by a test program since the first execution of each requires a disc search. Where possible combine the ALLINK files needed.

4) Reduce the disc search time when a search is required by storing the ALLINK file in the current job and as near the beginning of the disc as possible. To do this, see Application Note 34, Minimize Access Time for Assembly Language Overlays.

5) If more than one ALLINK file is to be executed repeatedly by a test program, origin the files so that they do not overlap.

SECTION 3

USER OVERLAYS

User overlays are discussed in this section. The following topiscs are covered.

- Calling User Overlays
- Programming Rules and Techniques
  1) Entry Conditions
  2) Exit Conditions
  3) System Constants and Subroutines
  4) Using the FST-2 ALLINK Storage Buffer
- CPMAIN Subroutines and Re-entry Points
  1) COMPROC
  2) CPREAD
  3) CPWAIT
  4) CPECHO
  5) CPRINT
  6) CPERR
  7) OVLYERR

3.1 CALLING USER OVERLAYS

A User Overlay may be called while in TOPSY or in analysis mode in TOPSY whenever no station is active.

When the command entered is not one of the system commands, the system job on the disc directory is searched for a coreimage file with the command name preceded by a colon (:). If found, the file is loaded into memory and executed. If not found the error message 'COMMAND?' is output.

The system assumes that the User Overlay is origined at a location which will not overlap the operating system or the stack if the overlay is executed during a station pause. The overlay should be origined as high in the first 16K of memory as possible, in the test program and ALLINK file area. As of revision 10.2, the user overlay does not need to force the test program to be reloaded. Revision 10.4 and revision 11 handle all memory management required. If an FST-1 CPU is in use, the test program, if any is executing, and any ALLINK files which were resident are reloaded into memory at the next test execution. If an FST-2 is in use, only the ALLINK files are reloaded from disc. The reload occurs whether or not the files were actually lost by the execution of the user overlay.

Cell 1160B in GLOBAL may be used to determine what revision software a user overlay is operating under if the overlay is to execute with several different revisions. The revisions are denoted as follows:

positive value	-	Rev. 9.6
0	-	Rev. 10, 10.1, 10.2
-3	-	Rev. 10.3
-4	-	Rev. 10.4
-110	-	Rev. 11

When creating a coreimage file which will be used as a user overlay the first character of the file name must be a colon.

```
CREATE    ':FILE'   COREIMAGE    '=FILE'        32000B
```

The user overlay ':FILE', is executed by the following TOPSY command.

```
FILE    (optional parameters)
```

The following names are names of system commands and may not be used as names of user overlays (i.e. not the first word of a command). However, these names may be used as modifiers.

ALTER	LOAD	RESET
ANALYSIS	LOOP	RESTART
CLEAR	MAGT	SET
CLOSE	MANUAL	SN
CONTINUE	MEASURE	START
DATALOG	MODIFY	STOP
DEBUG	NOTE	SWITCH
DISPLAY	OPEN	SYNC
DOPSY	OVERRIDE	TITLE
ETC	PAUSE	TOPSY
GLOBAL	READ	VAR
		WRITE

3.2 PROGRAMMING RULES AND TECHNIQUES

3.2.1 Entry Conditions

The entry to the user overlay is at the PROC statement. The PROC does not have to be at the first location of the program and BSS statements may be at the start of the program. As long as the user verifies that the overlay does not overlap the stack (if executing at a station pause) or the operating system, and the file will not be executed as an ALLINK file, putting all BSS statements before any other statement (except ORG) will reduce the load time of the file.

Parameters are passed to the user overlay in registers and in a buffer area at the start of the command processor, CPMAIN. (This is the TOPSY system overlay which handles system input and command decoding.) Information is also passed back to CPMAIN through this buffer area if a CPMAIN subroutine such as COMPROC is called.

Information available to the user overlay is as follows:

Register	Contents
A	The flag indicating the source of the call. =2 call is a user overlay request. =0 call is by EXEC, a test program request.
E	Contents of the flag COMCOM plus any output device specified is in E. This cell indicates the parameters input on the command. Contents of E register: bit 23-17 set on 15 a string was entered 13 'ON' specified 12 'OFF' specified 11 A number specified 10 A 2nd number specified 6 CLO specified 4 MTW specified 3 LP specified 1TTP specified
X1	Pointer to the station variables for the station indicated in the input command. The station variables are listed below.
X2	Pointer to the CPMAIN buffer containing command information and subroutine locations.
X7	Pointer to the PMF for the system command. The current pointer is at zero.

Station Variable Table

Information in the system's station variable table pointed to by index register 1 on entry is as follows:

word (octal)	contents
0-10B	test program PMF
11B	DLCNTRL
16B	DCDLY
17B	DLFREQ
20B	DLSKIP
21B	LGMSK (datalog request)
24B-43B	Datalog title
46B	MACTRL (analysis request)
52B	SN
54B	TPFIT (FST-2 test plan fits flag)
55B-63B	ALLINK directory image
64B	VALUE (0)
65B	SWITCH (1)
66B	(DUMMY GLOBAL - accessed through FACTOR as TIME)(2)
67B-112B	GLOB1 - GLOB20 (3-23)

The system variables are in locations 0-54B in rev. 10.4 and rev. 11, however these sometimes are expanded with new revisions. To aid access of the GLOBALS regardless of the revision, a system constant is set with the size of the system variables plus the ALLINK directory image area. With index register 1 pointing to the station variables as on entry GLOB1 could be accessed as follows:

```
        LDA             67B, X1
```

However, the method to access the globals which will not be changed if a later revision changes the number of system variables is this:

```
VARSIZ  EQU     1215B
        LAX     X1      START OF TABLE
        ADD     VARSIZ  INCREMENT OVER VARIABLES
        LXA     X1
        LDA     3, X1   GET VALUE OF GLOB1
```

CPMAIN Buffer

The buffer area of CPMAIN, pointed to by index register 2, contains data cells with information from the input command, addresses of subroutines in CPMAIN which may be called by the user overlay, and return points to CPMAIN for error message output.

Theses are the data calls for transfer of information between the user overlay and the CPMAIN subroutine COMPROC. The use of this subroutine and the detailed description of the data cells' contents are descrived under CPMAIN Subroutines.

Word (decimal)	Contents	Description
5	COMCOM	Contains the bit image of the command as it is built by COMPROC. It does not contain the output devices which are in the E register on entry.
6	COMIO	Contains the input and out units specified in the input command.

9	NUMPA2	Contains the first number if two numbers are entered.
10	TABBOT	Start of the user's command table.
11	TABTOP	End of the user's command table.
15	STAT	Station ID. This is set to -1 if no station is entered.
20	NUMPAC	Contains the first number if one number entered, the second number if two entered.
24	STRNG1	First word of string.
25	STRNG2	Second word of string.

These calls may be accessed through index register 2 as follows:

```
COMIO      EQU    6
           LDA    COMIO, X2   GET I/O DEVICE SPECIFIED
```

These are the CPMAIN subroutines which may be called by the user overlay. Their usuage is described under CPMAIN Subroutines and Re-entry Points.

Word (decimal)	Contents	Description
1	COMPROC	Decodes the user command into a 24 bit representation.
12	CPREAD	Inputs a record into the command buffer.
18	CPRINT	Outputs a record.
26	CPWAIT	Waits for completion of a read TTK. Called after a call to CPREAD.
27	CPECHO	Echoes an input command if not from TTK. Called after a call to CPREAD. (Revision 11 only.)

These subroutines are called indirectly as follows:

```
CPWAIT     EQU    26
           BSM*   CPWAIT, X2
```

The following are addresses of entry points to CPMAIN for output of error messages. They are described under CPMAIN Subroutines and Re-entry Points.

Word (decimal)	Contents	Description
2	CPERR	Outputs one of the standard system error messages.
21	OVLYERR	Outputs an error message defined by the user.

These error message routines are accessed as follows:

```
OVLYERR    EQU    21
           BRU*   OVLYERR, X2
```

3.2.2 Exit Conditions

On exit the user overlay must restore the index registers to their state at entry.

The normal exit from the user overlay is at CALL + 1.

An error exit is provided at CALL + 2. If this exit is taken the standard TOPSY error message "DUPL./MISSING PARM." is output.

The user overlay may also be exitted by a branch to the CPMAIN routines CPERR and OVLYERR which output error messages. Also, if COMPROC is called to decode an input command there may be no return if COMPROC finds an error. These conditions are described in CPMAIN subroutines.

3.2.3 System Constants and Subroutines

The system constants and system subroutines available to ALLINK files are also available to user overlays and are accessed in the same way.

3.2.4 Using the FST-2 ALLINK Storage Buffer

The FST-2 ALLINK storage buffer may also be used by user overlays. The user must determine that there is no conflict between the use of this buffer by ALLINK files and user overlays.

3.3 CPMAIN SUBROUTINES AND RE-ENTRY POINTS

3.3.1 COMPROC

Purpose

To scan an input command in BUF4 and create a word code representing the command and modifiers. The station number, two numbers, a string, and I/O devices specified in the command are saved in the CPMAIN Buffer which can be accessed via index register 2. By giving information to COMPROC through a table, COMPROC can require one of several modifiers, prohibit all but one of a group of modifiers, and set a bit for each modifier selected even when none are required.

Calling Sequence

This routine is called by a user overlay when it is desired to obtain more information from the input command or when the user overlay reads a record and then wishes to scan it.

```
COMPROC         EQU    1
TABBOT          EQU    10
TABTOP          EQU    11
*       NOTE    -   INDEX 2 POINTS TO CPMAIN
                LDA    address of beginning of table
                STA    TABBOT, X2
                LDA    address of end of table
                STA    TABEND, X2
                BSM*   COMPROC, X2
```

Return

COMPROC returns control at CALL + 1 unless the input command is incomplete or has conflicting modifiers as defined by the user's table. When such an error is found in the command an error message is output and control is not returned to the caller.

Output Data

```
COMCOM    EQU    5   Contains the bit image of the command.
COMIO     EQU    6   bit  23-12     input device
                     bit  11-0      output device
NUMPA2    EQU    9   First number if two numbers entered.
STAT      EQU   15   Station ID. This is negative if no station is entered.
NUMPAC    EQU   20   First number if one entered, second number if two
                     entered.
STRNG1    EQU   24   First word of string (command other than TITLE,
                     NOTE).
STRNG2    EQU   25   Second word of string (commands other than TITLE,
                     NOTE).
```

Input Command

A command is made up of identifiers, strings, and numbers which are defined as follows:

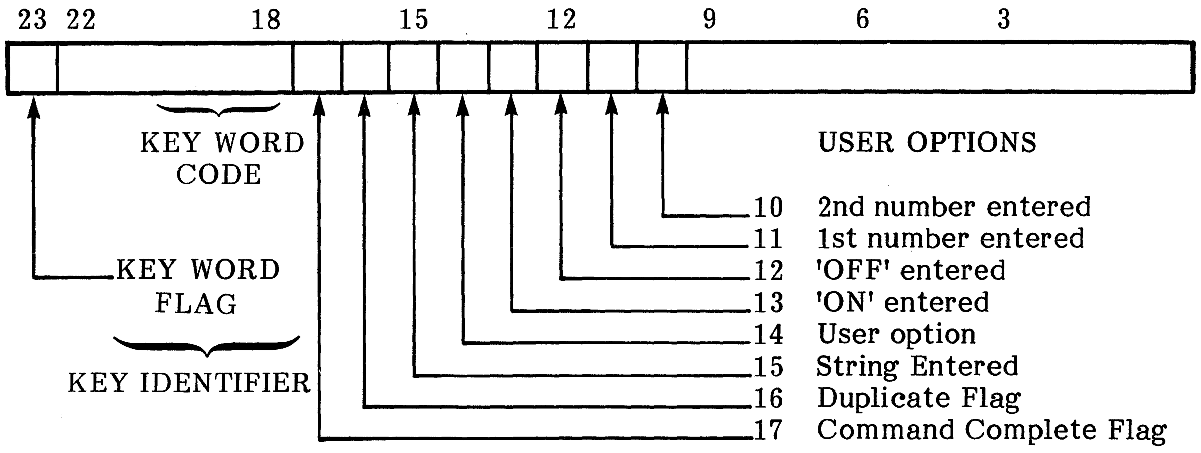
identifiers: A name preceded by an alpha character and containing only alphanumeric characters. The first identifier must be the keyword of command, otherwise an error message will result. The following identifiers are modifiers or options which may be defined in the user's table as required, optional, or allowed only when another modifier is not allowed. All identifiers are truncated to four characters. The remaining characters are ignored.

string: A string is a series of any characters except blanks enclosed in single quotes. Strings are truncated to six characters and stored in STRNG1 and STRNG2.

number: Any decimal, octal, or scientific notation type number may be entered. A decimal point, or plus or minus sign indicate to COMPROC that a number is input. If these special characters are not followed by a number an error message results.

Input Command Tables

The command table consists of any number of two word entries. The first word is the first four characters of the identifier and the second word is the bit code and flags to COMPROC which define the identifier and the second word is the bit code and flags to COMPROC which define the identifier. The second word (definition word) is defined as follows:



The command table must contain at least one identifier defined as a keyword, the first identifier to be entered in the command statment. A keyword is defined by setting bit 23 in the definition word. COMPROC scans the input command and finds the first identifier. It is then comparred against the input command table. If an entry in the table is found, with bit 23 set in word 2, which matches the command's first identifier then word 2 is stored into COMCOM, the cell which will contain the composite command image.

If no match is found or a blank line is entered the error message 'COMMAND?' is output and no return to the calling program takes place.

For user overlays the keyword code may be 21B to 37B or for the keyword the entire keyword identifier is 61B to 77B.

Once the keyword has been found, additional identifiers in the command are modifiers or options. One identifier will be treated as a modifier for a command if it passes a series of checks. Once it passes, the contents of word 2 of the command table are ORed into the flag word COMCOM where the command image is being formed. Bits set in the user filed of word 2, bits 0-9 and 14 indicate to the calling program which modifiers have been entered. If the identifier does not pass the checks it is treated as a noise word and ignored. In this case the table scan continues until a match is found or the table is exhausted. The the input command is scanned for the next entity.

These are the checks made on an identifier after the keyword has been found:

1) The first four characters of the identifier must match the first word of the table entry in the command table. Once an identifier match is found the remaining checks are made using the corresponding definition word table entry.

2) The modifiers may not be defined as keywords, i.e. bit 23 set will cause the table entry to be skipped and the table search to continue in case there is another table entry which might match the identifier. This allows keywords to be used as modifiers even though only one word of command may be the keyword. (For example, DATALOG MEASURE).

3) If the keyword code (bit 18-22 of the definition word) is zero the corresponding modifier is allowed on any command and the definition word is ORed into COMCOM. For example in the system tables 'ON' and 'OFF' are allowed in any command so bits 18-22 are zero. The definition word of 'ON' has bit 13 set and the definition word of 'OFF' has bit 12 set so that if these modifiers appear in any system command or command processed by a user overlay the corresponding bit will be set in COMCOM.

4) If the keyword code of the identifier is not zero then it must be the same as bits 18-22 of the keyword identifier already stored in COMCOM. This allows defining more than one keyword in a table with unique modifiers for two keywords where different bits are to be set depending on the keyword. (For example, FCT appears in the system's tables twice, with the DATALOG keyword code and the DISPLAY keyword code because for DISPLAY bit Ø is to be set in COMCOM and for DATALOG bit 3 is to be set. Also the identifier 'SAVE' appears in the table with the 'LOAD' keyword code so that a bit will be set if 'SAVE' is entered in a LOAD command but if it is entered in any other command it will be ignored).

5) Next the duplicate flag bit (bit 16) is checked. If it is set in both the modifiers definition word and in the COMCOM being formed, then two incompatible modifiers have been entered and the command decoding halts and the error message 'DUPL./MISSING PARM.' is output. No return is made to the calling program. (For example, in the OPEN command both DIF and DOF may not be specified in a command so bit 16 is set in the definition word for each. Once 'DIF' has been entered bit 16 is set in COMCOM so if 'DOF' also appears the error occurs).

If an identifier does not match any entry in the user's command table, the system command table is checked for identifiers which may be in any command, 'ON', 'OFF', 'STAT.' 'ON' causes bit 13 to be set. 'OFF' causes bit 12 to be set. No duplicate check is made so the calling program must verify that both have not appeared in the command. If a station identifier has been entered, the station ID translated to a value between 0 and 15 is stored in STAT 9 cell 15 of CPMAIN). STAT equals -1 if no station identifier is entered. The calling program must verify that a station number has been entered if one is required and that it is an assigned station.

At the first entry to a user overlay, this has already been done and index register 1 points to the station variables. If the user overlay is calling COMPROC to scan another command where the station number might be different, this code will verify that the station has been assigned and leave index register 1 pointing to the station variables.

```
STHVAR    EQU    146B        STATION VARIABLE TABLE ADDRESSES
STAT      EQU    15          STATION ID
CPERR     EQU    2
X2        EQU    2
X1        EQU    1
D9        DATA   9
          DATA   10
           '
           '
           '
          LDA    STAT,X2     GET STATION ID
          BN     ERR10       NOT ENTERED
          LXA    X1
          LDA    STHVAR,X1   ADDRESS OF STATION VARIABLES
          BZ     ERR9        STATION NOT INITIALIZED
          LXA    X1          X1 POINTS TO VARIABLES
           '
           '
           '
ERR9      LDA    D9          OUTPUT MESSAGE
          BRU*   CPERR,X2    "STATION UNASSIGNED"
ERR10     LDA    D10         OUTPUT MESSAGE
          BRU*   CPERR,X2    "STATION MISSING"
```

If an identifier has still not been matched I/O tables are checked. If the identifier does name a device in the table a check is made. No more than one input device and one output device are allowed. If more than one of either is entered, the error message 'DUPL./MISSING PARM.' is output and no return is made to the calling program. Also, if 'CLO' is specified and the com link driver is not part of $TOPSY the error message '$TOPSY NOT CREATED WITH CLIO' is output and no return takes place. If an I/O device does match, the device code is ORed into COMIO (word 6 of CPMAIN). The codes are:

TTK	1000B	TTP	1B
CR	2000B	LP	3B
MTR	4000B	MTW	4B
DIF	5000B	CLO	6B

The input field is bits 12-23. The output field is bits 0-11.

If a number is sensed in the input command the number is decoded into a floating point number unless appended by an asterisk in which case bits 0-18 contain the binary value and bits 19-23 contain the number of bits entered. The first number entered is stored in NUMPAC and bit 11 is set in COMCOM. When the second number is sensed, NUMPAC is stored into NUMPA2 and the second number is in NUMPAC and bit 10 of COMCOM is set. SAVE + 9 (in GLOBAL location 1017B) contains the octal value of the last octal number entered, i.e. it is not floated. If an illegal number is entered such as 9B or 1.37B it is ignored.

If a string is entered bit 15 is set in COMCOM. If the string does not have an end quote or contains blanks 'BAD NAME' is output and no return to the calling program takes place.

At the end of the command record COMCOM is checked again. If bit 17 is not set some required modifier has not been entered so the error message 'DUPL./MISSING PARM/' is output and control is not returned to the calling program. If only a keyword is required in a command, the keyword's definition word must have bit 17 set. However, if one modifier of several is required, but 17 is zero in the keyword's definition word and set in each modifier's word. (For example, in the OPEN command 'DIF' or 'DOF' must be entered but no both. As above, bit 16, the duplicate bit, is set in each to prohibit both being entered and bit 17 is zero in the 'OPEN' definition word and on for 'DIF' and 'DOF' so that if either is entered bit 17 (as well as another bit in the user field to determine which was entered) is set so that either one causes the command to be complete).

This is an example of how the operating system uses COMPROC to decode the OPEN command. The full table entries for the OPEN command and its modifers are:

```
DATA   'OPEN',     61000000B
DATA   'DIF',      21600001B
DATA   'DOF',      21600002B
```

When the command is decoded and passed to TCMD2, the suboverlay knows that it is the OPEN command because this is the only command code of '21B' which would be passed to it. It knows the command is complete because bit 17 must be set and only 'DIF' or 'DOF' can complete it; and that both 'DIF' and 'DOF' could not have been entered. TCMD2 will determine which was entered by checking bits 0-1. Then if 'DIF' was entered bit 15 is checked to make sure a string was entered. No string entered causes an error message and otherwise the file name is in words 24 and 25 of CPMAIN.

The following table shows how the bit fields in the definition word of a modifier are used to describe the use of a modifier. In this case neither bit 16 or 17 is set in the keyword code.

duplicate bit (16)	complete bit (17)	user options	description
1)		X	The modifier is allowed but is not required. It's use does not restrict the use of another modifier.
2) X		X	No other modifier may be entered which also has the duplicate bit set. This modifier is disallowed if the duplicate bit has been set in the keyword definition word.
3)	X	X	The modifier will complete the command.
4) X	X	X	This modifier will complete the command and no other modifier with the duplicate bit set may be entered.

3.3.2 CREAD

Purpose

To read a record from the PID or a device indicated by the A register into BUF4 or BUF4 plus some displacement. BUF4 is at location 657B in GLOBAL and is the record scanned by COMPROC.

Calling Sequence

```
CPREAD     EQU    12
*NOTE - X2 POINTS TO CPMAIN
           LDA    DEVICE (OR ZERO)
           LDE    PROMPTING CHARACTER + 1
           BSM*   CPREAD,X2
           DATA   Ø
            -     return
```

Return

Return is always at CALL + 2.

Input Data

A register - The A register contains the device code for the device to be read. If zero, the PID is read. (The PID device code is in bits 5-7 of M1CTRL). The device codes are as follows:

0	=	PID
1	=	TTK
2	=	CR
4	=	MTR
5	=	DIF

Any undefined device code causes a read TTK.

E register - The E register contains the prompting character and the opcode (1) for a read TTK if the input device is the VKT. The prompting character is in the upper 6 bits of the word. For example, to issue a prompting character of '>' to the VKT this code should be used.

```
                  PROMPT     DATA   '>'+1
                             LDE    PROMPT
```

CALL + 1 - CALL + 1 must contain the displacement length from BUF4 for the input data to be read into. Most input records to be read for user overlays are read into BUF4 because this is the buffer which COMPROC scans to decode input commands. To do this, CALL + 1 contains zero.

Output

If the input unit is the VKT keyboard the command input is echoed (by TTRIO) as it is entered character by character on the VKT screen. If the input command is not from the VKT keyboard it will not be displayed on the screen. To display it with revision 11 and later, call CPECHO. (See CPECHO). On revision 10.4 determine that the input device is not the VKT and echo the command as follows:

```
     TTPIO   EQU    521B
     BUF4    EQU    657B
     DCB     DATA   20,BUF4
              '
              '
              '
             BSM*   TTPIO   PRINT   BUF4
             DATA   4
             DATA   DCB
             BSM*   TTPIO   WAIT
             DATA   0
             BRU    *-2
```

Description

If the input device is other than the VKT, CPREAD calls INREC to read the input record. INREC always performs a wait before return so that the input record is in BUF4 (plus the displacement) on return.

If the input device is the VKT, TTRIO is called directly. No wait for completion occurs. (See CPWAIT).

On revision 11 or later software an appropriate sequence to call CPREAD, echo a command if the input device is not the VKT, and wait for the device if it is the VKT is as follows:

```
     D0          DATA   0
     CPREAD      EQU    12
     CPWAIT      EQU    26
     CPREAD      EQU    27
     PROMPT      DATA   '>' +1
     X2          EQU    2
                  '
                  '
                  '
                 LDA    D0          INPUT FROM PID
                 LDE    PROMPT
```

```
                BSM*    CPWAIT,X2  WAIT IF TTK IS INPUT DEVICE
                BSM*    CPECHO,X2  ECHO IF TTK NOT INPUT DEVICE
                                   RECORD IS INPUT READY TO PROCESS
```

3.3.3 CPWAIT

Purpose

To wait on the VKT input after a call to CPREAD. This must only be called after a call to CPREAD. No action occurs if the input device last used by CPREAD was not the VKT.

Calling Sequence

```
    CPWAIT      EQU     26
                BSM*    CPWAIT,X2
```

Return

Return is always at CALL + 1.

Input/Output Data

None

Description

CPWAIT checks the device last read by CPREAD. If it is not the VKT, no action occurs. If the last device read by CPREAD is the VKT a wait occurs until the device is not busy.

3.3.4 CPECHO

CPECHO is only available on revision 11 and later software.

Purpose

To display on the VKT the input command read by CPREAD. This must only be called after a call to CPREAD. No action occurs if the input device last used by CPREAD was the VKT.

Calling Sequence

```
    CPECHO      EQU     27
                BSM*    CPECHO,X2
                 -      return
```

Return

Return is always at CALL + 1.

Output

The input record last read by CPREAD is displayed to the VKT unless the VKT was the input device.

Description

CPREAD checks the device last read by CPREAD. If it is the VKT, no action occurs. If the last device read by CPREAD Is not the VKT the input record is displayed to the VKT and a wait until the VKT is not busy takes place. The input record may have been read into BUF4 or BUF4 plus any displacement.

3.3.5 CPRINT

Note that references to the com link refer to revision 11 or later of the software.

Purpose

To output a record to the TTP and also to the com link or to the line printer if CLO or LP is specified in a command processed by COMPROC, or to the line printer if it is the POD.

Calling Sequence

```
CPRINT     EQU    18
DCBADDR    DATA   DCB
DCB        DATA   n, test-address, text
            '
            '
            '
           LDA    DCBADDR
           BSM*   CPRINT, X2
            -     return
```

Return

Return is always at CALL + 1 unless a com link error occurs. In this case the error is printed on the VKT and no return occurs.

Input Data:

TTPOUT - Cell 29 of CPMAIN - revision 11 only. If this cell is non-zero the output to the TTP is suppressed. Also if output is done to the com link it will be black print on a white background. Normally the cell is zero and output is always to the TTP.

COMIO - Cell 6 of CPMAIN - The low order 12 bit contains the output device specified in the last command processed by COMPROC. This cell may also be set by the calling program to force a specific output device regardless of the command input or the system POD.

COMIO	=	6	output to CLO, not to LP
	=	3	output to LP, not CLO
	≠	0	no more output other than TTP
	=	0	check M1CTRL for POD

M1CTRL - Cell 101 of memory - The system POD is checked if COMIO output unit is zero. If the POD is the line-printer, the record is output to the lineprinter.

Output

The output record is displayed to the device or devices indicated by the input parameters. Output to the com link is white print on a black background.

Description

The general purpose is to output a record to the TTP and also to the line printer or Integrator is either is specified in the input command, or to the line printer if LP is the POD. Output to the TTP is suppressed by setting TTPOUT not equal to zero.

Output to the com link occurs if COMIO equals 6.

Output to the line printer occurs if COMIO equals 3 or COMIO equals 1 and the system POD Is LP.

3.3.6 CPERR

Purpose

To output one of the standard system error messages.

Calling Sequence

```
    CPERR      EQU    2
               LDA    ERROR-NUMBER
               BRU*   CPERR,X2
                -     no return
```

Return

There is no return from CPERR. This is not a subroutine but an entry point into CPMAIN.

Input Data

The A register must contain a valid error number. No error check of the number takes place. The numbers and the corresponding error messages are:

1	COMMAND?
2	BAD NAME
3	DUPL./MISSING PARM.
4	ERROR -- COM LINK
5	FILE ID WRONG
6	INVALID NUMB
7	$TOPSY NOT CREATED WITH COM LINK
8	STATION UNASSIGNED
10	STATION MISSING

Output

The error message corresponding to the error-number list above is displayed on the VKT. If the original input message was from the host, the error message is also displayed at the host in bold type (black type on a white background).

After the error message is output the system PID is set to the TTK. Control then passes to the CPMAIN routine to read the next system command.

3.3.7 OVLYERR

Purpose

To output an error message defined by the calling program.

Calling Sequence

```
    OVLYERR    EQU    21
    DCBADDR    DATA   DCB
    DCB        DATA   n, test-address, text
                 '
                 '
                 '
               LDA    DCBADDR
               BRU*   OVLYERR,X2
```

Return

There is no return from OVLYERR. This is not a subroutine but an entry point into CPMAIN.

Input Data

The A register must contain the address of a DCB suitable for a call to TTRIO (or CLIO if CLIO might ever by used).

Output

The error message of the DCB Is displayed on the VKT. If the original input message was from the host, the error message is also displayed at the host in bold type (black type on a white background).

After the error message is output the system PID is set to the TTK. Control then passes to the CPMAIN routine to read the next system command.

APPENDIX A

STANDARD ASSEMBLY LANGUAGE PROGRAMS AVAILABLE

ALLINK Files on a Revision 11 System Tape

IBUS	PSCAN
LMLOAD	SPLOT
LMMOD	TTIME
LMTSF	XGRAPH
LPLF	

User Overlays on a Revision 11 System Tape

:IBUS	:PPLOG
:LMMOD	:PSCAN
:LMIO	:SPLOT

ALLINK Files on a Revision 10.4 System Tape

LMMOD	SPLOT
LMTSF	TTIME
LPLF	XGRAPH
PSCAN	

User Overlays on a Revision 10.4 System Tape

:DSPLM	:PSCAN
:LMIO	:SPLOT
:PPLOG	

APPENDIX B

EXECUTION TERMINAL ERROR NUMBERS

Certain set up and programming errors cannot be detected at compilation time; these errors are discoverable only while testing. The Terminal Error lamp goes 'ON' for errors described in the following table. The error number is logged. Both the Parameter FAIL and Parameter PASS lights are 'ON', but the EOT light is 'OFF'. The error number is displayed (in binary format) in the EIR register, bits 0-10; least significant bit to the left (bit 0).

Terminal Error Number	Description
1	A program has not been loaded for this station.
2	Station is disabled (power off)

3	Value programmed is negative or exceeds the hardware limit:
	
	SET DELAY, DC > 5.734 sec
	SET MAJOR loop count >4096
	SET MINOR loop count >4096

4	DMA statement execution process did not start. (Hardware error).
5	Magnitude programmed exceeds hardware limit: FORCE [E0/E1/EA0/---/EC1] >10 bits → -1024 or 1023 SET [S0/S1/SA0/SA1] >10 bits FORCE [VF1/VF2/VF3] >10 bits FORCE [IF1/IF2/IF3] >10 bits ENABLE [TRIPI1/TRIPI2/TRIPI3] >10 bits ENABLE [TRIPV1/TRIPV2/TRIPV3] >10 bits SET DCT >10 bits SET TG [DELAY/WIDTH] >10 bits

6	Value programmed is negative, zero or is outside of limit:
	
	SET PERIOD >40 milliseconds
	SET PERIOD >12 bits → -4096 or 4095
	SET PERIOD >200 nanosec for 5 MHZ
	SET PERIOD >100 nanosec for 10 MHZ
	SET TG(X) [DELAY/WIDTH] <10 nanosec

21	Value outside of limits set by ENABLE IHI/ILO

Terminal Error Number		Description
22		Value outside of limits set by VHI/VLO
23		Pin number is greater than 120; CPMU PIN
24		Value programmed for RVS exceeds hardware limit: [SET S0/S1/SA0/SA1] +6, -30V for 5 MHZ [SET S0/S1/SA0/SA1] +6, -16V for 10 MHZ FORCE [E0/E1/---EC1] → as above
26		Illegal OPCODE in FACTOR interpretive tester statement.
31		FACTOR magtape read error (File skip forward executed to move tape to the next tape file).
33		FACTOR magtape write error (File skip backward executed to move tape to the start of the last file. When start is pressed the program continues execution from this tape location).
35		FACTOR magtape EOT on write.
36		FACTOR magtape EOT on read.
37		FACTOR magtape memory protect on tape read.
40		FACTOR magtape data count less than 7 or greater than assigned buffer size. Also a memory overflow may have occurred.
42		FACTOR magtape irrecoverable error.
50	(a)	No DCL statement appears before this reference to the array element.
	(b)	Array has zero or negative number of elements.
51	(a)	The number of actual parameters does not agree with the number of formal parameters.
	(b)	TOPSY internal address error during store of a value, array element or formal value.
52	(a)	Array subscript exceeds 8388607, is negative or greater than the array size.
	(b)	Attempt to change array element 0 (i.e,, the array size)
53	(a)	The number of entries on the top of the working stack is less than required for current statement execution. (System error.)
	(b)	A block header memory address of zero has been encountered during update of Current Active Block pointer table. (System error.)
54	(a)	Array size declared exceeds 8388607 or available memory.
	(b)	Memory buffer available for tie test program is less than 1 disc sector (48 words).
55		The statement to be executed is not within the FACTOR object file. (System error).
56		Illegal FACTOR data code. (System error).
57		The number of array elements being initialized exceeds the declared array size.

Terminal Error Number	Description
58 (a)	FACTOR I/O started without previous I/O being completed.
(b)	Text is to be output without I/O being initialized.
(c)	Column formatting outside of I/O Process.
(d)	Literal variable outside of I/O process.
(e)	Column formatting is allowed on output only.
59	For statement loop control start value is less than end value.
60	Assembly Language Linkage program or PPM microprogram is not on the disc.
61	Assembly Language Linkage program or PPM microprogram load entry point overlaps the top of the working stack.
62	Arithmetic or logical operation overflow (ADD, SUBTRACT, MULTIPLY, DIVIDE, EXPONENTIATION, AND, OR, EOR, NOT, NEGATE).
67	DOF (disc output file) is not open.
68	Attempt to read beyond EOF (end of file) if DIF (disc input file) or DIF not open.
69	Attempt to write beyond EOF (end of file) of DOF (disc output file)
70	Attempt to execute a program without a SET PAGE statement on a SII and SVII or a SII and SVII Program on a S200/400 or a system without a tester.
71	Local memory size requested exceeds local memory available.
72	Programmed timing generator delay/width error (checked on SET PERIOD): Delay + Width ≥Period SII/SVII Delay or Width ≥ Period SII/SVII + 1 nsec. option Delay or Width range ≥Period range Width range ≥Delay range
74	Local memory address is negative or exceeds size requested by SET PAGE.
75	Attempt to execute a 10 MHZ FACTOR statement on a 5 MHZ system.
76	Attempt to execute a program with SET PAGE, SPO on a Sentry II without the Sequence Processor Module.

77	Attempt to execute a PPO program on a Sentry II without the Pattern Processor Module, i.e., one of the following was encountered:
	
	SET APERIOD
	SET ATG4 [DELAY/WIDTH]
	SET PPM
	REXEC

† Tape status issued in octal. On terminal errors 35 or 36 a tape rewind is executed.

Terminal Error Number	Description
79	Attempt to execute a 2v /2mv program on standard hardware.
81	The microprogram called by REXEC contains an assembly error.
82	The module number passed to the microprogram by the REXEC statement is negative.
83	The module number passed to the microprogram by the REXEC statement is greater than the number of modules in the micro-program.
84	Error in DMA loading the control RAM or in DMA Loading the PPM registers when a PPM microprogram is executed.
100-999	Terminal errors generated by ALLINK programs.