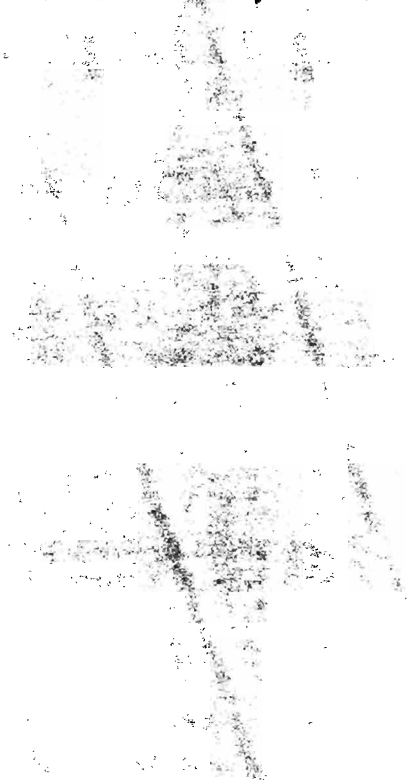
TABLE OF CONTENTS

1 GETTING STARTED

The purpose of this manual is to teach you enough about the Reality system so that you will feel comfortable writing assembly language programs. If you follow the instructions and do all of the exercises, you will quickly learn the basic components of the system.

Certain words in the text will guide you. "Read" means to chew and swallow the information; "scan" means to taste the information; "study" means to chew well and swallow; "review" means to recall certain flavors.

This manual assumes that you have access to a terminal attached to a Reality system with the full complement of software tools.



2 SOME MANUALS YOU WILL NEED

Several manuals provide most of the information you will need to program in assembly language. You should have a copy of each of the following for your own use:

Manual name	Mnemonic
Reality Programmer's Reference Manual	PRM
Reality CPU Reference Manual	CPU
Reality Assembler Manual	ASM
Reality System Software Subroutines	SSS
Reality Editor Programming Manual	EDM
Reality Assembly Language Programming Manual, Section "Interactive Debugger: DEBUG"	DEB

The mnemonics at the right are used in this manual for easy reference. For instance, PRM, section 4.3, refers to section 4.3 of the Reality Programmer's Reference Manual.

3 CREATING AN ACCOUNT

First, create an account for your use. Read PRM, sections 1-6, and then study sections 1, 2, 4, 5.1, 5.5, 10.1, and 10.25.

READ AND STUDY***READ AND STUDY***READ AND STUDY

Turn on your terminal. If the "LOGON PLEASE" message is not displayed, depress the RETURN key several times. When the message is displayed, type

SYSPROG

to enter the SYSPROG account. If the system asks for a password, ask your supervisor for the password, and enter it.

To create an account, type

CREATE-ACCOUNT

The CREATE-ACCOUNT PROC will ask a series of questions. Except for the following, answer all questions with a carriage return. For USER NAME reply with your last name. When the PRIVILEGES message is output, you must reply

SYS2 (065)

in order to perform assemblies.

After the account is established, the system must be informed that you intend to perform assemblies in your

account. Do this by typing:

```
SETUP-ASSY account-name
```

where account-name is the name that you gave to your account. The PROC will reply with a series of questions. For simplicity type a Y in reply to each question except this one:

```
ENTER 'Y' TO GIVE THE ACCOUNT SYSPROG'S ACCESS CODES AND
SYS2(25) PRIVILEGES, ELSE 'N' (Y/N)
```

Reply with an N. Your account is now ready for use in assembling programs.

To enter your account, type:

```
LOGTO account-name
```

After TCL displays a colon, you are operating in your own account.

4 DEFINING A VERB

To understand how verb definitions are created, learn to use the Editor, especially the following commands: G, I, L, DE, R, F, and FI. Read EDM, sections 2 and 3.1 through 3.8. Also, review PRM, sections 6.5 and 4.3.

READ AND REVIEW***READ AND REVIEW***READ AND REVIEW

TIME is a TCL-I verb. When invoked, it displays the date and time-of-day. Type

```
TIME
```

on the terminal to see the format of the reply. Following the directions given below, create a synonym for the verb TIME.

Copy the verb TIME to the terminal by typing

```
CT MD TIME
```

The system will display

```
001 PZ
002 3033
```

The P indicates that this is a verb. Since the second letter of the first line, or attribute, is not a Q, TIME is not a PROC. The second line contains the mode-id of the program. Since this value is not 2 or 35, TIME is a TCL-I verb, rather than a TCL-II or ENGLISH verb.

As an exercise in creating verb definitions, produce a synonym for the verb TIME. Enter the Editor specifying file MD and item T. Type

```
ED MD T
```

when the Editor replies, use the I command to insert two lines:

```
I
000+ PZ
000+ 3033
000+
```

Remember to terminate the insert with a carriage return on a null line.

To display the information you have inserted, type

```
L22
```

where 22 is normally used because 22 lines can be displayed on the terminal.

Write the item to the file by typing

```
FI
```

Now you can invoke the function by typing

```
T
```

The reply should have the same format as the reply for TIME.

```
G1
R
your name or the name of the mode
FI
```

This will notify other programmers that you are using frame 337.

5.2 CREATING A FILE FOR YOUR MODE

Read ASM, section 9, and review the procedure for creating a file in PRM, section 6.2.

READ AND REVIEW***READ AND REVIEW***READ AND REVIEW

Start a source mode by creating a file and editing the first six lines of a mode with documentation information. For ease of reference the file name FF will be used in this manual. To create file FF type

```
CREATE-FILE (FF 1,1 1,1)
```

the ones are the modulo and separation for the dictionary and data portions of the file. You can use other values when you learn more about the system.

5.3 EDITING YOUR MODE

Create the first six lines of your mode by invoking the Editor:

```
ED FF MOD1
```

The first line should be a FRAME statement with the FID of your mode. When you have edited these six lines, they should look similar to this:

```
 FRAME 337
* TEST MODE
* 11JAN80
* 00
* 00
* SMITH
```

NOTE: DO NOT assemble the mode with only these six lines.

6 INTRODUCING THE SYSTEMS SUBROUTINES

Much of the system programming that is done on Reality uses the system software subroutines. Read SSS, sections 1 through 4.

READ***READ***READ

In a sense these subroutines are the system with which you must interface. We will look at several of the subroutine descriptions, but you should peruse the SSS manual each time you design or change a mode.

When one of the exercises in this manual directs you to read the description of a subroutine, do not try to grasp every detail in the description, rather read for a general understanding. When you are told to concentrate on certain elements in a description, ignore other elements because they contain the correct data for the exercises.

7 TCL-I VERBS

Review PRM, sections 4.1, 4.2 and 4.3.

REVIEW***REVIEW***REVIEW

Generally speaking, TCL-I verbs perform utility functions that do not reference an explicitly specified file. The verb TIME translates the system clock into a time and date that it prints on the terminal. The arithmetic commands (PRM, 9.1) are TCL-I verbs that require parameters. BLOCK-TERM and BLOCK-PRINT (PRM, 9.4) reference a file to translate the input letters into block letters, but this file is not specified by the user.

7.1 CREATING A VERB AND A MODE

Using the frame number that you recorded in file FRAM, create a verb definition in your master dictionary. Call the item AQ. Put a P in the first line and the mode-id of your program in the second. The mode-id must be in hexadecimal. Since the mode entry will be the zeroeth entry, the mode-id will be simply the FID in hexadecimal. No leading zero is needed.

Invoke the Editor by typing

```
ED MD AQ
```

and then insert the two lines. If your FID were 337, the verb definition would look like this:

```
AQ
001: P
002: 151
```

Do not invoke verb AQ on the terminal until after you have developed the mode that will be called by the verb.

In SSS read section 5 and scan the descriptions of the following subroutines: TCL-I, CHANCE2, and WRAPUP.

READ AND SCAN***READ AND SCAN***READ AND SCAN

TCL-I is not a subroutine: its main importance lies in the initialization that it performs. We will study its output interface in more detail later.

CHANCE2 is a simple subroutine that must be called by means of the Branch and Stack Location (BSL) instruction. Because the heading line

```
CHANCE2 (9,167)
```

is not followed by an asterisk, CHANCE2 is not defined in PSYM. Hence, its mode-id must be defined in your mode with the DEFM directive. Use entry point 9 and FID 167 (decimal) with the DEFM.

When a program has completed its processing, it must do an External Branch (ENT) to one of WRAPUP's entry points. We will use entry point MD999.

Write a mode that will be called by the AQ verb. In the mode call CHANCE2 and test its output for an A. If CHANCE2's output is an A, call CHANCE2 again. Otherwise, go

to WRAPUP. The following statements exemplify the logic:

```
Repeat
     Call CHANCE2
Until CHANCE2's output does not equal 'A'.
Go to MD999.
```

Notice that the output interface section of CHANCE2's description specifies that the character entered from the terminal is returned in H1 and that R15 points to H1.

Conventionally, the documentation information at the beginning of the mode is followed by entry-point Branch Local instructions into the main logic of the mode. Since a mode may have more than one entry point, additional branch instructions can be added as they are needed. Constants and symbol definitions follow the branch entry points.

Read ASM, section 8, especially about the FRAME, DEFM and EQU directives.

READ***READ***READ

In CPU study the following instructions:

```
B     L          - BRANCH LOCAL

BSL   M          - BRANCH AND STACK EXTERNAL

ENT   M          - BRANCH EXTERNAL

BCE   N,R,L      - BRANCH IMMEDIATE COMPARED TO CHARACTER
```

STUDY***STUDY***STUDY

Write the code for the program, and using the Editor, add it to the mode you have already started. When you enter the data, start all labels in the first position. If there is no label in the label field, space one character only. In other words, do not try to line up the fields of all the instructions. After a label, space one. After the operation mnemonic, space one. Separate the operand field and any commentary by one space. Use the Editor's AS function (EDM, section 3.12) to list the instructions aligned by fields.

After you have edited the mode, read ASM, sections 12.1 through 12.3.

READ***READ***READ

Assemble the mode:

```
AS FF MOD1
```

List the mode on the printer:

```
MLIST FF MOD1 (P)
```

and if there are no errors, load it into its absolute frame:

```
MLOAD FF MOD1
```

The mode's listing should look similar to the one in Figure 7-1.

Invoke the verb by typing

```
AQ
```

It should reply with the message

```
TYPE A (AGAIN) OR Q (QUIT)
```

If you type "A", it should retype the message. If you enter any other character, it should return to TCL, which will prompt with a colon.

```
PAGE     1              MOD1              FRAME 337          15:09:24  16 SEP 1981

001 000 7FF00151                  FRAME 337
    001
002                    *
003                    *26JUN80
004                    *
005                    *
006                    *MOORE
007                    *
008 001 1E02                      B       !AQ
009                    *
010     M 9 A7         CHANCE2    DEFM    9,167
011                    *
012                    !AQ        EQU     *
013 003 1190A7                    BSL     CHANCE2
014 006 4F410802                  BCE     C'A',R15,!AQ
015 00A 10100A                    ENT     MD999
    FFE 0241
EOF
```

FIG. 9-1

7.2 A TCL-I VERB WITH PARAMETERS

In EDM read about ME in section 3.4.

READ***READ***READ

Create another verb, called DADD, in your master dictionary by entering the Editor with

```
ED MD DADD
```

Use the ME command to merge the contents of verb AQ into this item:

```
ME999"AQ"1
F
```

The 999 is just a large number. Inputting the F command allows you to modify the data that you have merged. Replace line two with a mode-id into entry point one of your frame. If your frame were 337, the item would look like this when you are finished:

```
DADD
001: P
002: 1151
```

The logic of verb DADD is simple. We want to type in the verb DADD followed by two decimal numbers. The mode will add the numbers together and print the sum on the terminal. The logic, in terms of system subroutines, follows:

Call CVDIS to convert the first number to binary.

Save the first binary number in location FP2.

Call CVDIS to convert the second number to binary.

Add the first number to the second.

Call MBDSUB to convert the sum to ASCII in position six of the output buffer.

Call WRTLIN to print the result on the terminal.

Go to MD999.

Functional element FP2 is not used by any of the called subroutines or any of the secondary subroutines.

In SSS read the descriptions of subroutines TCL-I and CVDIS.

READ***READ***READ

In the output interface for MD1B of TCL-I note that register IS points one character before the beginning of the edited input line. This means that if the line

```
DADD 3 4
```

were entered, IS would point one character position before the 3. In the description of CVDIS read that register IS must be set in exactly this condition. When CVDIS returns to the caller, IS points at the character which stopped the conversion. In the example above, it would point at the space before the 4, which means that it is ready for a call to convert the second number. CVDIS outputs the binary

number in FP0, the extended accumulator.

Use the Store Accumulator (STORE) instruction to save the first number in FP2, and use Add Accumulator (ADD) to add the first number to the second.

In SSS study subroutines MDBSUB and WRTLIN.

STUDY***STUDY***STUDY

MBDSUB, which converts a binary number in FP0 to a string of ASCII characters, expects R15 to point one prior to an area where the string is to be stored. WRTLIN, on the other hand, expects storage register OBBEG to point one prior to the output data and address register OB to point to the last character to be printed.

To gain some appreciation of the initialization done by the system read the description of subroutine MD0_in SSS.

READ***READ***READ

This subroutine initializes a process before LOGON starts the process. Notice that PINIT is called: Read the description of that subroutine.

READ***READ***READ

PINIT calls WSINIT, which initializes many of the triads. In the output interface section of WSINIT's description see that work space OB is filled with blanks and that OBBEG and register OB both point to the beginning of this space.

SEE***SEE***SEE

This information suggests a method of interfacing with subroutines MBDSUB and WRTLIN.

Increment the contents of register 0B by six since we want to output the result starting in position six. Work space 03 has been filled with blanks so do not initialize the area. Use the instruction

```
INC 0B,6
```

which will generate a literal for the six. After this instruction executes, register 0B will be pointing at the proper output location. Because MBDSUB requires its buffer pointer in R15, move the contents of 0B to R15. See the discussion of the MOV R,R instruction in CPU.

SEE***SEE***SEE

When MBDSUB returns, R15 will be pointing at the last converted character. Since WRTLIN expects 0B to point to the last character in the buffer, move the contents of R15 to 0B. After calling WRTLIN, go to MD999.

Remember to insert an entry-point branch instruction to the new logic. It should be placed immediately after the entry point zero branch.

Use the Editor to enter your new code. Then assemble, list and load it. See Figure 7-2 for comparison with your code. The listing in this figure was produced using the M option so that macro expansions would be evident.

Test your mode by typing the verb DADD followed by two decimal numbers.

```
001 000 7FF00151                FRAME 337
    001
00                  *
00                  *26JUN80
00                  *
00                  *
006                 *MOORE
007                 *
008                         B       !AQ
009                         B       !DADD
010                 *
011     M 9 A7      CHANCE2 DEFM    9,167
012                 *
013                 !AQ     EQU     *
014 003 1190A7              BSL     CHANCE2
015 006 4F410802            BCE     C'A',R15,!AQ
016 00A 10100A              ENT     MD999
    FFE 0241
017                 *
018                 !DADD   EQU     *
019 003 115008              BSL     CVDIS
020 006 A22DD9              STORE   FP2
021 009 115008              BSL     CVDIS
022 00C A22DD3              ADD     FP2
023 00F E1105B              INC     OB,6
024 012 15BF                MOV     OB,R15
025 014 110008              BSL     MBDSUB
026 017 16FB                MOV     R15,OB
027 019 112006              BSL     WRTLIN
02  01C 10100A              ENT     MD999
    01F 00
    020 0006
    FFE 08CE
EOF
```

FIG. 9-2

7.3 READING INPUT

Change the logic of the mode to solicit more addends after adding the first two entered. The following statements specify the new logic for DADD:

Call CVDIS to convert the first number to binary.

Save the first binary number in location FP2.

Repeat

Call CVDIS to convert the next number to binary.

Add the contents of FP2 to the binary number.

Save the sum in FP2.

Call MBDSUB to convert the sum to ASCII in position six of the output buffer.

Call WRTLIN to print the results on the terminal.

Call READLIN to read another number.

Until READLIN's input is null.

Go to MD999.

In SSS read the entire description of subroutine READLIN and the functional description of subroutine GETBUF.

READ***READ***READ

READLIN calls subroutine GETBUF to read a line of input from the terminal. GETBUF prints a prompt character and reads the input. When READLIN returns to the caller, IBBEG and IB both point one byte before where the input begins. Since subroutine CVDIS expects register IS to point one byte before its input data, use the MOV R,R instruction to move the contents of IB to IS.

The input line must be checked for no input. Since IB points one byte before the data, increment IB to point at the next byte and then test if the byte contains a segment mark. A segment mark in this position would indicate that no data was entered. In CPU see instructions INC R and BCE N,R,L.

SEE***SEE***SEE

In PSYM there is a symbol, SM, that is a constant having the value of a segment mark (X'FF'). This symbol should be used in the instruction.

Assemble, load, and list your mode. Compare it with Figure 7-3. The asterisks on the right side of the listing indicate the new lines. Invoke the mode by typing

DADD

followed by two decimal numbers. After the mode prints the sum, it will print a colon, prompting you to enter another number. Type another number and see the result. Continue entering numbers as the colon prompts you. Note that a negative number can be entered: Just precede the number with a minus sign.

To terminate the mode enter a carriage return without a number. The next colon that prints will be TCL's prompt.

```
001 000 7FF00151                FRAME 337
    001
002                             *
003                             *14JUL80
004                             *
005                             *
006                             *MOORE
007                             *
008 001 1E04                    B       !AQ
009 003 1E0E                    B       !DADD
010                             *
011     M 9 A7                  CHANCE2 DEFM    9,167
012                             *
013                             !AQ     EQU     *
014 005 1190A7                  BSL     CHANCE2
015 008 4F410804                BCE     C'A',R15,!AQ
016 00C 10100A                  ENT     MD999
017                             *
018                             !DADD   EQU     *
019 00F 115008                  BSL     CVDIS
020 012 A22DD9                  STORE   FP2
021                             AGAIN   EQU     *               *****
022 015 115008                  BSL     CVDIS
023 018 A22DD3                  ADD     FP2
024 01B A22DD9                  STORE   FP2                     *****
025 01E E11D5B                  INC     OB,6
026 021 16BF                    MOV     OB,R15
027 023 110008                  BSL     MBDSUB
028 026 16FB                    MOV     R15,OB
029 028 112006                  BSL     WRTLIN
030 02B 110006                  BSL     READLIN                 *****
031 02E 16A4                    MOV     IB,IS                   *****
032 030 3A                      INC     IB                      *****
033 031 4AFF0836                BCE     SM,IB,OUT               *****
034 035 1E14                    B       AGAIN                   *****
035                             OUT     EQU     *               *****
036 037 10100A                  ENT     MD999
    03A 0006
    FFE 113A
EOF
```

FIG. 7-3

7.4 CHANGING THE PROMPT CHARACTER

Since both DADD and TCL use the colon as a prompt character, it is difficult to know whether the colon is a prompt for another addend or for a TCL verb. In the input interface sections of READLIN and GETBUF see that location PRMPC contains the prompt character.

SEE***SEE***SEE

Let us add instructions that will move a question mark into PRMPC before READLIN is called.

In ASM read about the FRAME directive. In CPU read about the MCC R,W instruction.

READ***READ***READ

The FRAME statement sets the location counter to one, leaving byte zero of the frame unused. This byte can be used to hold a frequently used character or half word constant, usually an attribute mark, a segment mark, or a blank. In ASM read about the ORG and the CHR directives.

READ***READ***READ

Just before the entry-point branch instructions set the location counter to zero and define a character constant containing a question mark. At some convenient place in DADD's logic move the constant to PRMPC. Remember that address register one (R1) points to byte zero of the mode so use the instruction

```
MCC R1,PRMPC
```

Assemble, list and load your mode. Compare it with the listing in Figure 7-4. DADD should now print a question mark to prompt for input.

```
001 000 7FF00151                FRAME 337
    001
002                 *
003                 *16JUL80
004                 *
005                 *
006                 *MOORE
007                 *
008 000                         ORG   0                  *****
009 000 3F                      CHR   C'?'               *****
010 001 1E04                    B     !AQ
011 003 1E0E                    B     !DADD
012                 *
013     M 9 A7      CHANCE2     DEFM  9,167
014                 *
015                 !AQ         EQU   *
016 005 1190A7                  BSL   CHANCE2
017 008 4F410804                BCE   C'A',R15,!AQ
018 00C 10100A                  ENT   MD999
019                 *
020                 !DADD       EQU   *
021 00F D00201                  MCC   R1,PRMPC           *****
022 012 115008                  BSL   CVDIS
023 015 A22DD9                  STORE FP2
024                 AGAIN       EQU   *
025 018 115008                  BSL   CVDIS
026 01B A22DD3                  ADD   FP2
027 01E A22DD9                  STORE FP2
028 021 E11F5B                  INC   OB,6
029 024 16BF                    MOV   OB,R15
030 026 110008                  BSL   MBDSUB
031 029 15FB                    MOV   R15,OB
032 02B 112006                  BSL   WRTLIN
033 02E 110006                  BSL   READLIN
034 031 16A4                    MOV   IB,IS
035 033 3A                      INC   IB
036 034 4AFF0839                BCE   SM,IB,OUT
037 038 1E17                    B     AGAIN
038                 OUT         EQU   *
039 03A 10100A                  ENT   MD999
```

FIG. 9-4

```
03D 00
03E 0006
TFE 12D4
```

EO.

7.5 REMOVING EXTRA BLANKS

The verb DADD will now handle valid input. If you enter leading blanks, however, the program will treat them as zeros. Extra blanks on the verb line are eliminated by TCL-I, but they are not eliminated on subsequent input lines.

In CPU read sections 9.6 through 9.6.4 and read about the SCD R,N and DEC R instructions. In the output interface section of TCL-I's MD1 routine see that SC1 and SC2 are both set to blank when your mode is started.

READ AND SEE***READ AND SEE***READ AND SEE

Change the mode so that it bypasses leading blanks in READLIN's input. The logic for this part of the mode would be:

Call READLIN.

Scan the input buffer using register IB until a non-blank is found.

If the non-blank character is a segment mark, go to OUT.

Decrement IB.

Move IB to IS.

Go to AGAIN.

We decrement IB before moving it to IS because CVDIS expects IS to point one byte before the data to be converted.

After you change your mode, assemble, list and load it. See Figure 7-5 for comparison.

DADD should now ignore leading blanks.

```
001 000 7FF00151                FRAME 337
    001
002                 *
003                 *15JUL80
004                 *
005                 *
006                 *MOORE
007                 *
008 000                         ORG   0
009 000 3F                      CHR   C'?'
010 001 1E04                    B     !AQ
011 003 1E0E                    B     !DADD
012                 *
013     M 9 A7      CHANCE2     DEFM  9,167
014                 *
015                 !AQ         EQU   *
016 005 1190A7                  BSL   CHANCE2
017 008 4F410804                BCE   C'A',R15,!AQ
018 00C 10100A                  ENT   MD999
019                 *
020                 !DADD       EQU   *
021 00F D00201                  MCC   R1,PRMPC
022 012 115008                  BSL   CVDIS
023 015 A22DD9                  STORE FP2
024                 AGAIN       EQU   *
025 018 115008                  BSL   CVDIS
026 01B A22DD3                  ADD   FP2
027 01E A22DD9                  STORE FP2
028 021 E1205B                  INC   0B,6
029 024 16BF                    MOV   0B,R15
030 026 110008                  BSL   MBDSUB
031 029 15FB                    MOV   R15,0B
032 02B 112006                  BSL   WRTLIN
033 02E 110006                  BSL   READLIN
034 031 6A0801                  SCD   IB,X'01'          *****
035 034 4AFF083C                BCE   SM,IB,OUT
036 038 2A                      DEC   IB                *****
037 039 16A4                    MOV   IB,IS             *****
038 03B 1E17                    B     AGAIN
039                 OUT         EQU   *
```

FIG. 2-5

```
040 03D 10100A          ENT · MD999
    040 0006
    FFE 1344
```

(FIG 9-5)

8 TCL-II VERBS

Review PRM, sections 4.1 through 4.4, 2.4, and 2.5. In SSS read about TCL-II.

REVIEW AND READ***REVIEW AND READ***REVIEW AND READ

TCL-II verbs create, modify, and move items in files. Some examples of TCL-II verbs are : EDIT, AS, MLOAD, and MLIST.

8.1 RETRIEVING AN ITEM

Create a TCL-II verb in your master dictionary. Use the same frame that you used for your TCL-I verbs. Although it is possible to add the logic for the TCL-II verb to the mode as it now exists, replace the old mode completely and point the verb at entry point zero. Call the new verb DISP. If your frame number were 337, the verb definition would look like this:

```
DISP
001:  P
002:  2
003:  151
```

The 2 in line two, which marks the definition as a TCL-II verb, is the mode-id of TCL-II's MD200 entry point. The mode-id of your mode goes into line three.

The logic of verb DISP is as follows:

Input an item.

Call WRTLIN to print the item's first line on the terminal.

Go to MD999.

The first step, inputting an item, is done by TCL-II. Notice in TCL-II's output interface section that IR points to the attribute mark following the item-id, that is, it points one byte prior to the first line of the item. SR4 points to the last attribute mark of the item. The contents of these two elements delineate the item so that we can reference any line we wish.

To print the first line of the item move the first line to the buffer pointed at by OBBEG because WRTLIN expects OBBEG to be pointing one byte prior to the output data. OB should be pointing to the last character in the output. In CPU study the following instructions:

```
MOV  S,R        - LOAD ADDRESS REGISTER

MIID R,R,N      - INCREMENT AND MOVE STRING UNDER
                  DELIMITER CONTROL

DEC  R          - DECREMENT ADDRESS REGISTER BY ONE
```

STUDY***STUDY***STUDY

Move the item's first line to work space OB by first moving OBBEG to register OB. Then move the string pointed at by IR to the buffer pointed at by OB until an attribute mark is encountered. Since OB will be pointing at the attribute mark, decrement it by one because WRTLIN expects it to be pointing at the last byte of print data.

Edit your mode for the verb DISP, remembering to include a Branch Local instruction as an entry point. After assembling, loading, and listing your mode, compare it with Figure 8-1.

Invoke the verb by typing DISP followed by a filename and an item-id, for example:

```
DISP FF MOD2
```

The first line of the item should be displayed on the terminal.

```
PAGE    1            MOD2              FRAME 337          16:16:34   16 SEP 1981

001 000 7FF00151                  FRAME 337
    001
000                   *
000                   *17JUL80
000                   *
000                   *
006                   *MOORE
007                   *
008 001 1E02                      B       !DISP
009                   *
010                   !DISP       EQU     *
011 003 E0EAEB                    MOV     OBBEG,OB
012 006 66B0A0                    MIID    IR,OB,X'A0'
013 009 2B                        DEC     OB
014 00A 112006                    BSL     WRTLIN
015 00D 10100A                    ENT     MD999
    FFE 0542
EOF
```

FIG. 8-1

8.2 RETRIEVING MANY ITEMS

Review section 4.4 in PRM. In SSS review the discussion of system element RMODE in section 3.8 and in the TCL-II description.

REVIEW***REVIEW***REVIEW***

Retrieving many items is the same as retrieving one: TCL-II handles the fetching of each item specified by the user in the terminal input statement. Hence, the mode as now written will handle more than one item.

Notice in the subroutine usage section of TCL-II's description that MD201, which is called by WRAPUP through RMODE, calls WSINIT. Each time your mode exits to WRAPUP, before MD201 returns with the next item, all of the initialization performed by WSINIT will have been done.

Invoke the DISP verb with multiple items:

```
DISP FF MOD1 MOD2

DISP FF *
```

8.3 USER OPTIONS

User options are the options that can be specified with the verb entered at the terminal. They are not related to the options specified in line five of a TCL-II verb definition, which should be thought of as designer options. An

explanation of designer options is beyond the scope of this introductory manual.

User options can be defined in the mode to mean anything you want them to mean. However, the following options are historically assigned these meanings:

Option	Meaning
N	Suppresses the pause at the end of each page when the listing is output to the terminal.
P	Routes output to the printer (spooler).

Add these two options to your mode, but before doing so, enter the following command at the terminal:

```
DISP MD *
```

The first line of every item in the master dictionary should print on the terminal without pause.

In SSS note that TCL-I zeros ABIT-ZBIT in the MD1 routine. Also note that MD1B calls subroutine GETOPT. Read GETOPT's description. Note the following system elements in WRTLIN's input interface section: LPBIT, PAGINATE, and PAGFRMT. Read subroutine SETLPTR's description. In CPU read about the following instructions:

```
SB   B       - SET BIT

BBS  B,L     - BRANCH ON BIT SET

BBZ  B,L     - BRANCH ON BIT ZERO
```

In your mode set bit PAGINATE to allow bit PAGFRMT to be operative. Since subroutine GETOPT will set NBIT if there is an N in the option list, test if bit NBIT is set. If it is not, set PAGFRMT. Likewise, if PBIT is set, call subroutine SETLPTR, which will set LPBIT.

Add the new code to your mode. Assemble, load, and list it, and compare it with Figure 8-2.

Invoke DISP with either option N or P, and note its action:

```
DISP MD *
DISP MD * (N)
DISP MD MOD1 MOD2 (P)
```

```
001 000 7=F00151                FRAME 337
    001
0                   *
0                   *18JUL80
00                  *
00                  *
006                 *MOORE
007                 *
008 001 1E02                    B       !DISP
009                 *
010                 !DISP       EQU     *
011 003 80F7                    SB      PAGINATE            *****
012 005 928D000A                BBS     NBIT,DI10           *****
013 009 80CC                    SB      PAGFRMT             *****
014                 DI10        EQU     *                   *****
015 00B 928F0811                BBZ     PBIT,DI20           *****
016 00F 118033                  BSL     SETLPTR             *****
017                 DI20        EQU     *                   *****
018 012 E0EAEB                  MOV     OBBEG,OB
019 015 66B0A0                  MIID    IR,OB,X'A0'
020 018 23                      DEC     OB
021 019 112006                  BSL     WRTLIN
022 01C 10100A                  ENT     MD999
    FFE 0BA2
EOF
```

FIG. 8-2

8.4 FETCHING ANOTHER ITEM

For this next exercise create a chain of items in your file. In the first line of the first item put the item-id of the second item; in the first line of the second item put the item-id of the third item; and so forth. Make the first line of the last item null. EDM, section 3.3, tells how to create a null line with the insert and replace commands. An example of a chain would be:

```
      T1
001:  T2

      T2
001:  T3

      T3
001:
```

CREATE A CHAIN***CREATE A CHAIN***CREATE A CHAIN

In SSS read about subroutine RETIX. Also in TCL-II's output interface description note that the elements BASE, MODULO and SEPAR point to the file specified by the user on the terminal. In PRM, section 2.4, review the meanings of base, modulo, and separation.

READ AND REVIEW***READ AND REVIEW***READ AND REVIEW

Change your mode to check the first line of the item for a null line. If it is null, go to WRAPUP; otherwise, print

the line and then read the item that it references, repeating the check for null. Develop a local subroutine (GETSYM) that will extract from the first line a symbol to be used as the item-id of the next item. The logic of the entire mode will be:

Set PAGINATE.

If N is an option, set PAGFRMT.

If P is an option, call SETLPTR.

While the item's first line is not null and there are no errors:

Call WRTLIN to print first line of item.

CALL GETSYM to extract symbol from first line.

Call RETIX to read next item.

Go to MD999.

Subroutine GETSYM

Eliminate leading blanks.

Move symbol to area BMS.

End symbol with an attribute mark.

Return.

The new logic does not change the previous handling of user

options so the first part of your mode can remain the same. Before calling WRTLIN (just after label DI20 in Figure 8-2), move IR to IB to save it for later use. To test if the first line of the item is null, increment IR. Use the BCE N,R instruction to test if IR is pointing at an attribute mark (AM); if IR is pointing to an attribute mark, go to OUT; if not, decrement IR.

The code for calling WRTLIN can remain intact at this point. Follow it with a call to GETSYM, which will return with a symbol in work space BMS with BMSBEG pointing one byte before the symbol. An attribute mark will follow the symbol. Call RETIX to read the item specified by BMSBEG. If RETIX returns with RMBIT equal to zero, go to NOITEM; otherwise, branch back to print the new item's first line (DI20). For now, equate symbol NOITEM to the normal return to WRAPUP.

In subroutine GETSYM move BMSBEG TO BMS. Using SCD R,N with IB eliminate leading blanks by scanning for a non-blank character (Remember that SC2 contains a blank). Decrement IB so that it points one byte before the non-blank character, and move a string from the buffer pointed to by IB to the buffer pointed to by BMS using the MIID R,R,N instruction until a blank or an attribute mark is encountered. Then use MCC N,R to move an attribute mark after the symbol. Return to the main routine with the RTN instruction.

After assembling, loading, and listing your mode, compare it with Figure 8-3. Test the mode by invoking DISP and the first item of the chain that you created:

```
DISP FF T1
```

```
001 000 7FF00151                FRAME 337
    001
002                     *
003                     *02APR79
004                     *
005                     *
006                     *MOORE
007                     *
008 001 1E02                    B       !DISP
009                     *
010                     !DISP   EQU     *
011 003 80F7                    SB      PAGINATE
012 005 928D000A                BBS     NBIT,DI10
013 009 80CC                    SB      PAGFRMT
014                     DI10    EQU     *
015 00B 928F0811                BBZ     PBIT,DI20
016 00F 118033                  BSL     SETLPTR
017                     DI20    EQU     *
018 012 166A                    MOV     IR,IB                   *****
019 014 36                      INC     IR                      *****
020 015 45FE082E                BCE     AM,IR,OUT               *****
021 019 26                      DEC     IR                      *****
022 01A E0EAEB                  MOV     OBBEG,OB
023 01D 66B0A0                  MIID    IR,OB,X'A0'
024 020 2B                      DEC     OB
025 021 112006                  BSL     WRTLIN
026 024 1831                    BSL     GETSYM                  *****
027 026 111007                  BSL     RETIX                   *****
028 029 909E082E                BBZ     RMBIT,NOITEM            *****
029 02D 1E11                    B       DI20                    *****
030                     NOITEM  EQU     *                       *****
031                     OUT     EQU     *                       *****
032 02F 10100A                  ENT     MD999
033                     *
034                     GETSYM  EQU     *                       *****
035 032 E07AE8                  MOV     BMSBEG,BMS              *****
036 035 6A0801                  SCD     IB,X'01'                *****
037 038 2A                      DEC     IB                      *****
038 039 6A80A1                  MIID    IB,BMS,X'A1'            *****
039 03C 48FE20                  MCC     AM,BMS                  *****
```

FIG. 8.3

```
040 03F 14                    RTN   *                 *****
    FFE 185A
E
```

8.5 RETRIEVING FROM ANOTHER FILE

In the last exercise you created a chain of items in one file. Extend this chain into another file by inserting the file name and item-id of the next item into the first line of the last item of the chain. For example, if the chain were to go from file FF into the MD (master dictionary) file and back to file FF, the continuation would look like this:

```
In file FF
     T3
001: MD T4

In file MD

     T4
001: T5

     T5
001: T6

     T6
001: FF T7

In file FF

     T7
001: T8

     T8
001: T9

     T9
001:
```

file item in MD D/ID in dict file

1 D D

2 base fid base fid

3 modulo modulo

4 sep sep

EXTEND THE CHAIN***EXTEND THE CHAIN***EXTEND THE CHAIN

In PRM read sections 1.7, 2.4, 3.1, and 3.2. In section 3.2 note especially the discussion of item-id and attributes 1 through 4 and the discussion about Figure A. In SSS, section 3.3, read about elements MBASE, MMOD, and MSEP. Also, read the descriptions of subroutines RETIX, GBMS, and GDLID.

READ***READ***READ

Change the logic of your mode to access an item from a file different from the one currently being accessed. The following statements specify what has to be done, exclusive of initializing the print options and returning to WRAPUP:

> While the first line is not null and there are no errors:
>
> > Call WRTLIN to print the item's first line.
> >
> > Call GETSYM to get a symbol.
> >
> > If GETSYM did not reach the end of the line
> >
> > > Call RETIX to read the file's definition item in the master dictionary.
> > >
> > > Call GBMS to get the file's dictionary-level base, modulo, and separation from its definition.

Call GDLID to get the file's data-level base, modulo, and separation from the DL/ID item in its dictionary.

Call GETSYM to get a symbol.

Call RETIX to read an item.

The logic of your mode can remain the same up through the BSL to GETSYM. After that, test if the end of the line was reached. Use a BCE N,R,L instruction to see if IB points to an attribute mark. Branch to where RETIX reads another item.

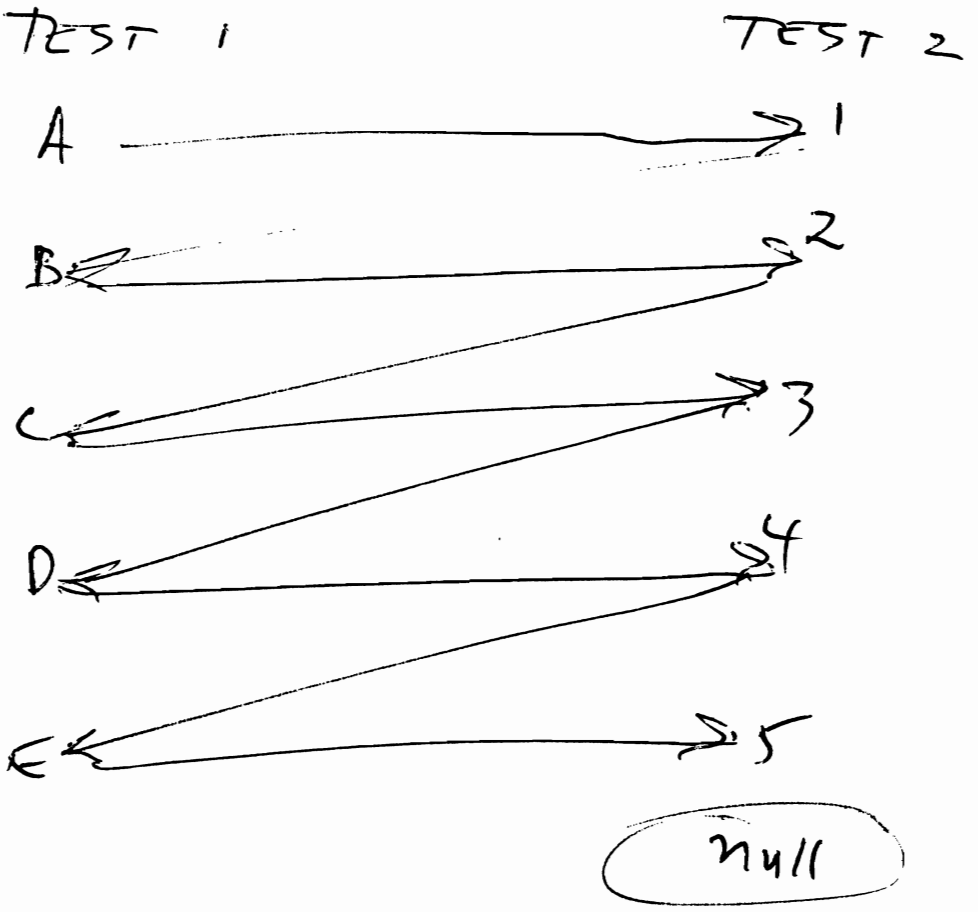
If GETSYM did not reach the end of the line then assume that a file name is present. RETIX must be used to read the file's definition from the master dictionary. Since GETSYM puts the symbol into a buffer pointed at by BMSBEG, that parameter is taken care of, but the master dictionary's base, modulo, and separation must be moved to elements BASE, MODULO, and SEPAR. This information is contained in elements MBASE, MMOD, and MSEP. Moving MBASE to BASE requires one instrucion. Moving MMOD and MSEP to MODULO and SEPAR can also be done with only one instrucion. PSYM defines a double word, MMODMSEP, which includes the contiguous words MMOD and MSEP. Likewise, the definition of MODULOSEPAR encompasses the two words MODULO and SEPAR. So, move MMODMSEP to MODULOSEPAR. Then call RETIX. If RMBIT is zero, go to NOFILE. Call GBMS, and if RMBIT is zero, go to NOFILE. Call GDLID; if RMBIT equals zero, go to NODATA. Call GETSYM to get another symbol from the original item's first line. A call to RETIX to read another item already exists in the mode.

Equate the symbols NOFILE and NODATA to the return to

WRAPUP.

Add the new logic to the mode. Assemble, load, and list it. See Figure 8-4 for comparison. Invoke the mode by referencing the item chain you have built:

```
DISP FF T1
```



```
001 000 7FF00151                FRAME 337
    001
                        *
                        *23JUL80
                        *
                        *
006                     *MOORE
007                     *
008 001 1E02                    B       !DISP
009                     *
010                     !DISP   EQU     *
011 003 80F7                    SB      PAGINATE
012 005 928D000A                BBS     NBIT,DI10
013 009 80CC                    SB      PAGFRMT
014                     DI10    EQU     *
015 00B 928F0811                BBZ     PBIT,DI20
016 00F 118033                  BSL     SETLPTR
017                     DI20    EQU     *
018 012 166A                    MOV     IR,IB
019 014 36                      INC     IR
020 015 46FE0851                BCE     AM,IR,OUT
021 019 26                      DEC     IR
022 01A E0EAEB                  MOV     OBBEG,OB
023 01D 66B0A0                  MIID    IR,OB,X'A0'
024 020 2B                      DEC     OB
025 021 112006                  BSL     WRTLIN
026 024 1854                    BSL     GETSYM
027 026 4AFE0848                BCE     AM,IB,DI80          *****
    02A F0288030                MOV     MBASE,BASE          *****
    02E F02A8032                MOV     MMODMSEP,MODLLOSEPAR *****
    032 111007                  BSL     RETIX               *****
031 035 909E0851                BBZ     RMBIT,NOFILE        *****
032 039 113003                  BSL     GBMS                *****
033 03C 909E0851                BBZ     RMBIT,NOFILE        *****
034 040 11D007                  BSL     GDLID               *****
035 043 909E0851                BBZ     RMBIT,NODATA        *****
036 047 1854                    BSL     GETSYM              *****
037                     DI80    EQU     *                   *****
038 049 111007                  BSL     RETIX
039 04C 909E0851                BBZ     RMBIT,NOITEM
```

Fig 8.4, Part 1

```
040 050 1E11                    B       DI20
041                     NOFILE  EQU     *               *****
                        NODATA  EQU     *               *****
                        NOITEM  EQU     *
                        OUT     EQU     *
045 052 10100A                  ENT     MD999
046                     *
047                     GETSYM  EQU     *
048 055 E07AE8                  MOV     BMSBEG,BMS
049 058 6A0801                  SCD     IB,X'01'
050 05B 2A                      DEC     IB
051 05C 6A80A1                  MIID    IB,BMS,X'A1'
052 05F 48FE20                  MCC     AM,BMS
053 062 14                      RTN     *
    FFE 27C6
EOF
```

Fig 8-4, part 2

8.6 RETRIEVING FROM A DICTIONARY FILE

The mode is now able to retrieve items from the current file, or it can go to another file's data level to fetch an item. It cannot, however, retrieve an item from a file's dictionary level.

Extend your chain of items from the data level of the file into the dictionary level:

In the FF data level file

```
     T9
001: DICT FF T10
```

In the FF dictionary level file

```
     T10
001: T11

     T11
001: T12

     T12
001: FF T13
```

In the FF data level file

```
     T13
001: T14

     T14
001: T15
```

```
      T15
001:
```

Change your mode to test for the keyword DICT before a file name. If the keyword is present, the mode should fetch items from the dictionary level rather than from the file's data level.

In ASM read sections 3.5 and 4, and in CPU read about the following instructions:

```
    Bc    Wi,Wj,L    - BRANCH ON WORD COMPARE

    SB    B          - SET BIT

    ZB    B          - ZERO BIT
```

IN SSS, section 2, recall that address register BMS is R8.

READ AND RECALL***READ AND RECALL***READ AND RECALL

The majority of the mode's logic need not change. The following is the pertinent section with the new steps marked by asterisks at the left.

```
                Call WRTLIN to print the item's first line.

    **          Zero bit DFLG (unused by any of the called
                subroutines) to indicate data level.

    **  DI30 .

                Call GETSYM to get a symbol.
```

```
**          Move BMSBEG to BMS to point to the symbol -1.

**          Increment BMS to point to the symbol.

**          If the double word pointed at by R8 (BMS) does
            not contain C'DICT', go to DI40. (The BU
            D,D,L instruction can be used with the literal
            C'DICT' as the first operand and the
            combination R8;D0 as the second.)

**          Set bit DFLG to indicate dictionary level.

**          Go to DI30.

**          DI40 .

            If IB points to an attribute mark, go to DI80.

            .
            .
            .

            Call GBMS to get the file's dictionary-level
            base, modulo, and separation.

            If RMBIT equals zero, go to NOFILE.

**          If DFLG is set, go to DI60.

            Call GDLID to get the file's data-level base,
            modulo, and separation.

            If RMBIT equals zero, go to NODATA.

**          DI60 .
```

```
        Call GETSYM to get a symbol.

        DI80 .
        .
        .
        .
```

After adding the new logic to your mode, assemble and load it. List the mode using the M option to see the macro expansions. Compare the listing with Figure 8-5. Test the mode by invoking the verb:

```
        DISP FF T1
```

```
001 000 7FF00151                FRAME 337
    001
                        *
                        *23JUL80
                        *
                        *
006                     *MOORE
007                     *
008 001 1E02                    B       !DISP
009                     *
010                     !DISP   EQU     *
011 003 80F7                    SB      PAGINATE
012 005 928D000A                BBS     NBIT,DI10
013 009 80CC                    SB      PAGFRMT
014                     DI10    EQU     *
015 00B 928F0811                BBZ     PBIT,DI20
016 00F 118033                  BSL     SETLPTR
017                     DI20    EQU     *
018 012 166A                    MOV     IR,IB
019 014 36                      INC     IR
020 015 46FE0865                BCE     AM,IR,OUT
021 019 25                      DEC     IR
022 01A E0EAEB                  MOV     OBBEG,OB
023 01D 66B0A0                  MIID    IR,OB,X'A0'
024 020 23                      DEC     OB
025 021 112006                  BSL     WRTLIN
026 024 7083                    ZB      DFLG                    *****
027                     DI30    EQU     *                       *****
028 026 1868                    BSL     GETSYM
029 028 E07AE8                  MOV     BMSBEG,BMS              *****
030 02B 38                      INC     BMS                     *****
031 02C F13C9800                BU      C'DICT',R8;D0,DI40      *****
    030 5035
032 032 8083                    SB      DFLG                    *****
033 034 1E25                    B       DI30                    *****
034                     DI40    EQU     *                       *****
035 036 4AFE085C                BCE     AM,IB,DI80
036 03A F0288030                MOV     MBASE,BASE
037 03E F02A8032                MOV     MMODMSEP,MODULOSEPAR
038 042 111007                  BSL     RETIX
```

Fig 8-5, part 1

```
039 045 909E0865            BBZ   RMBIT,NOFILE
040 049 113003              BSL   GBMS
    04C 909E0865            BBZ   RMBIT,NOFILE
    050 90830054            BBS   DFLG,DI60          *****
    054 11D007              BSL   GDLID
    057 909E0865            BBZ   RMBIT,NODATA
045                 DI60    EQU   *                  *****
046 05B 1868                BSL   GETSYM
047                 DI80    EQU   *
048 05D 111007              BSL   RETIX
049 060 909E0865            BBZ   RMBIT,NOITEM
050 064 1E11                B     DI20
051                 NOFILE  EQU   *
052                 NODATA  EQU   *
053                 NOITEM  EQU   *
054                 OUT     EQU   *
055 066 10100A              ENT   MD999
056                 *
057                 GETSYM  EQU   *
058 069 E07AE8              MOV   BMSBEG,BMS
059 06C 6A0801              SCD   IB,X'01'
060 06F 2A                  DEC   IB
061 070 6A80A1              MIID  IB,BMS,X'A1'
062 073 48FE20              MCC   AM,BMS
063 076 14                  RTN   *
    077 00
    078 44494354
    FFE 35E2
```

Fig 8.5, part 2

9 OUTPUTTING MESSAGES

9.1 OUTPUTTING ERROR MESSAGES

Although the mode that you have developed tests for error conditions, the logic for each test returns to the same point in WRAPUP. WRAPUP has several entry points providing different interfaces with the ERRMSG file. In SSS in the functional description for WRAPUP read about entry points MD995 and MD99. In PRM, APPENDIX B, note messages 13, 111, and 201.

READ AND NOTE***READ AND NOTE***READ AND NOTE

Entry MD995 expects a message number in C1 and a parameter in a buffer pointed at by BMSBEG. Since the mode has BMSBEG pointing at the file name at NOFILE and at the item-id at NOITEM, use entry MD995 to print an error message for these two conditions. At location NOFILE move the number 201 into C1 before branching to MD999; at NOITEM move 111 into C1 before branching.

Error message 13 does not require a parameter so to have it printed, move 13 into REJCTR and transfer control to MD99. The following steps specify the logic in more detail:

```
          NOFILE    .

     **         Move 201 to C1.

     **         External branch to MD995.
```

```
NODATA    .
**        Move 13 to REJCTR.
**        External branch to MD99.
NOITEM    .
**        Move 111 to C1.
**        External branch to MD995.
OUT       .
          External branch to MD999.
```

Add the new logic to your mode and compare with the listing in Figure 9-1. To test your mode's error checking logic add an item to the chain of items. Let it reference an non-existent item, for instance,

```
     T15
001: FF NILITEM
```

When you invoke the verb DISP with the first item of the chain, all items should be printed until this one is processed. The message

```
[111] ITEM 'NILITEM' IS NOT ON FILE
```

should be displayed on the terminal. Likewise, if a non-existent file is referenced,

```
     T15
001: XX T1
```

the following message will be displayed on the terminal:

```
[201] 'XX' IS NOT A FILE NAME
```

To test the lack of a DL/ID item create a file:

```
CREATE-FILE (GG 1,1 1,1)
```

Edit two items in GG's dictionary:

```
ED DICT GG DLID DL/ID
```

When the Editor presents DLID for editing, enter

```
ME999"DL/ID"1
FI
```

When the Editor presents DL/ID for editing, type

```
FD
```

to delete the item.

If you change your item-chain to reference an item in file GG:

```
T15
001: GG XX
```

The message

```
[13] DATA LEVEL DESCRIPTOR MISSING
```

should be listed. To restore GG's DL/ID item reverse the

```
001 000 7FF00151                FRAME 337
    001
                    *
                    *24JUL80
                    *
                    *
006                 *MOORE
007                 *
008 001 1E02                    B       !DISP
009                 *
010                 !DISP       EQU     *
011 003 80F7                    SB      PAGINATE
012 005 928D000A                BBS     NBIT,DI10
013 009 80CC                    SB      PAGFRMT
014                 DI10        EQU     *
015 00B 928F0811                BBZ     PBIT,DI20
016 00F 118033                  BSL     SETLPTR
017                 DI20        EQU     *
018 012 166A                    MOV     IR,IB
019 014 36                      INC     IR
020 015 45FE087A                BCE     AM,IR,OUT
021 019 26                      DEC     IR
022 01A E0EAEB                  MOV     OBBEG,OB
023 01D 6580A0                  MIID    IR,OB,X'A0'
024 020 2B                      DEC     OB
025 021 112006                  BSL     WRTLIN
026 024 7083                    ZB      DFLG
027                 DI30        EQU     *
028 026 187D                    BSL     GETSYM
029 028 E07AE8                  MOV     BMSBEG,BMS
030 02B 38                      INC     BMS
031 02C F1469800                BU      C'DICT',R8;D0,DI40
    030 5035
032 032 8083                    SB      DFLG
033 034 1E25                    B       DI30
034                 DI40        EQU     *
035 036 4AFE085C                BCE     AM,IB,DI80
036 03A F0288030                MOV     MBASE,BASE
037 03E F02A8032                MOV     MMODMSEP,MODULOSEPAR
038 042 111007                  BSL     RETIX
```

Fig 9-1, part 1

```
039 045 909E0865            BBZ   RMBIT,NOFILE
040 049 113003              BSL   GBMS
041 04C 909E0865            BBZ   RMBIT,NOFILE
042 050 9083005A            BBS   DFLG,DI60
043 054 11D007              BSL   GDLID
044 057 909E086C            BBZ   RMBIT,NODATA
045                 DI60    EQU   *
046 05B 187D                BSL   GETSYM
047                 DI80    EQU   *
048 05D 111007              BSL   RETIX
049 060 909E0873            BBZ   RMBIT,NOITEM
050 064 1E11                B     DI20
051                 NOFILE  EQU   *
052 066 F2014148            MOV   201,C1          *****
053 06A 10300A              ENT   MD995           *****
054                 NODATA  EQU   *
055 06D F058414A            MOV   13,REJCTR       *****
056 071 10000A              ENT   MD99            *****
057                 NOITEM  EQU   *
058 074 F2014149            MOV   111,C1          *****
059 078 10300A              ENT   MD995           *****
060                 OUT     EQU   *
061 07B 10100A              ENT   MD999
062                 *
063                 GETSYM  EQU   *
064 07E E07AE8              MOV   BMSBEG,BMS
065 081 6A0801              SCD   IB,X'01'
066 084 2A                  DEC   IB
067 085 6A80A1              MIID  IB,BMS,X'A1'
068 088 48FE20              MCC   AM,BMS
069 08B 14                  RTN   *
    08C 44494354
    090 00C9
    092 006F
    094 000D
    FFE 387F
EOF
```

fig. 9-2, Part 2

procedure used above. That is, enter the Editor with the command

```
ED DICT GG DL/ID DLID
```

Merge the contents of DLID into DL/ID and delete DLID.

9.2 PUTTING PARAMETERS INTO HS FOR WRAPUP

In SSS in WRAPUP's input interface section read about the contents of HSBEG and HSEND and the contents of the buffer to which they point, especially the output message format. Also read the description of subroutine PRTERR.

READ***READ***READ

Copy error message 4 to the terminal by entering the command

```
CT ERRMSG 4
```

Error message 4 contains the following:

```
      4
001   HMODE '
002   A
003   H'
004   S(18)
005   HCHECKSUM ERROR; FRAME =
006   R(4)
007   H  MODE =
008   R(4)
009   H ABS =
010   R(4)
```

Study PRTERR's functional description and the description of TS in the input interface.

STUDY***STUDY***STUDY

Error message 4 is representative of the error messages in file ERRMSG. When you design a program that requires error messages, you should review PRM, Appendix B, for messages that can be used with your program. If new messages are required, they should be added to ERRMSG.

As an exercise, write a TCL-II mode that will set parameters into work area HS for message 4. Lines 2, 6, 8, and 10 of message 4 indicate that four parameters are needed. Use the item name pointed at by BMSBEG for the first parameter and the binary contents of RECORD, SIZE and ACF for the second, third and fourth parameters, respectively. The binary values will have to be converted by calls to MBDSUB. After the parameters are prepared, transfer control to WRAPUP at entry MD999. The detailed logic should be as follows:

Branch to !ERRT (entry point).

!ERRT .

Move HSBEG to R15. R15 is used because subsequently we are going to call MBDSUB.

Using the MCI instruction with R15, move a segment mark (SM), the letter "0", an attribute mark (AM), the number "4", and another attribute mark to the bytes pointed at by R15.

Move BMSBEG to BMS.

Move the string pointed at by BMS to the area pointed at by R15 until an attribute mark is encountered.

Load into the extended accumulator the contents of RECORD.

Call MBDSUB to convert the value in the extended accumulator to an ASCII string in the area pointed at by R15.

Move an attribute mark to the byte pointed to by R15.

Load into the extended accumulator the contents of SIZE.

Call MBDSUB to convert the value in the extended accumlator to an ASCII string in the buffer pointed at by R15.

Move an attribute mark to the byte pointed to by R15.

Load into the extended accumulator the contents of ACF.

Call MBDSUB to convert the value in the extended accumulator to an ASCII string in the buffer pointed to by R15.

Move an attribute and then a segment mark to the bytes pointed at by R15.

Move R15 to HSEND.

```
        Go to MD999.
```

Assemble, list, and load your mode. Compare it with Figure 9-2. The verb DISP can be used to invoke the mode.. For example,

```
        DISP FF MOD4
```

should result in error message 4 being printed with "MOD4" and three numbers as parameters.

```
Move the item-id to work space HS.

Call PRTMSG to print message.

Repeat

     Increment OS.

     If OS points to a '1'

     Then
          Move '1' to work space HS.

          Call PRTMSG to print message.

     Else
          If OS points to a '3'

          Then
               Move '72' to work space HS.

               Call  MBDSUB to convert SIZE to ASCII  in
               work space HS.

               Call PRTMSG to print message.

          Else
               If OS points to a '4'

               Then
                    Move '210' to work space HS.

                    Move the string in  work space OS to
                    work space HS.
```

If OS points to an attribute mark, branch to OUT.

If OS is not pointing to a '1', branch to D120.

Load the address of M1 into R14.

Move the string pointed at by R14 to the area pointed at by HS until an attribute mark is encountered.

Call PRTMSG to print the message.

Branch to DI10.

DI20 .

If OS is not pointing at a '3', branch to DI30.

Load the address of M72 into R14.

Move the string pointed to by R14 to the area pointed to by HS.

Move HS to R15. MBDSUB uses R15 to point to its output buffer.

Load the extended accumulator with SIZE. Message 72 needs a number as a parameter: SIZE is used as a convenience for this exercise.

Call MBDSUB to convert the value in the accumulator to ASCII.

Increment R15 and move an attribute mark to the byte pointed at by R15.

Move R15 to HS.

Call PRTMSG to print message.

Branch to DI10.

DI30 .

If OS is not pointing to a '4', branch to DI10.

Load the address of M201 into R14.

Move the string pointed at by R14 to the area pointed at by HS until an attribute mark is encountered.

Move OSBEG to R14.

Move the string pointed at by R14 to the area pointed at by HS until an attribute mark is reached. This string has no meaning in the message. It is just convenient for this exercise.

Call PRTMSG to print the message.

Branch to DI10.

OUT .

External branch to MD999.

PRTMSG .

Increment HS and move a segment mark to the byte pointed at by HS.

Move HSBEG to TS.

Call PRTERR, which will really print the message.

Move HSBEG to HS to initialize it.

Increment HS so that input to the buffer skips the first two bytes.

Return.

After assembling, listing, and loading your mode compare it with Figure 9-3. Verb DISP can still be used to invoke the mode:

```
DISP FF ITEM1
DISP FF ITEM2 ITEM1 ITEM3
```

```
001 000 7FF00151                FRAME 337
    001
                        *
                        *27JUL80
                        *
                        *
006                     *MOORE
007                     *
008 001 1E0F                    B       !DISP
009                     *
010                     M223    EQU     *-1
011 003 323233                  TEXT    C'223',X'FE'
    006 FE
012                     M1      EQU     *-1
013 007 31                      TEXT    C'1',X'FE'
    008 FE
014                     M201    EQU     *-1
015 009 323031                  TEXT    C'201',X'FE'
    00C FE
016                     M72     EQU     *-1
017 00D 3732                    TEXT    C'72',X'FE'
    00F FE
018                     *
019                     !DISP   EQU     *
020 010 E068E5                  MOV     OSBEG,OS
021 013 6650A0                  MIID    IR,OS,X'A0'
022 016 E05CE3                  MOV     HSBEG,HS
023 019 33                      INC     HS
024 01A E1023E                  SRA     R14,M223
025 01D 6E30A0                  MIID    R14,HS,X'A0'
026 020 E07AE8                  MOV     BMSBEG,BMS
027 023 6830A0                  MIID    BMS,HS,X'A0'
028 026 186F                    BSL     PRTMSG
029 028 E068E5                  MOV     OSBEG,OS
030                     DI10    EQU     *
031 02B 35                      INC     OS
032 02C 45FE086C                BCE     AM,OS,OUT
033 030 4531003D                BCU     C'1',OS,DI20
034 034 E1063E                  SRA     R14,M1
035 037 6E30A0                  MIID    R14,HS,X'A0'
```

```
036 03A 186F                BSL   PRTMSG
037 03C 1E2A                B     DI10
0                   DI20    EQU   *
03  03E 45330058            BCU   C'3',OS,DI30
04  042 E10C3E              SRA   R14,M72
0   045 6E30A0              MIID  R14,HS,X'A0'
042 048 163F                MOV   HS,R15
043 04A A0275C              LOADX SIZE
044 04D 110008              BSL   MBDSUB
045 050 4FFE40              MCI   AM,R15
046 053 15F3                MOV   R15,HS
047 055 186F                BSL   PRTMSG
048 057 1E2A                B     DI10
049                 DI30    EQU   *
050 059 4534002A            BCU   C'4',OS,DI10
051 05D E1083E              SRA   R14,M201
052 060 6E30A0              MIID  R14,HS,X'A0'
053 063 E068EE              MOV   OSBEG,R14
054 066 6E30A0              MIID  R14,HS,X'A0'
055 069 186F                BSL   PRTMSG
056 06B 1E2A                B     DI10
057                 OUT     EQU   *
058 06D 10100A              ENT   MD999
059                 *
060                 PRTMSG  EQU   *
061 070 43FF40              MCI   SM,HS
062 073 E05CED              MOV   HSBEG,TS
063 076 11000C              BSL   PRTERR
06  079 E05CE3              MOV   HSBEG,HS
0   07C 33                  INC   HS
06  07D 14                  RTN
    FFE 3B99
EOF
```