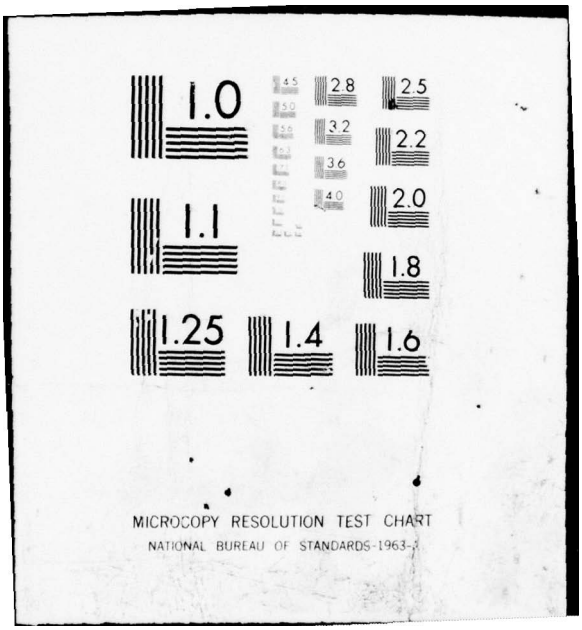
AD-A072 471 ARMY SATELLITE COMMUNICATIONS AGENCY FORT MONMOUTH NJ F/G 9/2
PROJECT ARIES. USER'S MANUALS FOR ARIAN II AND ASSOCIATED SUBSY--ETC(U)
JUL 79 R L CONN

UNCLASSIFIED NL

1 OF 3
AD A072471


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

AD A072471

DDC FILE COPY

DDC
RECEIVED
AUG 8 1979
C

This document has been approved for public release and sale; its distribution is unlimited.

79 08 06 015

AD A072421

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (6)
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle) (9) PROJECT ARIES - USER'S MANUALS FOR ARIAN II AND ASSOCIATED SUBSYSTEMS
5. TYPE OF REPORT & PERIOD COVERED
Interim rept
16 May [redacted] — 24 Jul 79
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s) (10) RICHARD L. CONN
8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
USA SATELLITE COMMUNICATIONS AGENCY
USA CORADCOM, Attn: DRCPM-SC-4G (Lt. Conn) (16)
FORT MONMOUTH, NJ 07703
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
6.11.01.A (17) 33
1L1 61101 A91A
33, 131

11. CONTROLLING OFFICE NAME AND ADDRESS
-- Same as in 9 -- (11)
12. REPORT DATE
24 July 1979
13. NUMBER OF PAGES
220

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
-- Same as in w -- (12) 237p.
15. SECURITY CLASS. (of this report)
Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for Public Release; Distribution Unlimited.

18. SUPPLEMENTARY NOTES

Project ARIES is supported by an ILIR Grant from the CORADCOM Research and Development Council.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Interactive Environment, Assembler, Microprocessor, Microcomputer, Operating System, Software Development System, Project ARIES, ARIAN, Z80

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of the research conducted under Project ARIES is to investigate and analyze an interactive assembly language software development technique which is similar in several basic concepts to the techniques employed in Project Gandalf at CORADCOM. Tests comparing the development capability and speed of conventional assembly language programming techniques (cross assembly, resident assembly via a "conventional" operating system, and cross compilation) and an early implementation of the proposed technique have

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

041 000

20. (continued) shown that this new tech-ique promises to provide a factor of from 3 to 10 decrease in program development time for assembly language programs written for the 8080 and Z80 microprocessors.

This document is a compendium of the User's Manuals for the ARIAN II Operating and Software Development System and its associated subsystems. Designed as a reference for the functional user of the system and the test and evaluation group at the University of Virginia, this document is a manual meant to be employed by the functional user of ARIAN II as a reference and tutorial on how the system can be used and what the functions supported by the system are and how they are invoked.

This manuscript is divided into ten distinct sections which cover various aspects of ARIAN II and its associated subsystems. Information is included on TFS, a text formatting system, DISASM, an 8080/pseudo-Z80 disassembler, DBUG, an interpretive simulation system, XEDIT, a dynamic screen-oriented intra-line editor, and ARIAN II itself.

USER'S MANUALS FOR ARIAN II AND ASSOCIATED SUBSYSTEMS

PROJECT *ARIES*

by

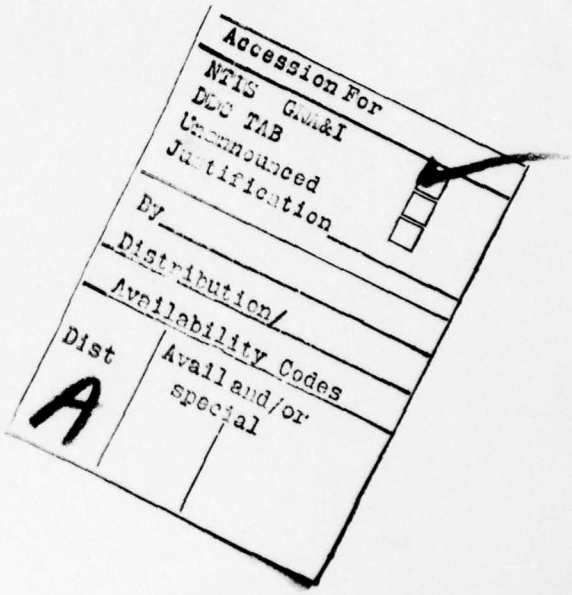
RICHARD L. CONN

U.S. ARMY
SATELLITE COMMUNICATIONS AGENCY
FORT MONMOUTH, NEW JERSEY

PROJECT ARIES
USER'S MANUALS for ARIAN II and ASSOCIATED SUBSYSTEMS

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703



24 July 1979

USA Satellite Communications Agency
Fort Monmouth, New Jersey

PROJECT ARIES

USER'S MANUALS for ARIAN II and ASSOCIATED SUBSYSTEMS

This document is a compendium of the User's Manuals for ARIAN II and its associated subsystems. Designed as a reference for the functional user of the system, this document is not a detailed manual which covers the principals of operation of ARIAN II; rather, it is a manual designed to be employed by the functional user of ARIAN II as a reference and tutorial on how the system can be used and what the functions supported by the system are and how they are invoked.

This document is divided into 10 distinct sections. Each section covers a particular feature or features of ARIAN II, and it is organized as a manual in itself. Each section contains its own title page and table of contents, and the section pages are numbered consecutively.

Richard Conn
1LT SigC

Contents

Section	Title
1	Utility PROM Reference
2	SDL -- A String Description Language
3	The USER'S MANUAL for the TEXT FORMATTING SYSTEM
4	SELECTED USER'S MANUALS The ARIAN DISASSEMBLER USER'S MANUAL DEBUG (DBUG) USER'S MANUAL MEMORY TEST USER'S MANUAL XEDIT USER'S MANUAL
5	The ARIAN II USER'S MANUAL
6	APPENDIX I: The ARIAN II EXECUTIVE COMMANDS
7	APPENDIX II: The ARIAN II BUFFERS and JUMP TABLE
8	APPENDIX III: SUMMARY of the ARIAN II ASSEMBLER
9	APPENDIX IV: The ARIAN II LEVEL 3 SYSTEM COMMANDS

Utility PROM Reference

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

18 July 1979

NAME	ADDR	INPUT REGS	REGS AFFECTED	FUNCTION AND COMMENTS
LINK	D000	-NONE-	-ALL-	LINK SUBROUTINE
INIRP	D003	-NONE-	A	RESET CHANNEL (CHAN) 1
INIP	D006	-NONE-	A	SET CHAN 1
INIR2	D009	-NONE-	A	RESET CHAN 2
INI2	D00C	-NONE-	A	SET CHAN 2
SETCI	D00F	HL	A	SET CUSTOMIZED (CUST) INPUT; HL=ADDR OF CI ROUTINE
RESCI	D012	-NONE-	A	RESET CUST INPUT
SETCO	D015	HL	A	SET CUST OUTPUT; HL=ADDR OF CO ROUTINE
RESCO	D018	-NONE-	A	RESET CUST OUTPUT
INPUT	D01B	-NONE-	A	INPUT FROM PRINCIPAL I/O CHANNEL (PIOCHAN)
OUTPUT	D0C3	A	-NONE-	OUTPUT CHAR IN A TO PIOCHAN
INPB	D021	-NONE-	A,B	INPUT FROM PIOCHAN; CHAR IN A,B
OUTB	D0C0	B	A	OUTPUT CHAR IN B TO PIOCHAN
COUT	D01E	A	-NONE-	OUTPUT CHAR IN A TO PIOCHAN; CTRL CHAR PROCESSED
COUTB	D024	B	A	OUTPUT CHAR IN B TO PIOCHAN; CTRL CHAR PROCESSED
INP1	D0CC	-NONE-	A	INPUT FROM CHAN 1; CHAR IN A
OUT1	D0CF	A	-NONE-	OUTPUT CHAR IN A TO CHAN 1
INP1B	D0C6	-NONE-	A,B	INPUT FROM CHAN 1; CHAR IN A,B
OUT1B	D0C9	B	A	OUTPUT CHAR IN B TO CHAN 1
INP2	D027	-NONE-	A	INPUT CHAR FROM CHAN 2 TO A
OUT2	D02A	A	-NONE-	OUTPUT CHAR IN A TO CHAN 2
INP2B	D02D	-NONE-	A,B	INPUT CHAR FROM CHAN 2 TO A,B
OUT2B	D030	B	A	OUTPUT CHAR IN B TO CHAN 2
INP3	D033	-NONE-	A	INPUT CHAR FROM CHAN 3 TO A
OUT3	D036	A	-NONE-	OUTPUT CHAR IN A TO CHAN 3
INP3B	D039	-NONE-	A,B	INPUT CHAR FROM CHAN 3 TO A,B
OUT3B	D03C	B	A	OUTPUT CHAR IN B TO CHAN 3
CRLF	D03F	-NONE-	A	<CR> <LF> TO PIOCHAN
CR	D0D5	-NONE-	A	<CR> TO PIOCHAN

LF	D0D8	-NONE-	A	<LF> TO PIOCHAN
PRBL	D042	-NONE-	A	<SP> TO PIOCHAN
PCCC	D045	-NONE-	-NONE-	'^C' TO PIOCHAN
BSLSH	D048	-NONE-	-NONE-	'\' TO PIOCHAN
PCC	D0D2	A	-NONE-	PRINT CHAR IN A ON PIOCHAN; IF CTRL CHAR, PRINT '^' FOLLOWED BY (CHAR+40H); <BELL>,<BS>,<LF>,<CR> ECHOED AS SUCH
PRINT	D04B	-NONE-	A	PRINT STRING ENDING IN 0 PTED TO BY RET ADR ON PIOCHAN W/CTRL CHARS
PRINTL	D0BD	HL	HL,A	PRINT STRING ENDING IN 0 PTED TO BY HL ON PIOCHAN W/CTRL CHARS; HL PTS TO ONE CHAR BEYOND 0 WHEN DONE
SCRN	D051	HL	HL,A	PRINT STRING ENDING IN <CR> (NOT PRINTED) PTED TO BY HL ON PIOCHAN W/CTRL CHARS; HL PTS TO <CR> WHEN DONE
SCRNC	D054	HL	HL,A	SAME AS SCRN, BUT <CR><LF> PRINTED
PPRINT	D04E	HL	HL,A	SAME AS SCRN, BUT PRINT ON CHAN 3
PEM	D057	-NONE-	A	PRINT '**' AND TWO CHARS PTED TO BY RET ADR
PCHR2	D05A	-NONE-	A	PRINT TWO CHARS PTED TO BY RET ADR
PCHAR	D05D	-NONE-	A	PRINT ONE CHAR PTED TO BY RET ADR
PA2HC	D060	A	-NONE-	PRINT A AS TWO HEX CHARS
PA3DC	D063	A	-NONE-	PRINT A AS THREE DEC CHARS
PA2DC	D066	A	-NONE-	PRINT A AS TWO DEC CHARS
PADC	D069	A	-NONE-	PRINT A AS ONE TO THREE DEC CHARS; <SP> FILL FOR LEADING ZEROES
PHL	D06C	HL	A	PRINT CONTENTS OF HL AS FOUR HEX CHARS
PHLB	D06F	HL	A	PRINT CONTENTS OF HL AS FOUR HEX CHARS FOLLOWED BY <SP>
PHL5D	D072	HL	A	PRINT HL AS FIVE DEC CHARS
PDE	D075	DE	A	PRINT DE AS FOUR HEX CHARS
PQ	D078	-NONE-	A	PROMPTS USER, GETS ONE CHAR FROM PIOCHAN, RET W/ZERO SET IF 'N'
INK	D07B	-NONE-	A	POLLS CHAN 1, RET IF NO CHAR, RET W/ZERO SET IF <ESC>
LMOV	D07E	DE,HL,C	DE,HL,C,A	MOVE (DE) TO (HL) UNTIL CHAR (C) FOUND IN (DE); INCR DE,HL
LMOVC	D081	DE,HL,C	DE,HL,BC	MOVE (DE) TO (HL) FOR (C) CHARS; INCR DE,HL
LMOVL	D084	DE,HL,BC	DE,HL,BC	MOVE (DE) TO (HL) FOR (BC) CHARS; INCR DE,HL
RMOV	D087	DE,HL,C	DE,HL,C,A	MOVE (DE) TO (HL) UNTIL CHAR (C) FOUND IN (DE); DECR DE,HL
RMOVC	D08A	DE,HL,C	DE,HL,BC	MOVE (DE) TO (HL) FOR (C) CHARS; DECR DE,HL
RMOVL	D08D	DE,HL,BC	DE,HL,BC	MOVE (DE) TO (HL) FOR (BC) CHARS; DECR DE,HL
MOVE	D090	DE,HL,BC	DE,HL,BC,A	MOVE BLOCK PTED TO BY DE TO HL; BC BYTES LONG

CAPS	D105	A	A	CAPITALIZE CHAR IN A IF LOWER CASE
CHTA	D093	A	A	CONVERT LOW NYBBLE OF A TO ASCII HEX CHAR EQUIV (0-9,A-F)
CATH	D096	A	A	CONVERT ASCII HEX CHAR IN A TO BINARY IN LOW NYBBLE; RET 20H IF ERROR
EN	D099	A	A	EXCHANGE NYBBLES OF A
HLMDE	D09C	HL,DE	A,BC	BC=HL-DE
RANGE	D09F	HL,DE	A,BC	BC=DE-HL+1
TABS	D0A2	-NONE-	-ALL-	SET TAB STOPS FOR READ
TABE	D0A5	-NONE-	-ALL-	EXAMINE TAB STOPS
TABR	D0A8	-NONE-	HL,BC,A	CLEAR TAB STOPS; RESET TO SYSTEM TAB STOPS
SCALE	D0AB	-NONE-	A,C,D	PRINT SCALE
READ	D0AE	-NONE-	A	READ INPUT LINE FROM PIOCHAN W/EDITING; IBUF-1=CHAR CNT, IBUF=STRING TERMINATED BY <CR> <EOL> (1); A=CHAR CNT=NUMBER OF CHARS IN LINE COUNTING <CR><EOL>
GETIN	D0B1	-NONE-	A,HL,DE	READ INPUT LINE FROM PIOCHAN; A=FIRST NON-BLANK CHAR, HL=FIRST HEX VALUE, DE=SECOND HEX VALUE
GETV	D0B4	HL	HL,DE,A	CONVERT HEX STRING PTED TO BY HL TO 16-BIT VALUE IN DE; HL PTS TO INVALID CHAR WHICH TERMINATED CONVERSION WHEN DONE
GETD	D0DB	HL	HL,DE,A	SAME AS GETV, BUT UP TO FIVE DEC CHARS CONVERTED; IF MORE THAN 5 DEC CHARS IN STRING, PTS TO 6TH CHAR
JREL	D0BA	-NONE-	-NONE-	JMP RELATIVE LONG; USAGE: CALL JREL ! DW ADDR-$
CREL	D0B7	-NONE-	-NONE-	CALL RELATIVE LONG; USAGE: CALL CREL ! DW ADDR-$
PRIO	D0DE	-NONE-	HL,A	REDIRECT I/O TO CHAN 3
VDMSET	D0E1	-NONE-	HL,A	REDIRECT OUTPUT TO VDM
VDMOUT	D0E4	A	-NONE-	DISPLAY CHAR IN NEXT LOC ON VDM; HANDLE <CR> <LF> <BS>
OUTAB	D0E7	A,B	-NONE-	OUTPUT CHAR IN A TO CHANNELS SPECIFIED BY B; FOR B, BIT 0 ON SELECTS CHAN 1, BIT 1 ON SELECTS CHAN 2, AND BIT 2 ON SELECTS CHAN 3; ANY COMBINATION OF CHANNELS MAY BE SELECTED
INK1	D0EA	-NONE-	A	POLL CHAN 1; RET W/ZERO SET IF <ESC>

INK2	D0ED	-NONE-	A	POLL CHAN 2; SAME AS INK1
INK3	D0F0	-NONE-	A	POLL CHAN 3; SAME AS INK1
INPT1	D0F3	-NONE-	A	CHECK INPUT STATUS OF CHAN 1; RET W/NON ZERO IF DATA IS PRESENT
INPT2	D0F6	-NONE-	A	CHECK INPUT STATUS OF CHAN 2; SAME AS INPT1
INPT3	D0F9	-NONE-	A	CHECK INPUT STATUS OF CHAN 3; SAME AS INPT1
LEDOUT	D0FC	A	-NONE-	OUTPUT BYTE IN A ON FRONT PANEL LEDS
OUT12	D0FF	A	-NONE-	OUTPUT CHAR IN A TO CHANS 1&2
OUT32	D102	A	-NONE-	OUTPUT CHAR IN A TO CHANS 3&2

SDL -- A String Description Language

By
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, NJ 07703

24 June 1979

Contents

Title	Page
The Rationale for SDL	1
SDL Variables and Constants	2
SDL Operators	3
SDL Statements	4
Examples	5
Summary	6

SDL -- A String Description Language

The Rationale for SDL

String Description Language, hereafter referred to as SDL, is a language which enables one to describe the format of character strings. It gives one the ability to precisely, or, if desired, imprecisely, describe a generalized character string. For example, if the user wishes to describe all character strings which consist of an arbitrary number (from 1 to n) of 'A's followed by an arbitrary number of 'B's, he may describe this precisely in SDL as follows:

```
<ABSTRING> : 'AB' ! 'A'* <ABSTRING> 'B'*
```

As the user can see, this is a rather elegant and simple description of such character strings, while the English words used to describe such strings are somewhat awkward.

SDL was developed by the author as a result of his lack of satisfaction with other string description languages such as BNF (Backus-Naur Form) and Extended BNF. SDL is a combination of what the author considers to be the good points of several string description languages and the features he finds lacking in them.

Particularly, the features carried over into SDL from other string description languages include the relatively simple form of the Extended BNF statements, the BNF and Extended BNF operators (with some modification), the form of constant strings employed by Extended BNF, and recursion.

Features incorporated into SDL which are not found in BNF and Extended BNF include the multiplicity form of the '*' and '+' operators, the use of SDL-defined non-terminal variables, and the ability to imprecisely define strings using an English description.

SDL, as described in this document, is a refinement of the SDL described in the author's MS thesis "ARIAN -- An Implementation of a Microcomputer Operating System."

SDL Variables and Constants

All constants in SDL are string constants, which take the form of zero or more characters of the ASCII character set enclosed in single quotes. The double single quote ('') is used to represent a single quote constant. Examples of SDL constants are 'abc', 'hello', ' ', '0', '1234ABC', and '*:bc1,.'.

A variable in SDL is a symbol which may be assigned a string value. Two types of variables are used in SDL -- terminal and non-terminal variables. A non-terminal variable is a string of alphanumeric characters enclosed in angle brackets (<>), and it is used to represent certain special characters (such as carriage return) and characters or strings the user wishes to use to help define other non-terminal and terminal variables in his SDL equations. A terminal variable is a string of alphanumeric characters (the first of which must be alphabetic) which is not enclosed in angle brackets. Terminal variables are final products; they may not be used to help define other variables. Non-terminal variables may be used to help describe themselves in their SDL expression. This is a recursive definition. The first example with <ABSTRING> is a recursive definition.

The underscore (_) can be used in variable names to improve readability; if used, its location in the variable name is as significant as the location of any other character in the variable name.

SDL defines a number of non-terminal variables which may be used in SDL equations. These non-terminals are:

1. <cr> or <CR> -- the ASCII carriage return character,
2. <bs> or <BS> -- the ASCII backspace character,
3. <lf> or <LF> -- the ASCII line feed character,
4. <null> or <NULL> -- the ASCII null character,
5. <empty> or <EMPTY> -- the empty or null string.

These SDL-defined non-terminal variables may be used in any SDL expression without being defined elsewhere. These symbols may also be redefined by the user at his discretion, but care must be taken not to

think of their original SDL definitions if the redefinition of the variable assigns it a meaning different from its SDL meaning.

SDL Operators

SDL allows the user to employ several operators in the SDL expressions he creates. These operators are:

Symbol	Name of Operator
"	double quote
*	asterisk
+	the plus symbol
!	the exclamation mark
?	the question mark
...	the mark of ellipsis
(	the left parenthesis
)	the right parenthesis
:	colon

The uses of these SDL operators are as follows:

1. double quote - this is used to enclose a description phrase. A description phrase is an English phrase used to describe a variable; an example of a description phrase is "THE DIGITS 0 AND 1".

2. asterisk - this is one of SDL's two multiplication symbols. It means that the preceeding symbol or expression may be repeated zero or more times. For example, <digit>* means that <digit> may be repeated zero or more times. If the asterisk is followed by a number, the number specifies the upper limit of the multiplication. For example, <digit>*4 says that there may be zero to 4 digits in the string.

3. the plus symbol - this is the other multiplication symbol. It is used exactly like the asterisk, but it specifies one or more. The number following it may also be present, indicating an upper limit. For example, <digit>+4 says that there may be from 1 to 4 <digit>s in the string.

4. the exclamation mark - this is the logical OR operator in SDL. Used in expressions, it means that the right-hand side or the left-hand side are possible. For example, <alpha> ! <digit> says that whatever is being discussed may be an <alpha> or a <digit>.

5. question mark - this means that the preceeding symbol or expression is optional. For example, <alpha>? means that an <alpha> is optional.

6. the marks of ellipsis - implies that the symbols or characters are continuous by order. For example, 'A' ! ... ! 'F' refers to 'A', 'B', 'C', 'D', 'E', 'F'.

7. parentheses - used as grouping symbols. They may be nested as deeply as desired.

8. colon - this is the assignment symbol. It must be present in every SDL statement, and it assigns the expression on its right to the variable on its left.

SDL Statements

An SDL description of a string or set of strings consists of a group of SDL statements. An SDL statement consists simply of a variable, which may be used to represent the string in question, followed by a colon and an SDL expression. An SDL expression is a grouping of variables, operators, description phrases, and ellipses which describes a string. In this grouping each variable is separated by one or more delimiters, where a delimiter is an operator, a space, or a new line.

Examples

The following is a group of examples of SDL statements:

1. <digit> : '0' ! ... ! '9'
   This statement defines <digit> to be any of the characters '0' to '9'.
2. <hexdigit> : <digit> ! 'A' ! ... ! 'F'
   This statement defines <hexdigit> to be a <digit> or one of the characters 'A' to 'F'.
3. HEX_ADDRESS : <hexdigit>*4
   HEX_ADDRESS is defined as a string of from 0 to 4 <hexdigit>s. For example, HEX_ADDRESS may be "", "ABC", "0123", "00".
4. NUMBER : <digit>+4
   NUMBER is defined as a string of from 1 to 4 <digit>s. Note that NUMBER and HEX_ADDRESS are terminal variables and may not be used in expressions other than those like " VALUE : TERMINAL ".
5. ITIS : <alpha>+ '.'?
   ITIS is defined as a string of one or more <alpha>s followed optionally by a period.
6. ENGLISH_SENTENCE : "ANY COMBINATION OF ASCII CHARACTERS"
   ENGLISH_SENTENCE is defined by a description phrase.
7. THING : <digit>+4 ! <hexdigit>+4
   THING is defined as being either from 1 to 4 <digit>s or from 1 to 4 <hexdigit>s.
8. <neat> : 'A' ! 'B' ! 'A' <neat> 'B'
   <neat> is recursively defined. Strings like 'A', 'B', 'AAB', 'AAAABBB', and 'ABBBBB' are possible.

The user will note that spaces are never implied in SDL. They must be placed in the string explicitly.

SDL -- A String Description Language

Summary

SDL is a recursive String Description Language which is used to define strings of ASCII characters. It permits implicit and explicit definition of strings, and it can be generally used to describe almost any language or set of strings. Derived from BNF and Extended BNF, it contains many features from these languages as well as several extensions made by the author.

Some features of SDL are:

1. SDL-defined non-terminal variables which need not be defined by the user.
2. The use of ellipsis to indicate items left out of a definition.
3. Multiplication of symbols or expressions with * or + may be followed by a number to indicate the limit of multiplication.
4. Recursive definition which allows a variable to be defined in terms of itself.
5. The use of a description phrase in a variable definition.

Like BNF and Extended BNF, SDL may be used to describe context free languages. It was not intended to be used as such, and SDL lacks some of the formality of BNF and Extended BNF, but the value of this application is present. SDL was designed primarily to be used as its name implies -- to describe character strings, and, as the user may note, a computer language can be referred to in terms of character strings.

The USER'S MANUAL for the TEXT FORMATTING SYSTEM

by

Richard L. Conn

USA Satellite Communications Agency

Fort Monmouth, New Jersey 07703

24 July 1979

Table of Contents

Title	Page
Chapter 1: Introduction to the Text Formatting System	1
TFS IN GENERAL	1
TFS C-SPECS	3
THE FORM COMMAND	4
Chapter 2: Output Control C-specs	5
LINE OUTPUT	6
PARAGRAPHS	7
HEADINGS and CHAPTERS	8
CENTERING	9
NUMBERING and BACKSPACING	9
COPYING	11
Chapter 3: Parameter Set C-specs	12
Chapter 4: Data File Manipulation C-specs	15
OPENING and CLOSING DATA FILES	16
READING TFS DATA FILES	16
EXAMPLES of the USE of TFS DATA FILES	17
Chapter 5: Miscellaneous C-specs, including Macros	19
APPENDING FILES	20
MACROs	21
Chapter 6: TFS User and Error Messages	22
TFS USER MESSAGES	23
TFS FATAL ERROR MESSAGES	24
TFS NON-FATAL ERROR MESSAGES	26

The Users' Manual for the Text Formatting System

CHAPTER 1

Introduction to the Text Formatting System

The Text Formatting System, hereafter referred to as TFS, is a program which produces a modified printer listing of the contents of an ARIAN file. This listing is formatted, that is, it is modified according to control specifications contained within the file itself. TFS is a program which interprets these control specifications and produces the formatted listing.

TFS IN GENERAL

These control specifications, hereafter referred to as C-specs, are commands of the form 'ZNAME', where 'Z' designates that a command name follows and 'NAME', a character string of up to four characters, is the name of the C-spec. For example, the C-spec which turns on underlining in TFS is 'ZUL'; every word which follows a ZUL is underlined until another ZUL is encountered.

Naturally, as one may notice at this point, a string starting with a 'Z' designates a C-spec, and this seems to prohibit the use of the 'Z' character as the first character of a word. This assumption, however, is not the case. An escape sequence, 'ZZ', has been designated to eliminate this problem. Any word starting with a 'ZZ' is displayed as that word beginning with a single 'Z'; for example, 'ZZthis' is printed by TFS as 'Zthis'.

A word, as defined by TFS, is a string of characters delimited by

either two blanks or a blank and a carriage return. Hence, a word is any string like 'string', where 'string' is delinited by blanks like ' string ', and words may not cross line boundaries in an ARIAN file, since ARIAN terminates each line of its files with a carriage return character (ASCII 0D hexadecimal). TFS is a word-oriented text formatter. It places words in an output line until the line is filled, and then it prints the line. TFS analyzes each word as it reads the word from the ARIAN file, checks to see if the word is a C-spec, formats the word if it is not a C-spec and executes the C-spec if it is, and then continues by reading the next available word in the file. TFS continues until it reaches the end of the file.

TFS operates in basically two modes: (1) ASIS mode (see the %ASIS command) and (2) normal mode. In normal mode, TFS will read words from the ARIAN file, building the current output line as it goes. All output lines are of a definite length, defined either by default or explicitly by the user, and TFS puts the words it reads into the current output line buffer until it reads a word too large to fit in the remaining space. If there are no words in the line at this time, it will force the word into the line and print the line; otherwise, it will output the line. In outputting a line with right justification, TFS inserts blanks between words in the output line buffer until there is a specified number of characters in the line (the length of the line). It will insert these blanks starting at the right end of the line, going to the left. As a result, the left and right margins of a page produced by TFS line up if right justification is employed. If right justification is not employed, TFS simply outputs the line without filling it. Right justification is the default.

While creating the output line in normal mode, TFS follows the convention that all words ending in a special character ('.','!',':', or '?') are followed by two blanks; all other words are followed by one blank. It is felt that this convention improves readability of the line.

In ASIS mode, TFS simply outputs the lines following the %ASIS C-spec exactly as they appear. Refer to the documentation on the %ASIS command.

TFS C-SPECS

As mentioned earlier, TFS recognizes many commands (or C-specs) during the formatting of an ARIAN file. These C-specs all take the form of '%NAME', where '%' indicates that a C-spec name follows, and 'NAME' is the name of the C-spec. The following is a list of all C-specs recognized by TFS; the chapters of this manual will describe these C-specs in detail.

Type of C-spec: Output Control
UL, RJ, NORJ, ASIS, BR, CR, P, PX, PAGE, TP, SKIP n, HEAD text, C text, CH n text, COPY n, ENDC, LOOP, ENDL, LEX, BS, and N

Type of C-spec: Parameter Set
PAR n m, PARX n m, LMAR n, LLEN n, LINE n m, PGON n, PGOF, PNUM n, SP n, and SETN

Type of C-spec: Data File Manipulation
OPEN filename, CLOS, and READ

Type of C-spec: Miscellaneous
SAV, RES, PAUS, EXIT, REM, APND, MAC, and ENDM

THE FORM COMMAND

TFS is invoked by typing the FORM (Format) command. This command causes the TFS program to be loaded into memory and executed.

The FORM command has three options. These options permit the user to: (1) have TFS stop after printing each page, thereby allowing the user to change paper, (2) to display the formatted output on the user's CRT one line at a time, and (3) to have TFS skip to a specified page and start printing the report at that page. 'FORMP' invokes the first option, while 'FORM n' or 'FORMP n', where 'n' is a page number (1-99), cause TFS to skip to the specified page and start printing on this page. 'FORMV' invokes the second option, and 'FORMV n' also functions as one would expect.

CHAPTER 2

Output Control C-specs

The output control C-specs control the output of TFS directly. They include:

1) Line control C-specs, such as %ASIS, %BR, %CR, and %SKIP.
2) Page control C-specs, such as %PAGE, %TP, and %CH.
3) Format control C-specs, such as %UL, %HEAD, %C, %RJ, %NORJ, %BS, and %N.
4) Paragraph control C-specs, such as %P and %PX.
5) and the Copy control C-specs, %COPY, %ENDC, %LOOP, %ENDL, and %LEX.

The %ASIS C-spec instructs TFS to display the following lines exactly as they are without filling the output lines. Only the spacing control is carried over the %ASIS C-spec; for instance, if the output is double-spaced when the %ASIS is encountered, the lines within the %ASIS block are also double-spaced.

An %ASIS block consists of a line of the ARIAN file to be formatted which contains the C-spec '%ASIS' followed by an optional comment, the lines to be printed "asis", and a terminating line beginning with the character '%'. The terminating line must start with the '%' character in the first valid character position of the line. For example, a typical %ASIS block would be:

```
0100 ZASIS
0200 [text to be displayed without formatting]
0300 ZASIS
```

The only restriction placed on an 'ZASIS' block is that the 'Z' character may not be placed in the first valid character position of a line without terminating the 'ZASIS' block.

The 'ZUL' C-spec is used to start and stop the underlining process. The group of characters to be underlined are enclosed in 'ZUL' C-specs. For example, a typical use of ZUL is:

```
0100 This text will not be underlined. ZUL This text will be
0200 underlined. ZUL This text will not.
```

LINE OUTPUT

Lines may be terminated in one of three ways: (1) automatically when TFS determines that the next word will not fit in the current output line, (2) in an implied way when certain TFS C-specs, such as ZP, force the output of the current line by the nature of their function, and (3) explicitly in the use of the ZBR and ZCR C-specs.

ZBR (break) and ZCR (carriage return) force the output of whatever is in the output line buffer. If more than 10 characters are required to fill the line to make the margins line up, the line is not filled -- it is output exactly as it exists in the buffer. Otherwise, the line is filled as described earlier.

ZBR breaks the output line. If the output line buffer is empty, nothing happens on the printer; if it contains something, it is printed, and a carriage return - line feed is output. ZCR always performs a carriage return - line feed. If the output buffer is not empty, it acts like a ZBR; if the buffer is empty, it outputs just a carriage return - line feed. Hence, if the user wishes to skip down one or two lines, he may insert two ZCR's at the appropriate place.

Along the same lines as ZBR and ZCR, the ZSKIP C-spec also can be used to terminate a line. ZSKIP is always to be followed by a number 1-99; it acts like the specified number of ZCR's. Unlike ZCR, however, if the end of the page is encountered before the skipping is completed, a page eject is done and the skipping is stopped. For example, if only three lines are left on the page and a ZSKIP 10 is encountered, then only a page eject will be done.

If the user wishes to skip down for the purpose of inserting a diagram in the text, ZSKIP alone may not be adequate. If the diagram is to require ten lines and only five lines are left on the page, ZSKIP will leave only five lines for the diagram.

To get around this problem, the ZTP (test page) C-spec is implemented. This C-spec, which is also always followed by a number from 1-99, tests to see if the specified number of physical lines is left on the current page, and, if such is not the case, it forces a page eject. Hence, to ensure leaving ten lines for the diagram, a ZTP 10 followed by a ZSKIP 10 may be used. If the ten lines are not available on the current page, a page eject is done followed by the skip; otherwise, just the skip is done.

Another C-spec available to the user is the ZPAGE C-spec, which forces a page eject. The advantages of this C-spec are obvious.

The normal output of TFS is right- and left-justified, and TFS provides two control C-specs which may be used to selectively engage and disengage the right justification feature. ZRJ engages right justification, and ZNORJ disengages it.

PARAGRAPHS

TFS supports basically two types of paragraphs -- normal, indented paragraphs and exdented paragraphs. Indented paragraphs are as one would expect a paragraph to be. The paragraph starts with the first word indented a specified number of characters in from the left margin. TFS permits indentation of from 0 to 99 characters.

Exdented paragraphs are displayed as the first line extending a

specified number of characters to the left of the left margin. The lists presented in this manual are formed using exdented paragraphs. Obviously, the left margin must be greater in length than the number of characters to be extended; if such is not the case, an error message is printed in the output. It makes no sense to extend a line beyond the first character space the printer can print in.

The ZPAR and ZPARX C-specs define the characteristics of the paragraphs to follow. These C-specs, which are described in detail later, set the number of characters to be indented and exdented.

The ZP and ZPX C-specs tell TFS that an indented or exdented paragraph starts with the next word. The current output line is broken (ZBR) and the new paragraph is started when these C-specs are encountered.

HEADINGS and CHAPTERS

The ZHEAD C-spec permits the user to place a heading at the top of each page if the page numbering option is on (see ZPGON). This C-spec takes the form of 'ZHEAD text' on one line of the file. No C-specs may be placed in the text following the ZHEAD C-spec, and all of the text following this C-spec to the end of the line in the file is used as the heading.

When a page eject occurs and ZPGON and ZHEAD are in effect, the first printed line contains the page number. This is followed by the number of blank lines specified by the line spacing C-spec (ZSP n -- see later) and the heading followed by one additional blank spaced line. For instance, if the output is double spaced, one blank line follows the line containing the page number and three blank lines (1 blank, 1 blank for the next normal line in spacing, and 1 blank to follow that line) follow the heading. Note the headings on the pages of this manual as examples.

The chapter C-spec, ZCH, is of the form 'ZCH n text', where 'n' is the chapter number 1-99 and 'text' is the chapter title. ZCH forces a page eject, skips down 10 physical lines, centers the word 'CHAPTER' and the chapter number, skips down two blank spaced lines (3 physical lines if double spaaed, 1 physical line if single spaced, 5 if triple, etc.), and centers the text of the chapter title. As with all centering, the next word after the ZCH C-spec must be a C-spec which breaks the output line with a carriage return (see the section on centering). For example, such a

C-spec may be ZCR, ZP, or ZPX.

CENTERING

Centering is done explicitly in TFS by using the ZC C-spec and implicitly by using the ZCH C-spec. Centering always involves breaking the current line and starting a new line. The ZC C-spec is of the form 'ZC text', where text is terminated by the end of the current line in the ARIAN file. When ZC is encountered the output buffer is broken, and the centering is done on the next physical line of the printed output. Since spacing is done after a line is printed, centered lines in a single-spaced section of text will be on the next physical line and centered lines on double-spaced sections of text will be on the second physical line following the broken line.

NUMBERING and BACKSPACING

Two C-specs included at this time are ZBS and ZN. These C-specs give the user some additional control over the format of the output that is particularly advantageous.

ZBS is the backspace C-spec. It has no arguments, and its function is to perform a backspace in the output line buffer. As each word is stored in the output line buffer, it is followed by one or two blanks. If it does not end in a terminating character like '.' (specified earlier), it is followed by one blank; if it ends in such a character it is followed by two blanks. ZBS backs up the pointer which points to the next available character position in the buffer; hence, it erases a blank following the last word placed in the buffer. ZBS effectively concatenates the next word to be placed in the buffer with the last word placed in the buffer. This is particularly useful in cases where a C-spec which affects its arguments globally must be restrained.

As a case in point, ZUL is such a C-spec. If the user wishes to underline a word, for instance, and terminate the word with a piece of punctuation which is not underlined, ZBS makes this possible. For example,

```
0100 ZUL TFS ZUL ZBS .
```

makes the string 'TFS.' possible.

Two problems with using ZBS should be noted at this time. The first problem is that if the string to be concatenated to the word in the buffer would result in a line overflow, TFS will not permit the concatenation to occur. For example, if 'stand' is concatenated to 'under' and there are only two spaces left in the output line buffer (i.e., 'understand' will overflow the right margin by three spaces), concatenation will not occur and 'stand' will appear as the first word of the next line. Hence, as a general rule, it is best to use ZBS to append a single character to the last word in the output buffer. One may safely append larger strings only if he is sure that an overflow will not occur.

The second problem is that ZBS will not work if the output buffer is empty. No significant error will occur, but the concatenation to the desired word may not occur. Specifically, if the word to be appended to was the last word of the line just printed, then the backspace will be ineffective.

Numbering is the second item covered in this section. It was included primarily because backspacing often accompanies numbering in TFS, such as in the numbering of list items.

The ZN C-spec is used to enable automatic numbering. TFS supplies a number buffer to the user; this buffer is initialized to 1 when TFS is first invoked and whenever the ZSETN C-spec (see later) is encountered. Whenever ZN is encountered the value in this buffer (1-99) is placed in the output buffer as a two-character word. If the value is between 1 and 9, the first character is a blank. After the word is placed in the output buffer, the value of the number buffer is incremented. Hence, successive occurances of ZN result in successive numbers being placed in the output line, like 'ZSETN ZN ZN ZN' results in 1 2 3. This is particularly useful in producing numbered lists, where the user may wish to insert an element into a list at a later time and does not wish to manually renumber the lists in the ARIAN file. All lists in this manual are produced by using ZN followed by a ZBS and a ')' as the first words in an extended paragraph (see the list at the beginning of this chapter).

COPYING

The last C-specs to be described in this chapter are %COPY, %ENDC, %LOOP, %ENDL, and %LEX. These C-specs allow the user to copy sections of the file up to 99 times. With these C-specs, all or selected sections of the TFS ARIAN file may be duplicated in the output.

%COPY takes the form '%COPY n', where 'n' is the number of copies (1-99) to be made. The first word of the block to be copied is the first word of the line following the line containing the %COPY C-spec. For example,

```
0100 %COPY 2 %P This
0200 will be copied twice. %ENDC This once.
```

For instance, this paragraph is copied twice by %COPY and %ENDC.
For instance, this paragraph is copied twice by %COPY and %ENDC.

The entire file may be copied by placing %COPY after the last macro definition in the file (see the section on macros later) and by placing %ENDC as the last word in the file.

The %LOOP and %ENDL C-specs perform the same type of function as %COPY and %ENDC, but they establish an "infinite" loop. This loop can only be terminated by a loop exit C-spec, %LEX, or a TFS exit C-spec, %EXIT. The %LOOP and %ENDL C-specs find their value in applications where the user doesn't know how many copies he wants, the user wants more than 99 copies, and the user is generating his copies from a data file and he is terminating the process by placing a %LEX or an %EXIT C-spec in the data file. All words following %LEX in a Data File record are ignored.

%LOOP and %ENDL, like %COPY and %ENDC, bracket the text to be repeated. %LEX must appear somewhere within a loop bracketed by %LOOP and %ENDL; an error is given and processing is terminated if %LEX does not appear within such a loop.

Loops may not extend beyond the resident TFS source. That is, a loop may not be begun in one TFS source file, an append be performed, and the loop is terminated in the appended file. Any attempt to do so results in a TFS fatal error.

CHAPTER 3

Parameter Set C-specs

The Parameter Set C-specs are used to assign values to the various control settings used by TFS. These C-specs include:

1) Paragraph parameter C-specs, such as ZPAR and ZPARX.
2) Line parameter C-specs, such as ZLMAR and ZLLEN.
3) Page parameter C-specs, such as ZLINE, ZPGON, ZPGOF, and ZPNUM.
4) the Spacing parameter C-spec, ZSP.
5) and the N parameter C-spec, ZSETN.

All Parameter Set C-specs except ZPNUM (see below) are effective immediately. For example, once ZPAR is used, all subsequent paragraphs created by ZP are affected; additionally, the current paragraph is also affected immediately.

The paragraph parameter C-specs set the indentation or exdentation number and the number of spaced lines to be placed between paragraphs. Both ZPAR and ZPARX have two numeric arguments -- the first sets the indentation or exdentation and the second sets the number of spaced lines to be placed between paragraphs. As in most numeric parameters, these may

take on the values from 1 to 99. For example, 'ZPAR 5 1' establishes an indentation of 5 spaces and the number of spaced lines to 1. If the output is double spaced, three blank lines (1 associated with the last line of the paragraph and 2 associated with the skipped line) are placed between each paragraph. Indented and exdented paragraphs are discussed in the previous chapter.

The ZLMAR and ZLLEN C-specs set the location of the left margin and the length of the output line. 'ZLMAR n' sets the left margin at 'n' characters right of the physical left end of the carriage; 'ZLLEN n' sets the length of the line to 'n' characters, starting at the current position of the left margin.

The effects of these C-specs are order-dependent to some extent. Since ZLLEN sets the position of the right margin based upon its argument and the position of the left margin and ZLMAR ignores the length of the line, if an ZLMAR C-spec is executed after a ZLLEN C-spec, the new line length is the difference between the old line length and the new left margin. For example, if 'ZLLEN 80 ZLMAR 10' is encountered in the text, the new line length would be 70. However, if 'ZLMAR 10 ZLLEN 80' is encountered, the new line length would be 80. The ZLLEN C-spec adds its argument to the current position of the left margin.

The ZLINE C-spec sets the format of the output page. ZLINE has two arguments -- the number of physical text lines on the output page and the number of physical lines on the output page. For instance, 'ZLINE 40 51' specifies that there are to be 40 physical text lines (including spacing) on a page and 51 physical lines on a page. Hence, when TFS is started and this instruction is encountered, TFS starts counting down from 40 with the line on which the user set the Top-of-Form, and, when it has counted 40 lines, it skips down 11 (51-40) for the next page.

The ZPGON and ZPGOF turn on and off the page numbering and heading facilities of TFS. ZPGON has one argument, the column number from which the page numbers will be right-justified. For example, 'ZPGON 60' right-justifies the page numbers in column 60, so page 1 will print the 1 in column 60 and page 10 will print the 0 in column 60 and the 1 in 59. The heading facility described earlier is turned on by this command, so the heading buffer should be loaded by the ZHEAD command (described earlier) before ZPGON is encountered. ZPGOF turns off the page numbering and heading facility. Note that paging itself is always engaged with TFS.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROJECT ARIES - USER'S MANUALS FOR ARIAN II AND ASSOCIATED SUBSYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Interim 16 May 79 - 24 Jul 79
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) - RICHARD L. CONN		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS USA SATELLITE COMMUNICATIONS AGENCY USA CORADCOM, Attn: DRCPM-SC-4G (Lt. Conn) FORT MONMOUTH, NJ 07703		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6.11.01.A 1L1 61101 A91A 33, 131
11. CONTROLLING OFFICE NAME AND ADDRESS -- Same as in 9 --		12. REPORT DATE 24 July 1979
		13. NUMBER OF PAGES 220
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) -- Same as in w --		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for Public Release; Distribution Unlimited.

18. SUPPLEMENTARY NOTES

Project ARIES is supported by an ILIR Grant from the CORADCOM Research and Development Council.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Interactive Environment, Assembler, Microprocessor, Microcomputer, Operating System, Software Development System, Project ARIES, ARIAN, Z80

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The objective of the research conducted under Project ARIES is to investigate and analyze an interactive assembly language software development technique which is similar in several basic concepts to the techniques employed in Project Gandalf at CORADCOM. Tests comparing the development capability and speed of conventional assembly language programming techniques (cross assembly, resident assembly via a "conventional" operating system, and cross compilation) and an early implementation of the proposed technique have

20. (continued) shown that this new tech-ique promises to provide a factor of from 3 to 10 decrease in program development time for assembly language programs written for the 8080 and Z80 microprocessors.

This document is a compendium of the User's Manuals for the ARIAN II Operating and Software Development System and its associated subsystems. Designed as a reference for the functional user of the system and the test and evaluation group at the University of Virginia, this document is a manual meant to be employed by the functional user of ARIAN II as a reference and tutorial on how the system can be used and what the functions supported by the system are and how they are invoked.

This manuscript is divided into ten distinct sections which cover various aspects of ARIAN II and its associated subsystems. Information is included on TFS, a text formatting system, DISASM, an 8080/pseudo-Z80 disassembler, DBUG, an interpretive simulation system, XEDIT, a dynamic screen-oriented intra-line editor, and ARIAN II itself.

.ZPNUM sets the number of the next page to be printed to the value of its argument. Unlike the other C-specs, .ZPNUM is not effective immediately -- its value applies to the next page rather than the current page. Hence, '.ZPNUM 5 .ZPAGE' would result in the page supplied by the page eject (.ZPAGE) command having a number of 5. .ZPGON must be in effect for this C-spec to work.

.ZSP sets the spacing of the line. This C-spec is of the form '.ZSP n', where 'n' may take on values from 1-99. '.ZSP 2' sets double spacing, '.ZSP 1' sets single spacing, etc. '.ZSP 0' is not recommended since results are sometimes unpredictable.

Finally, .ZSETN sets the value of the number buffer to 1. This C-spec is used to initialize the number buffer for subsequent uses of the .ZN C-spec (described earlier).

CHAPTER 4

Data File Manipulation C-specs

TFS supports two basic designations of files -- TFS Source Files and TFS Data Files. Physically, there is no difference between these types of files; both may be created by ARIAN II in the same way, and they may be stored on disk and referenced by TFS. The difference between these two files is in their application. TFS Source Files are processed directly and normally by TFS. TFS Data Files, however, are processed indirectly by TFS; while processing a TFS Source File, elements from the TFS Data File are inserted into the output listing one record at a time, a record being defined as one line of a TFS Data File.

The TFS C-specs which are used to manipulate TFS Data Files are:

1) The %OPEN C-spec, which is used to open a Data File,
2) The %CLOS C-spec, which is used to close a Data File, and
3) The %READ C-spec, which is used to read a record from a Data File.

OPENING and CLOSING DATA FILES

Before a TFS Data File may be used, it must be opened. This function serves to locate the data file, initialize the resident buffer for it, and set a number of internal variables. This is done by using the ZOPEN C-spec. This C-spec is of the form 'ZOPEN FILENAME'. Only one data file may be opened at one time.

Similarly, when a user has finished with one data file and wishes to use another, the ZCLOS C-spec is used to close an opened Data File, thereby enabling the user to open another Data File. The ZCLOS C-spec is simply of the form 'ZCLOS'.

Data Files must reside on the Logged-In Disk Drive, and the user must take care not to change disks while a data file is in use. TFS makes no checks to see if this was done, and subsequent loads of the Data File buffers will come from the disk addresses of the data file on the original disk if this occurs. No prediction can be made as to the results.

READING TFS DATA FILES

Once a Data File has been opened, the user will want to read records from it and insert them into his output. This is done by using the ZREAD C-spec.

A record takes the form of a line in the data file. When a ZREAD is encountered, the internal TFS word pointer is saved, and it is then set to point at the first word of the next record in the data file. Words are then pulled from the data file until the end of the record (end of line) is reached. At this time, the original word pointer is restored and processing continues with the word following the ZREAD. An arbitrary number of words, including none at all, may exist in a record within the data file.

The records of a data file, then, are read and processed exactly as if they simply existed within the source file. This greatly extends the flexibility of TFS, since words within a record may be TFS commands. Only the ZREAD, ZLOOP, ZCOPY, and ZMAC and ZENDM commands may not exist within a

Data File, and TFS will generate a fatal error if one is encountered.

EXAMPLES of the USE of TFS DATA FILES

The applications of TFS Data Files, then, are enormous. With the ability to contain TFS commands, the power of a TFS Data File is greatly extended over that of conventional data files.

Two applications are of particular interest. The first is in creating a Data File which may be used to provide a common initialization of a TFS Source File. If several TFS Source Files require the same type of initialization, a TFS Data File may be created containing the proper statements, and it may be read by the Source Files in an infinite loop. The Data File, of course, would have %LEX as its last word.

Specifically, the following illustrates such an application:

TFS Source File	TFS Data File
0005 %LOOP %READ %ENDL	0005 %LMAR 0 %LLEN 85 %PGON 65
0010 %REM continue source	0010 %SETN %LEX

In this example, the loop in line 0005 of the TFS Source File is executed, reading the words in the TFS Data File one word at a time. Each time the %READ is executed, a line from the data file is read and processed. This loop will execute twice, and, when the %LEX C-spec is encountered, the processing will continue in the source file on line 0010.

The second application is in processing mailing lists. The following illustrates such an application:

```
TFS Source File                TFS Data File
---------------                -------------

0001 ZOPEN DATA1 ZREM Open Data File
0005 ZLOOP ZCOPY 4 ZREAD       0005 Mr. James Jones
0010 ZCR ZENDC Dear ZREAD ZBS , 0010 43 Ocean Avenue
0015 ZP Hello. How are you     0015 Apt. 4
0020 today?                    0020 Sea Bright, NJ
0025 ZASIS                     0025 Jim
0030           Sincerely,      0030 Mr. John Smith
0035           Your Friend     0035 29 Queens Blvd
0040 ZASIS                     0040
0045 ZPAGE ZENDL               0045 Sea Bright, NJ
0050 ZP This line is extra.    0050 John
0055 ZCLOS                     0055 ZLEX
```

In this example, the TFS Data File is a mailing list, consisting of four lines of address and a fifth line containing the person's first name. Note that the Data File ends with a ZLEX C-spec, and control is transferred to line 0050 of the Source File when this is encountered.

Line 0005 of the Source File reads the four lines of the address and inserts them into the output. The ZCR on line 0010 of the Source File terminates each line read, and the ZENDC terminates the copy.

After the address is read, the word "Dear" is output at the beginning of the fifth line (note the last ZCR before the ZENDC) followed by the name of the person from the fifth, tenth, etc., lines of the Data File. The internal pointer is backed up by the ZBS C-spec, and a comma is placed immediately after the inserted name.

The body of the letter is then output, followed by the page eject on line 0045 of the Source File.

Here, then, are just two examples of the use of Data Files in TFS. As the user can see, the possibilities are enormous.

CHAPTER 5

Miscellaneous C-specs, including Macros

The following are the miscellaneous C-specs supported by TFS:

1) The environmental control C-specs, %SAV and %RES,
2) The pause C-spec, %PAUS,
3) The exit C-spec, %EXIT,
4) The comment C-spec, %REM,
5) The append C-spec, %APND, and
6) The Macro C-specs, %MAC and %ENDM.

The %SAV and %RES C-specs are used to save and restore the TFS environment, respectively. The TFS environment consists of the following:

1) The number of lines to skip between paragraphs,
2) The right justification flag, which indicates if right justification is turned on,
3) The current page number,
4) The paging flag, which indicates if page numbering is currently turned on,
5) The indent and exdent counts,
6) The left margin setting,
7) The length of the output line,
8) The underline flag, which tells if underlining is engaged,

9) The heading flag, which tells if a heading is currently set,
10) The line spacing, and
11) The current value of the number buffer.

Whenever ZSAV is encountered, these values are saved in a reserve buffer. Their values remain unchanged. At this time, the user may change whichever values he wishes. When he wishes to restore the saved environment, he simply enters the ZRES command.

The ZPAUS C-spec is used to send a message to the user and suspend operation of TFS until the user responds by typing any key on the keyboard of the principal I/O device. It takes the form of ZPAUS <text>, where <text> is terminated by the end of the current line. If the user types an <ESC> after the message is displayed, TFS will abort.

The ZEXIT C-spec is used to terminate interpretation of the TFS source immediately. It forces a page eject and then returns to the operating system. It may be used for many reasons, but a very good use of this C-spec is to terminate infinite loops and the corresponding output when reading a data file.

The ZREM C-spec allows the user to enter a remark, or comment, into the file. The comment starts with the first character following the word 'ZREM' and continues to the end of the line in the file. The next line is then interpreted normally.

APPENDING FILES

In some cases, the user may wish to load another file and continue formatting in the same environment as the previous file (i.e., he may wish to have the same line length, the same paragraph settings, etc.). The ZAPND C-spec was created to permit the user to do this. This command gives the TFS user the additional flexability of producing a report which spans over several files.

The ZAPND C-spec is of the form 'ZAPND FILENAME'. Upon encountering

this C-spec, TFS clears the local file space and loads the specified file. It then continues formatting at the first word of the loaded file.

All the environmental attributes of the previous file are preserved, but the macros need to be redefined. Hence, the ZAPND C-spec easily allows the user to chain several files into one formatted listing.

MACROs

Macros are essentially subroutines placed within the TFS formatting file. Each macro is of the form of the ZMAC C-spec followed by the name of the macro; this name must be four characters or less -- any additional characters will be discarded. The first word following the macro name is the first word of the macro. The contents of the macro include all words following its name up to and including the ZENDM termination word.

Macros are not executed when they are defined. They are executed only when their names are referenced. For example, if the TFS file contained:

```
0100 ZMAC TFS ZUL TFS ZUL ZBS , ZENDM
...
1000 ZTFS you see
```

the phrase '<u>TFS</u>,' would only be printed when line 1000 was encountered.

All macros must be defined before they are first referenced. TFS is a one-pass formatter, so the table of macro names is only formed as the macros are defined.

The nesting of macros is permitted. Macro definitions, however, may not be nested, but one macro may call another macro which in turn calls another. This type of nesting is permitted up to ten levels deep. An indirect recursion is not checked for by TFS, and such a situation could be disasterous. Up to twenty macros may be defined.

The best way to discover exactly what macros can do is to try them. The macro facility of TFS is indeed very powerful, and it can be of enormous value in creating formatted text.

CHAPTER 6

TFS User and Error Messages

There are three types of messages issued by TFS: (1) User Messages, which are displayed on the principal I/O device, (2) Fatal Error Messages, which are displayed on the principal I/O device, and (3) Non-Fatal Error Messages, which are included in the output. This chapter identifies and describes the meaning of all TFS messages.

TFS USER MESSAGES

Message	Meaning
SET TOF -- TYPE ANY CHAR TO CONTINUE, <ESC> TO ABORT	Set the Top of Form (TOF) on the printer; to continue, type any character other than <ESC>; type <ESC> to abort TFS and return to the operating system
INSERT NEXT PAGE	Insert the next page into the printer; this message appears when 'FORMP' is used. When ready, the user types any character to continue or <ESC> to abort TFS and return to the operating system.
TYPE ANY CHAR WHEN READY, <ESC> TO ABORT	This appears when a ZPAUS C-spec is used. Again, the user types any character other than <ESC> to continue and <ESC> to abort TFS and return to the operating system.

TFS FATAL ERROR MESSAGES

Message	Meaning
$ DATA FILE NOT FOUND	The Data File named in an ZOPEN C-spec was not found on the Logged-In Drive.
$ DATA FILE NOT OPEN	A ZREAD was attempted when a Data File had not been opened.
$ EOF OF DATA	An attempt was made to read past the end of the opened Data File.
$ INVLD CMND IN DATA FILE	One of the restricted commands was encountered in the opened Data File. See Chapter 4 for a list of these commands.
$ LOOP ERR	An ZENDL or ZENDC was encountered without a preceeding ZLOOP or ZCOPY. This will also occur if the ZLOOP or ZCOPY is in one Source File and the ZENDL or ZENDC is in an appended Source File.
$ MACRO RET ERR	An error has occurred in the user's MACRO call structure. An ZENDM was encountered without a corresponding ZMAC.
$ NO SUCH FILE	The file referenced in an ZAPND C-spec

The Users' Manual for the Text Formatting System

was not found on the Logged-In Drive.

TFS NON-FATAL ERROR MESSAGES

Message	Meaning
$$INVLD NUM	An error exists in the TFS Source File after a command which expects a numeric constant. Such commands are %CH, %TP, etc.
$$XDENT ERR	A %PX was attempted, and the resulting starting location of the Xdented Paragraph was before the first column of the printer.

SELECTED USER'S MANUALS

by
Richard L. Conn

Due to the short length of these manuals, the following are included in one section.

The ARIAN DISASSEMBLER USER'S MANUAL

DEBUG (DBUG) USER'S MANUAL

MEMORY TEST USER'S MANUAL

XEDIT USER'S MANUAL

The ARIAN DISASSEMBLER USER'S MANUAL

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

17 July 1979

The ARIAN DISASSEMBLER USER'S MANUAL

The ARIAN Disassembler is a Level 3 Command which is a subsystem in itself. It is invoked by engaging Command Level 3 and issuing the DASM command.

A Disassembler is a program which decodes data in memory as instructions and displays the meaning of these instructions to the user mnemonically. It does the reverse of an Assembler. The Assembler takes a line of assembly language code, like 'MVI A,4', and translates it into machine code, like '3E 04'. The Disassembler, on the other hand, takes machine code, like '3E 04', and converts it into the equivalent assembly language mnemonics, like 'MVI A,04'.

That is the type of function performed by DASM, with the addition of memory examine functions. The ARIAN Disassembler program responds to MCS-like commands (refer to "The Monitor Command System User's Manual") and executes them. The SDL description of these commands is as follows:

```
<blank> : ' '
<hexdigit> : '0' ! ... ! '9' ! 'A' ! ... ! 'F'
<address> : <hexdigit>+4
DASM_COMMAND : (( 'D' ! 'E' ! 'P' ) <blank>* <address> <blank>+
    <address> ) ! 'X'
```

As can be seen, the ARIAN Disassembler recognizes four commands. In detail, these commands are:

D -- Disassemble the block of memory bounded by the two specified addresses. The first address should, but not necessarily must, be less than the second. The disassembly will start at the first address and continue until the second address is reached or passed. For instance, if the second address is the first byte

of a 2- or 3-byte machine code, the entire code will be disassembled.

E -- Examine the block of memory bounded by the two specified addresses. This is similar to the MCS block memory examine command.

P -- Print as ASCII characters the block of memory bounded by the two specified addresses. As each byte is encountered, if it has a value between 20 and 60 hexadecimal, inclusive, its ASCII character equivalent will be printed; otherwise, the hexadecimal value will be printed.

X -- Exit and return to the calling program (ARIAN II).

In all cases, the output is paged as in ARIAN II and MCS.

The mnemonics used to display the disassembled code are the same as those used by the ARIAN II Assembler. Refer to Appendix III of "The ARIAN II USER'S MANUAL" for a detailed description of these mnemonics.

DEBUG (DBUG) USER'S MANUAL

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

1 July 1979

DEBUG (DBUG) USER'S MANUAL

DBUG is a Level 3 command subsystem which may be run under ARIAN. It is designed to be used to assist in debugging a user program.

DBUG is a complete simulation system for the 8080 microprocessor. It loads 8080 machine code instructions from the microcomputer memory into an execution area and executes these instructions one at a time in an artificial environment. Certain 8080 instructions, such as HLT, IN, OUT, PCHL, JMP, Jc, CALL, Cc, RET, and Rc, are trapped by DBUG and artificially executed under DBUG control. The rest of the 8080 instructions are loaded into the execution area and executed normally.

DBUG, therefore, provides the user with the ability to simulate execution of his program. With the various DBUG options, the user may view the effect of each instruction on the registers of the 8080, run his program with predefined register test values, set breakpoints, and display the status of his registers and DBUG itself.

Input/Output in DBUG

As mentioned above, the IN and OUT instructions are trapped by DBUG and executed artificially. DBUG supports three basic modes of I/O execution:

1. Normal mode -- The Iport (Input port) or Oport (Output port) number is displayed on the console and the user is prompted for input if it is an Iport or given the output in hexadecimal if it is an Oport. In the case of an Iport, the user may type a hexadecimal number or a single quote (') followed by a letter to enter an ASCII character.
2. Activated mode -- The Iport or Oport is

activated in real time, and the corresponding IN or OUT instruction is actually executed.

3. Predefined mode -- The predefined value of the Iport is loaded into the A register. This mode is only valid for Iports; see the 'I' DBUG command.

The DBUG Commands

The following is a detailed list of the commands available under DBUG when the '-' prompt is given. All commands are one-character commands terminated by a <CR>.

1. X -- exit; return to the calling program,
2. R -- toggle register display under C or T execution,
3. S -- status set/display; display and set the values of the registers, clear selected activated Iports and Oports, clear and redefine selected predefined Iports, and clear selected breakpoints,
4. I portnumber value -- set the specified Iport to always have the specified value; this Iport is now predefined,
5. A portnumber -- activate the specified port; the user is asked if this is an Iport or and Oport, and this command must be executed twice to activate one port as both an Iport and an Oport,
6. B address -- set a breakpoint at the specified address,
7. E address -- execute code starting at the specified address; load the registers with the current defined values; don't display the instructions executed or the register values,
8. T address -- trace the program starting at the specified address; load the registers with the current defined values before execution starts, and
9. C address -- trace the program starting at

the specified address; clear (zero) all registers before execution starts.

These are the basic DBUG commands. They take the form of MCS commands, and refer to "The Monitor Command System Users' Manual" for more information on the format of MCS commands.

MEMORY TEST USER'S MANUAL

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

17 July 1979

MEMORY TEST USER'S MANUAL

The ARIAN II Memory Test program is invoked by either typing TMEM, in which case the test program is loaded into the B000 to BFFF region of memory, or AMEM, in which case the test program is loaded into the A000 to AFFF region of memory. This program performs an exhaustive test over 4K blocks of memory, with each test lasting for about 2 minutes.

TMEM responds to four commands:

A -- Auto Test. Scan all of memory, identify the 4K blocks of RAM, and test these blocks in order from block 0 to block F hexadecimal. This test just indicates the number of errors (bad chips) found in each block. This assumes a 4 X 8 memory array.

M -- Branch to MCS.

T -- Test a specific 4K block of memory. The user is prompted to give a block number, of the form of a single hexadecimal digit. Unlike the Auto Test, this test displays the block of memory as a 4 X 8 array (assuming a 2102-type of memory chip). This display indicates with 'G' the chips which are good and 'X' the chips which are bad.

X -- Exit (return) to calling program.

All tests performed by TMEM and AMEM are destructive.

XEDIT USER'S MANUAL

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

17 July 1979

XEDIT USER'S MANUAL

The Extended Intra-Line Editor is a Level 3 Command invoked by engaging Command Level 3 and typing 'XEDIT'. In all cases, XEDIT requires a line number as an argument.

This editor employs the specialized cursor control capabilities of the Video Display Module. It creates a dynamic display of the line to be edited, with the screen divided into several distinct sections. These sections are: (1) the input line, which is altered dynamically by the Extended Intra-Line Editor, (2) the information section, which provides brief summaries of the commands, and (3) the line statistics section, which displays dynamically the information on the size of the line, the edit mode the user is in, the cursor position, and the number of characters in the line.

Its dynamic attributes are what make XEDIT different from the EDIT Level 2 command. As the user edits a line under XEDIT, the display of the line changes dynamically. As characters are inserted, the line spreads apart; as characters are deleted, the line compresses; as characters are replaced, the changes are obvious. The cursor moves from one end of the line to the other under the user's control, and the statistics of the line are updated dynamically as the editing is done.

XEDIT responds to a number of editing commands, which take the form of single characters typed by the user. These commands are:

A -- Abort the edit and return to ARIAN II.

B -- Place the cursor at the beginning of the line.

D -- Delete the character currently pointed to by the cursor. As this deletion occurs, the line is compressed.

E -- Place the cursor at the end of the line.

I -- Insert characters into the line before the

character pointed to by the cursor. As this insertion occurs, the line expands. This mode is called the Insertion Mode, and it is exited by typing either <ESC> or <CR>.

R -- Replace the characters pointed to by the cursor with the characters typed. The cursor advances as each succeeding character is typed. This mode is called the Replacement Mode, and it is exited by typing either <ESC> or <CR>.

S -- Skip to the specified character. This is a two-character command which consists of the letter 'S' followed by the letter to skip to. The cursor advances until it encounters the specified character or the end of the line. If found, the cursor points to this character.

X -- Exit (abort) and reedit the original line.

<BS> -- Back up the cursor to the previous character.

<CR> -- Terminate the edit, placing a <CR> at the current cursor position. The edit buffer is then copied into the primary file.

<SP> -- Advance the cursor to the next character.

<TAB> -- Advance the cursor 8 character positions.

Both upper and lower case characters are recognized for all commands.

XEDIT operates in four basic modes, which are indicated in the dynamic display. These modes and their subcommands are:

1. Normal -- XEDIT is ready to receive a command. All the commands above are effective.

2. Insertion -- This is the insertion function, in which characters are inserted before the character pointed to by the cursor. This mode responds to four subcommands: (1) <BS> -- back up the cursor to the previous character, (2) <CR> -- function as <CR> above (terminate edit), (3) <ESC> -- terminate the mode, and (4) <TAB> -- advance the cursor to the next character.

3. Replacement -- This is the replacement

function, in which characters pointed to by the cursor are replaced by those typed. This mode responds to the same subcommands as the Insertion Mode.

4. Skip -- This is the skip function, invoked by the 'S' command. The mode message is displayed to indicate that XEDIT is waiting for the user to type the character to skip to.

The ARIAN II USER'S MANUAL

by

Richard L. Conn

USA Satellite Communications Agency

Fort Monmouth, New Jersey 07703

8 July 1979

Table of Contents

Title	Page
Chapter 1: INTRODUCTION	1
Chapter 2: The ARIAN II EXECUTIVE	4
1 The Executive Reset	5
2 The ARIAN II Prompt	5
3 The ARIAN II Input Line Editor	6
4 The ARIAN II Command	8
5 The ARIAN II Command Levels	10
Chapter 3: The ARIAN II FILE SYSTEMS	12
1 Local Text Files	13
2 Local Binary Files	15
3 Local File Manipulation	16
4 Disk Files	17
Chapter 4: LEVEL 1 and LEVEL 3 COMMANDS	20
1 Command Level 1	20
2 Command Level 3	21
3 Interfacing Level 1 and Level 3 Commands to ARIAN II	22
4 ARIAN II Entry Points	24
5 Use of Restart Locations by ARIAN II	25
6 The ARIAN II Jump Table	25
Chapter 5: The ARIAN II ASSEMBLER	27
1 The Assembler in General	28
2 Assembler Pseudo-ops	31
3 System Reserved Labels	32
4 The Standard 8080 Mnemonics	34
5 The Special Z80 Mnemonics	35
6 Operand Evaluation	37
7 Assembler Error Messages	37

Table of Contents, Continued

Chapter 6: The ARIAN II SUBSYSTEMS --
TERMINAL and UTILITY 39
1 The Terminal Subsystem 39
2 The Utility Subsystem 41

Chapter 7: SUMARY of the ARIAN II COMMANDS 42
1 System Control 43
2 Primary File Editing 43
3 Local File Control 44
4 Disk File Control 44
5 Local/Disk File Transfer 44
6 List/Print 44
7 Program Debugging 44
8 Assembler 45
9 Utility 45

References 46

Appendices
I The ARIAN II EXECUTIVE Commands
II The ARIAN II BUFFERS and JUMP TABLE
III SUMMARY of the ARIAN II ASSEMBLER
IV The ARIAN II LEVEL 3 SYSTEM COMMANDS

CHAPTER 1

INTRODUCTION

ARIAN II is an operating system, a program which allows its user to execute and control other programs, designed specifically for a microcomputer possessing the ARIES-I hardware configuration. Based on the original ARIAN, or ARIAN I, ARIAN II is extensively integrated with the ARIES-I hardware configuration; the microcomputer hardware and the ARIAN software are designed to work together with each other's configuration in mind. This type of design -- that of an integration of hardware, firmware, and software within a single system -- produces a unique and useful tool.

This tool is designed to be used for software development. The hardware configuration was organized and coordinated with ARIAN to give the user the maximum amount of benefit from his resources. This hardware configuration consists of the following:

1. at least 40K bytes of RAM,
2. the Z80 microprocessor,
3. two 5 1/4-inch floppy disk drives (Shugart SA-400) with North Star floppy disk controller,
4. a PROM programmer,
5. a high-speed (9600 baud) CRT and keyboard, a printer, a modem, and a second high-speed CRT VDM, with their associated serial I/O ports, and
6. three parallel I/O ports, including a front panel LED display.

As the user can see, the ARIES-I hardware configuration is designed specifically with software development in mind.

ARIAN II is specifically designed to aid in software development for the Z80 microprocessor. Its 34 commands give the user the abilities to create and manipulate text and binary files, to assemble the text of an assembly language file, to execute and breakpoint his assembly language programs with a number of debugging aids, to communicate with an external computer via the modem, and to examine and modify the microcomputer's memory directly.

While ARIAN II is running, the user may be in any one of a number of command or data entry modes. These mode are:

1. Block Line Entry Mode. This mode permits the entry of a block of lines into the primary file. It allows the user to enter lines into the primary file without typing the line numbers; ARIAN II automatically prefixes each line with a line number and optionally renumbers the primary file when the user exits this mode.
2. Command Mode. Command mode permits the user to type a command to ARIAN II.
3. Display Mode. Display mode is the mode in which ARIAN II is displaying a directory or lines of text to the user. He can stop this display while it is being created by keying <ESC>. All displays are paged, and the user is prompted with a "?" at the bottom of each page to find out if he wishes to continue the display; the user must respond with "N" or "n" to stop the display and anything else to continue.
4. Edit Mode. This mode is the command system invoked by the EDIT Level 2 command or XEDIT Level 3 command.
5. Terminal Mode. Terminal mode is an interactive subsystem of ARIAN II. The user can communicate with an external computer via the modem and acoustic coupler while in this mode. It is invoked by

the TERM command.

6. Utility Mode. This is the Utility Subsystem, invoked by the UTIL command.

This is a user's manual for ARIAN II. ARIAN II is different from ARIAN I in many respects, and the ARIAN I user is encouraged to read this manual and its appendices before using ARIAN II. It is hoped that the user will find ARIAN II to be at least as useful and as far above its predecessor, ARIAN I, as ARIAN I was above its predecessors. IDIC

CHAPTER 2

The ARIAN II EXECUTIVE

The heart of ARIAN II is a very small program called the Executive. This program performs the following functions:

1. Resets the system stack, sets interrupt mode zero, and disables interrupts,
2. Displays the ARIAN II prompt to the user via the principal I/O channel,
3. Inputs a command from the user through the input line editor,
4. Parses the input command line, and
5. Searches for and transfers control to the command routine if found.

The Executive was designed to take advantage of the modularity of ARIAN II. The five functions of the Executive exist as small linear blocks of code within the Executive itself or as subroutines within ARIAN II. Additionally, the command routines themselves exist as subroutines, either memory-resident or on disk. Hence, the Executive is very small and consists mainly of subroutine calls.

The body of this chapter describes these five sections of the ARIAN II Executive is detail.

1 The Executive Reset

The purpose of the system reset in the Executive is to stabilize the environment of ARIAN II. ARIAN II does not use interrupts, so the setting of interrupt mode zero and the disabling of interrupts ensures that the ARIAN II Executive will not be interrupted by a maskable interrupt. ARIAN II contains a trap for non-maskable interrupts which returns control to the Executive.

ARIAN II typically uses two stacks -- one for the ARIAN II system and one for the user. In this way, the user can employ the program debugging facilities by executing a section of his code, returning control to ARIAN II and examining registers, etc., and then continuing the execution of his code with a minimum of difficulty. Hence, the Executive's function of resetting the ARIAN II system stack ensures that the user's stack will not be altered by the execution of ARIAN II routines.

Secondly, abnormal termination of an ARIAN II routine, such as in the case of an error, may result in an unbalanced system stack. The feature of the Executive of resetting the system stack on each pass through the Executive ensures that the stack is balanced.

2 The ARIAN II Prompt

The prompt of ARIAN II serves two purposes: (1) it informs the user of the status of the disks and (2) it tells the user that ARIAN II is ready to receive a command.

The prompt consists of two digits followed by a greater-than sign ('>'). ARIAN II is designed to be used as a multiple-drive system, and two drives are of particular interest to ARIAN II. One is the Command Drive, and the other is the Logged-in Drive. The Command Drive holds a disk which contains the transient routines executed by ARIAN II. These are called the Level 3 commands, and this feature of transient routines is referred to as Command Level 3. If Command Level 3 is engaged and the ARIAN II Executive does not find the name of the current command in its local command directories, it will load the directory of the Command Drive and search for

a Level 3 command of the specified name. If an entry is found, it will be loaded and executed. Level 3, as well as Level 1 and Level 2, commands will be discussed later in this chapter.

The two digits in the prompt give the numbers of the Command and Logged-in Drives, in that order. ARIAN II supports up to four drives, and the first digit, which gives the number of the Command Drive, may take on values from 0 to 4, while the second digit, which gives the number of the Logged-in Drive, may take on values from 1 to 4. If the number of the Command Drive is 0, then Command Level 3 is turned off; otherwise, the given number is the number of the drive from which Level 3 commands will be loaded.

Examples of the ARIAN II prompt are:

```
01>  -- Command Level 3 is off, 1 is the number of the Logged-in Drive

22>  -- 2 is the number of both the Command Drive and the Logged-in Drive

41>  -- 4 is the Command Drive and 1 is the Logged-in Drive
```

3 The ARIAN II Input Line Editor

All typing done while the user is in ARIAN II with the exception of the immediate prompts is processed by the ARIAN II Input Line Editor. This editor collects each character as the user types it, stores it in a buffer, and allows the user to correct any typing errors he has made. When the user terminates the line by typing a carriage return, this buffer is terminated and ARIAN II processes the contents of the buffer. This routine is used while ARIAN II is in Command Mode and Block Line Entry Mode, and it may be called explicitly by the user (see the chapter on the ARIAN II assembler).

As each character is typed, it is checked to see if it is an editor control character. If it is, the function of the control character is executed; if not, the character is saved in the input line buffer and echoed to the principal I/O device. When the user has finished typing the line, he terminates it with a carriage return. No character is echoed as a

result of this, and control is returned to the calling program.
The following is a list of all the editor control characters:

1. the Escape (<ESC>) key. When typed, <ESC> is printed on the principal I/O device as a dollar sign ('$') followed by a <CR>. This key tells the editor to delete the line typed so far and start over with a new line.

2. the Line Feed (<LF>) key. This key echoes as a <CR> and does not affect the line contained in the input line buffer. The sole purpose of this function is to allow the user to continue typing his line on the next physical line of the principal I/O device. No character is entered into the buffer as a result of this key, so the user must ensure that he does not run characters together.

3. the Tab (<TAB>) or Ctrl-I key. This key causes the cursor to tab to the next tab stop. As the cursor is tabbing, spaces are copied into the input line buffer. The tab stops are set by the TABS command, which is discussed in detail in Appendix I.

4. the Backspace (<BS>) or Ctrl-H key. This key allows the user the delete the last character he typed. It echoes as the cursor backing up to the previous position, and it is designed to be used with output devices which perform this function when they receive the ASCII code for a backspace. For example, if the user typed "ABCD<BS>", only "ABC" is in the input line buffer; the "D" has been deleted. If <BS> is typed again, the "C" is deleted, and so on. The user cannot delete beyond the beginning of the line; if he attempts to do this, an <ESC> is processed, echoing as a "$" <CR>.

5. the Delete (<DEL>) or Rubout key. This key performs the same function that Backspace does, but it echoes differently. The deleted characters are enclosed in backslashes. For instance, if the user typed "ABCD<DEL>E", this would be echoed onto the principal I/O device as "ABCD\D\E", indicating that the "D" was deleted and the string in the input line buffer is "ABCE". If the user types more than one <DEL> in a row, all the deleted characters are enclosed in one set of backslashes. For

example, if the user types "ABCDE<DEL><DEL><DEL>ABE<DEL>C", this will appear on his terminal as "ABCDE\EDC\ABE\E\C", indicating that "EDC" and then "E" were deleted and the resulting string is "ABABC". This feature is provided to permit ease in the use of an I/O device that does not have a hardware backspace feature.

6. the Carriage Return (<CR>) key. The <CR> key always instructs the input line editor to terminate the input of the line and give the line to the calling program (ARIAN II) to interpret.

The input line editor is an extremely useful tool, and with practice it will soon become a very easy and natural tool to use.

4 The ARIAN II Command

The fourth section of the ARIAN II Executive is the command line parser. This subroutine breaks apart the command line into its various components and stores each element of the command in its appropriate internal buffers.

The ARIAN II Command is divided into five basic divisions:

1. the name of the command,
2. up to three "special" characters following the name of the command,
3. the name of a file or symbol,
4. one or two strings, and
5. up to three decimal or hexadecimal numeric arguments.

Only the first element is required, and any number of the rest of the elements are optional depending upon the nature of the command to be executed. These elements, however, must occur in the order specified; that is, the command name must be first, the special characters next, the file name next, etc. Fields 2, 3, 4, and 5 are separated by one or more delimiters, and if more one numeric argument is specified, these must also be separated by delimiters. If two strings are specified, they may

optionally be separated by delimiters. The delimiter characters in ARIAN II and spaces and commas; these may be used as one prefers to improve readability.

The name of the command consists of from one to four alphanumeric characters, the first of which must be alphabetic. There is one special case in which the command is a line number, and, in this case, the other fields don't exist and the text which follows the line number is a space followed by the text of the line (see Appendix I).

The special characters are options pertaining to the individual commands. If special characters are used, the command name must consist of exactly four characters. These special characters then appear as the 5th, 6th, and 7th characters of a string. The ARIAN II commands look for these characters specifically, and if the special characters typed do not match those recognized by the command they are simply ignored.

The file name is a string of from one to eight alphanumeric characters, the first of which must be alphabetic. It is separated from the command name and its special characters by one or more delimiters.

The strings are vectors of characters enclosed in double quotes ("). If two strings are specified they need not be separated by any delimiters; the double quotes which close the first string and open the second serve this purpose. An escape character is provided by the ARIAN II Command Parser to permit the user to specify a double quote as part of a string; this is the backslash (\). The backslash instructs the parser to interpret the following character literally, so \" translates into one double quote and \\ translates into one backslash. If there are not more characters following the last string in the command line, it need not be terminated by a double quote; the <CR> which terminates the line is sufficient.

The numeric arguments consist of from one to five decimal and hexadecimal digits, the first of which must be decimal. The arguments are separated by one or more delimiters.

To facilitate ease of use in text processing applications, the alphabetic characters in the command name, the special characters, the file name, and the numeric arguments are converted internally to upper case by the ARIAN II Command Parser. Therefore, the user may enter his commands in either upper- or lower-case, and they will be interpreted in upper-case. Of course, no translation is done on the contents of the strings. See Appendix I for a detailed description of the ARIAN II Command in SDL.

A command line beginning with an asterisk, semicolon, blank, or any ASCII character less in value than the character for zero is considered to be a comment and is not processed by ARIAN II. Upon encountering such a character, the Executive loops back upon itself and prepares to receive the next command line.

5 The ARIAN II Command Levels

After receipt of a command from the parser, ARIAN II then searches through up to three directories of commands for the specified command. These are called Command Levels, and are named Command Level 1, Command Level 2, and Command Level 3, in the order they are searched. The specified command is searched for at Command Level 1 first and Command Level 3 last.

Command Level 2 contains the ARIAN II Executive, or resident, commands. They are discussed in detail in Appendix I. These commands provide the basic control, file editing and manipulation, utility, and debug facilities of ARIAN II.

Command Level 1 is the Customized, or user-defined, command level of ARIAN II. These are created by the CUST Level 2 command, and it allows the user to specify the name of a command and its execution address in memory. This is a particularly useful feature, and the user can employ this to quickly and with a minimum of effort execute test programs or override any Level 2 command. The command itself can take the form of a simple subroutine, and is therefore very easy to create. Refer to Appendix I and the chapter on Level 1 and Level 3 commands for more information.

Command Level 3 is the disk-resident command level. Again taking the form of simple subroutines, these commands reside as files on disk and are loaded and executed by ARIAN II.

When ARIAN II receives a command, it first searches through the Level 1 directory. If it is found, it is executed and control is returned to ARIAN II. If it is not found, ARIAN II then searches through the Level 2 directory. Again, if it is found, it is executed and control is returned to ARIAN II.

If the command is not found at Level 2, ARIAN II checks to see if Level 3 is engaged. If not, an error message is given and control is

returned to ARIAN II. If so, ARIAN II loads the directory of the Command Drive and searches it for the command file. If found, this file is loaded at its execution address, executed, and control is returned to ARIAN II. If it is not found, an appropriate error message is given and control is returned to ARIAN II.

Level 3 files are binary files. Their directory entries contain the name of the command with a ".CMD" extension, like "NAME.CMD", and an execution address. Refer to the chapter on Level 1 and Level 3 commands for more information.

CHAPTER 3

The ARIAN II FILE SYSTEMS

ARIAN II supports three types of file systems -- local text files, local binary files, and disk files (both text and binary). This chapter describes the techniques for creating and manipulating files in these three systems and the characteristics of files in these three systems.

Local files are memory-resident, while disk files reside on disk. ARIAN II maintains two local file directories which are used to assist the memory manager in its function of monitoring and controlling the use of memory in the ARIAN II environment and the user in identifying his resident files. Each disk also contains a directory of its files, and this directory is loaded into memory and analyzed whenever a file is transferred to or from disk.

The ARIAN II user may create up to ten local text files and ten local binary files. Each disk is capable of holding up to 64 disk files. ARIAN II partitions memory for various functions, and the ARIAN II workspace is the partition of memory within which the local text files reside. Binary files may exist anywhere within the address space, as may Level 3 command files. Most Level 3 command files, however, load and execute in the transient program area (refer to the chapter on Level 1 and Level 3 commands).

1 Local Text Files

The local text files are structured organizations of the user's text which reside in memory. They consist of lines of text and are terminated by an end-of-file mark. Each line consists of a four-digit line number, a space, and the text of the line. Transparent to the user, each line also contains some diagnostic information which helps ARIAN II determine as to the validity of the file.

One of the local text files is designated as the primary file, while the others are referred to as secondary files. The primary file is the file which is currently being referenced by the user. The file and line editing commands, as well as the disk transfer commands, operate on the primary file. The secondary files reside in memory for the purpose of later being included in the primary file or being designated as the primary file.

When a local file is created, it is automatically made primary. ARIAN II can create local text files in a number of ways, but the most common way to initially create a local text file is by using the FILE command. "FILE FILENAME" will create a local text file starting at a memory location selected by the memory manager and make it primary. Once created, the common way of entering information into this file is by using the "APND" command. APND appends the following block of lines, entered in Block Line Entry Mode, to the end of the current primary file, or, if a line number is specified, after the given line. The Block Line Entry Mode is terminated by the escape sequence consisting of a Ctrl-C followed by a <CR>. Upon exiting this mode, ARIAN II automatically renumbers the file starting at 5 and incrementing by 5 unless the N option is used ("APNDN"), in which case each line in the appended block begins with the specified line number or 9999 if the no line number was given.

The primary file editing commands, namely <lnum>, APND, DEL, EDIT, FIND, ISRT, and RNUM, make up the resident file editor of ARIAN II. ISRT, or insert, is the same as APND, but it inserts the block of lines before the specified line. DEL is used to delete one line or a block of lines, <lnum> is used to enter a line directly by typing a 1-4 digit line number followed by a space and the text of the line, EDIT is the intra-line

editor, FIND is used to search for a specified string in the primary file, and RNUM renumbers the primary file.

All of these commands make the resident primary file editing facility of ARIAN II very responsive to the user. He may intermix these commands with the other system commands at his discretion.

For example, the following illustrates the use of some of these primary file editing commands. This is not a precise example; all ARIAN responses are in lower case and user entries are in upper case. Refer to the sample session for more detailed and precise examples.

```
01>FILE TEST
test     2a00  2a00
01>APND
?THE RAIN IN SPAIN
?FALLS MAINLY
?ON SUNDAY THROUGH MONDAY
?ON ODD WEEKDAYS
?IN THE PLAIN.
?^C
01>LIST
0005 the rain in spain
0010 falls mainly
0015 on sunday through monday
0020 on odd weekdays
0025 in the plain.
01>DEL 15,20
01>LIST
0005 the rain in spain
0010 falls mainly
0025 in the plain.
01>RNUM
01>LIST
0005 the rain in spain
0010 falls mainly
0015 in the plain.
01>FIND "SP
0005 the rain in spain
```

```
01>LIST 15
0015 in the plain.
01>RNUM 10,10
01>LIST
0010 the rain in spain
0020 falls mainly
0030 in the plain.
01>25 WHEN IT WISHES
01>LIST
0010 the rain in spain
0020 falls mainly
0025 when it wishes
0030 in the plain.
01>
```

As the user can see, the resident primary file editor of ARIAN II is quite versatile. There are also a number of Level 3 commands which supplement this editor, but they will be discussed later. Also, not every aspect of these commands has been discussed. Each command has several forms with several options, and Appendix I discusses them at some length.

The EDIT command invokes an intra-line editor which can be used to change the contents of a line of the primary file without retyping the entire line. It contains subcommands which permit the user to delete, replace, and insert characters into the line. EDIT is discussed in much greater detail in Appendix I.

2 Local Binary Files

The local binary files are unstructured organizations of programs and data which reside in memory. They are defined solely by their entries in the local binary file directory, and this entry only provides information as to the name of the file and the range of memory over which it exists. The memory managers of ARIAN II totally ignore the entries in the binary file directory, and these files are subject to destruction by the memory managers.

For this reason, it is recommended that the user create all binary

files outside the local text file workspace and transient program areas of ARIAN II. Also, by realizing that the local text files in the ARIAN II workspace grow upward in memory, the user may judiciously create his binary files in the upper regions of the workspace. This, of course, must be done with care and consideration of his future actions in the system.

Local binary files are created in one of three ways: (1) implicitly when ASSM is used with a file name specification, (2) implicitly when a binary file is loaded from disk, and (3) explicitly by the user via the FILEB option of the FILE command. When created by the assembler, the binary file is defined to exist in the space of memory covered by the assembly from the last ORG or the entire assembly range if no ORG is specified within the assembled file. The LDIRB command lists all the binary files, and the first line of this directory listing gives this range as two hexadecimal numbers. When created by the LOAD command, the binary file has a range of an integral number of blocks. It starts at the load address of the file and extends over the size of the file as it resides on disk. Finally, when created explicitly, the binary file has exactly the range specified by the user.

No management is done of the binary files by ARIAN II. These files are referenced solely for the convenience of the user, and it is up to him to use them at his discretion.

3 Local File Manipulation

A number of Executive commands are available through ARIAN II which manipulate, either directly or indirectly, the local files and provide information to the user on their validity.

Two commands, FCHK and RCVR, apply only to local text files. FCHK is used to determine the validity of a local text file. Employing the inherent structure of the ARIAN II text file, FCHK insures that the specific data of each line of the specified or implied file is valid. FCHK with no argument validates the primary file, and FCHK FILENAME validates the specified local text file without affecting the status of the other files. The other command, RCVR, is used to attempt to recover a local text file that has been deleted from the local text file directory. It uses the inherent structure of the ARIAN II text files to attempt to determine the

bounds and other pertinent information on the data starting at the address specified by the command in an attempt to rebuild the directory entry. If the file is found to be intact up to its end-of-file mark, a directory entry is created. The format of this command is RCVR FILENAME ADDRESS, where FILENAME is the new name of the file to be recovered and ADDRESS is the starting address from which to attempt the recovery.

The rest of the commands are similar in structural format. They are FILE, LDEL, LDIR, LNAM, and LSCR. Each command as given operates on the local text files, while FILEB, LDELB, LDIRB, LNAMB, and LSCRB operate on the local binary files.

The FILE command is used to create local files, and, in the case of text files, it can also specify the bounds and name of the current primary file. LDEL deletes the specified local file, and LSCR scratches (deletes all the files in) the specified local directory. In the cases of LDEL and LSCR, only the directory entries are affected; note that RCVR may be used to restore the deleted text files. LDIR lists the contents of the specified local directory, and, finally, LNAM is used to rename the specified local file.

4 Disk Files

Disk files are files which reside on disk; they may be either text (type 0) or binary (type 1). All disks contain directories which define the files contained on them, and these directories provide the only direct means of definition of such files. The directory entries contain the name of the file, its size in 256-byte blocks, its type, its starting disk address, and, if it is a binary file, its execution/load address.

The files themselves are organized collections of data stored in sequential 256-byte blocks on disk. They contain no particular distinguishing features, except for text files, whose internal organization is exactly the same as local text files. All disk files are loaded and stored sequentially to and from memory.

For this purpose, ARIAN II supports the LOAD and SAVE commands. LOAD will load the specified disk file into memory and make the appropriate entry in either the local text or binary file directory. SAVE will save the current primary file on disk under the name specified, and SAVEB will

create and store a binary file on disk.

In addition to the LOAD and SAVE commands, ARIAN II supports three Executive commands which perform functions similar to LDEL, LDIR, and LNAM, but they operate on disk files. These commands are DDEL, DDIR, and DNAM. DDEL deletes the specified disk file's directory entry, DDIR displays the disk file directory, and DNAM renames the specified disk file.

As with LOAD and SAVE, DDEL, DDIR, and DNAM operate on the Logged-in drive. As mentioned earlier, the number of the Logged-in drive is specified as the second digit of the ARIAN II prompt. As with all ARIAN II Executive commands, refer to Appendix I for a detailed description of their function.

The following are examples of the use of some of the ARIAN II commands discussed in this chapter. Again, refer to Appendix I for more information on these commands. The upper/lower case conventions of user/ARIAN II apply as before.

```
01>DDIR
fdos     10 1 2000       sdl      44 0
arain2   60 0            arian3   50 0
01>LDIR
test    2a00  2a6b
01>LNAME TEST
new name? TEST1
01>FILE
test1   2a00  2a6b
01>LDIRB
b800 b85a
01>SAVE IT
$ file saved
01>DDIR
fdos     10 1 2000       sdl      44 0
arian2   60 0            arian3   50 0
it        1 0
01>DDEL IT
01>DDIR
fdos     10 1 2000       sdl      44 0
arian2   60 0            arian3   50 0
```

```
01>LSCR
01>LDIR
01>LDIRB
b800 b85a
01>
```

CHAPTER 4

LEVEL 1 and LEVEL 3 COMMANDS

Command Levels 1 and 3 are quite similar in many respects. The commands in these levels take the form of subroutines, i.e., they are usually bodies of code ending in a RET instruction. They are memory-resident during execution. Finally, they may take advantage of the various entry points into ARIAN II and use the assembler-defined symbols.

1 Command Level 1

Command Level 1 is the customized command level. These commands, which are created by the CUST Executive command, are memory-resident and are defined only by their entries in the Customized Command Table. These table entries consist of the name of the command (1 to 4 characters, the first of which must be alphabetic) and the execution address of the command.

Customized commands are created by the CUST <cname> <hadr>? variant of the CUSTomize command. The subroutine starting at the specified or implied address becomes the new command, and, until the user deletes this customized command, whenever he types the command name given, he will execute this subroutine. If no address is specified, the default execution address is used to define the starting address of the command.

Examples of the use of the CUST command are:

CUST LIST
The LIST Executive Command of ARIAN II is redefined, and the user will execute the subroutine starting at the default address whenever he types "LIST".

CUSTD LIST
The user-defined LIST command is deleted, and the normal system LIST command is restored.

CUST TEST 6C00
A new customized command called TEST is created; its execution starts at location 6C00 hexadecimal.

CUSTL
All the customized commands defined by the user that are currently in effect are listed with their execution addresses.

CUSTS
All customized commands are scratched (deleted).

2 Command Level 3

The Level 3, or disk-resident, commands take the same form as the Level 1 commands, but they are loaded from disk when their command name is given. They reside on disk as binary (type 1) files whose names consist of the four characters of the command name followed by ".CMD", like "HELP.CMD". The load address in their disk directory entries specifies the memory location at which they are loaded and executed.

Level 3 commands may execute anywhere in memory, but ARIAN II has reserved an area of memory for these commands. It is called the transient program area, and it is the 1K section of memory which starts at 1K above

the end of the ARIAN II workspace; it usually starts at BC00 hexadecimal. Most Level 3 system commands execute in the transient program area, but there are exceptions. It is recommended that the user assemble all his Level 3 commands to execute in this region.

Like Level 1 commands, Level 3 commands can be created by creating a file of the program and assembling it (by using ASSM 0BC00 -- see the chapter on the assembler). The assembler will give the assembly limits specified by the last ORG and the end of the program, and, if these are the actual limits of the program, the user can save it on disk as a Level 3 command by issuing the SAVEB command, like "SAVEB NAME.CMD 0BC00 0BDEF". This will save the binary of the command on the logged in drive. To execute the command, the user should ensure that this drive is made the command drive and then type the name of the command, "NAME". It will then be loaded at its load address (BC00 hexadecimal in this case) and executed.

3 Interfacing Level 1 and Level 3 Commands to ARIAN II

In order to interface Level 1 and Level 3 commands to ARIAN II, the user must understand how ARIAN II parses commands. Commands are parsed into several distinct subfields, as mentioned earlier, and the elements of these subfields are stored in specified buffers within the ARIAN II scratchpad RAM area. All commands start with a command name followed by up to three special characters and any number of arbitrary characters. The command name is always interpreted as being four characters long, with unspecified characters being spaces. It is converted to upper-case letters and placed in the four-byte buffer CBUF by the parser. The special characters are placed in the three one-byte buffers, SPCHR1, SPCHR2, and SPCHR3, which immediately follow CBUF.

The second command subfield is a file or command name. This field consists of up to eight characters, the first of which must be alphabetic. It, too, is internally capitalized by the parser. This field is stored in FBUF, left justified and blank-filled on the right. If no element appears in this field, FBUF will contain only binary zeroes (0 hexadecimal).

The third command subfield consists of the two strings. Each string may consist of up to 40 characters, and they are stored in SBUF1 and SBUF2. No upper-case conversion is done to string fields. If a string is stored

in one of the string buffers, the first character of the buffer will be a double quote and the string is terminated by a carriage return character (0D hexadecimal). The ARIAN II parser permits the first string to be over 40 characters, in which case it will run into the second string buffer. In this case, there must be no second string, or the first string will be truncated.

The fourth command subfield is the numeric argument field. This field consists of from one to three numbers. The arguments parsed in this field are placed in ABUF (four ASCII characters per argument) and BBUF (two bytes per value). ABUF contains the ASCII characters of the numbers in groups of four (i.e., ABUF to ABUF+3 contains the first number, ABUF+4 to ABUF+7 contains the second, and ABUF+8 to ABUF+11 contains the third); if the numbers contain more than four digits, only the first four are contained in these fields, and if they contain less than four digits, these fields are left-filled (normalized) with the character for zero. BBUF contains their values, assuming they are hexadecimal numbers, in groups of two (i.e., BBUF and BBUF+1 contain the first value in the INTEL-standard low-order/high-order format, BBUF+2 and BBUF+3 contain the second, and BBUF+4 and BBUF+5 contain the third). If there are fewer than three numeric arguments specified, the corresponding ABUF and BBUF fields will contains binary zeroes (0 hexadecimal).

Hence, with the arguments of a customized command parsed in this manner, the user can create customized commands which perform like the normal ARIAN II Executive Commands. ABUF is usually used to contain line numbers for reference, BBUF is usually used to contain values used to specify hexadecimal quantities, SBUF1 and SBUF2 contain the strings, FBUF is usually used to contain a file or command name, SPCHR2, SPCHR2, and SPCHR3 contain the special characters, and CBUF contains the command name.

The entire input line, as a consequence of the input line editor, is available to the user in IBUF. The first character of the line is at IBUF, and the line ends in a <CR>. Location IBUF-1 contains a count of the number of characters in the line, including the character at IBUF-1 and the ending <CR>.

The following are examples of the use of these buffers. Note that <null> refers to the binary zero (0 hexadecimal), and <null>s refer to the number of binary zeroes required to fill the specified field.

```
     REGS BC 2000
     CBUF = "REGS", SPCHR1=SPCHR2=SPCHR3=<null>, FBUF =
"BC" and 6 blanks, SBUF1=SBUF2=<null>s,
ABUF(0-3)="2000", ABUF(4-11)=<null>s, BBUF(0)=0,
BBUF(1)=20 Hexadecimal, BBUF(2-5)=<null>s

     xdirx test
     CBUF = "XDIR", SPCHR1="X", SPCHR2=SPCHR3=<null>,
FBUF = "TEST" and 4 blanks, and the rest of the buffers
contain <null>s.

     saveb1 test.cmd 0a000 0afe0
     CBUF = "SAVE", SPCHR1="B", SPCHR2="1",
SPCHR3=<null>, FBUF = "TEST.CMD", SBUF1=SBUF2=<null>s,
ABUF(0-3) = "0A00", ABUF(4-7) = "0AFE",
ABUF(8-11)=<null>s, BBUF(0) = 0 hex, BBUF(1) = A0 hex,
BBUF(2) = E0 hex, BBUF(3)= AF hex, and
BBUF(4-5)=<null>s
```

The addresses of CBUF, ABUF, BBUF, FBUF, SPCHR1, SPCHR2, SPCHR3, SBUF1, and SBUF2 are given in Appendix II.

4 ARIAN II Entry Points

Aside from using a simple RET instruction to return to ARIAN II from a Level 1 or Level 3 command, ARIAN II has three reentry locations which may be branched to for a clean return to the system.

These reentry points are at memory locations 0, 4, and 66 hexadecimal. Location 0 is the ARIAN II dead start entry point; the entire ARIAN II system is initialized if execution begins at this point. Location 4 is the ARIAN II warm start entry point; only part of the system is initialized if execution begins at this point. Finally, location 66 hex is the non-maskable interrupt entry point. It transfers control directly to the ARIAN II Executive; no initialization is done if the user enters here. Location 66H is also referenced by the assembler-defined symbol ZEOR.

5 Use of Restart Locations by ARIAN II

Restarts 2, 3, 4, 5, and 6 are unused by ARIAN II and are free for the user to employ at his discretion.

Restart 0 is used for the dead and warm restarts of ARIAN II, and it is recommended that it not be used for any other purpose. Restart 1 is the ARIAN II breakpoint reentry restart, and it may be employed by the user if he does not wish to set any breakpoints under the BREK command. Finally, Restart 7 is the ARIAN II memory trap, and it, too, may be employed by the user at his discretion. This memory trap is a safety feature to halt programs which attempt to run in non-existant memory.

6 The ARIAN II Jump Table

Appendix II also summarizes the functions supported by the jump table in ARIAN II. Briefly, this jump table provides easy access to the following routines and Executive commands:

1. CKA11 - check for the presence of at least one numeric argument. Return with zero set if no numeric arguments, clear if at least one.
2. CKA21 - check for the presence of at least two numeric arguments. Return with zero set if two arguments are not present, clear if at least two are present.
3. CKFN1 - check for the presence of an argument in FBUF. Again, zero set means no argument, clear means there is an argument.
4. DSCAN - scan logged in disk directory for the file name contained in FBUF. If found, return with zero set and HL pointing to the first byte of the directory entry; otherwise, return with zero clear.
5. FILE - execute the FILE Executive command. Arguments are passed in the appropriate buffers.

6. FIND - find the line whose number is contained in ABUF (normalized). HL point to the first byte of the line upon return. Line number found is first line greater than or equal to the number in ABUF.

7. LINE - execute the <lnum> Executive command. The line contained in IBUF is entered into the primary file.

8. LOAD - execute the LOAD executive command. Arguments are passed in the appropriate buffers.

9. SAVE - execute the SAVE executive command. Arguments are passed in the appropriate buffers.

10. MESS - print the character string pointed to by HL and ending in <CR> on the principal I/O device and return to the ARIAN II Executive. This routine is good for printing fatal error messages.

11. FSEA - search the local text file directory for the file whose name is in FBUF. A number of flags are set by this entry; see Appendix II. Zero flag is clear if file name is found, set if not found.

12. RNUM - execute the RNUM executive command. Arguments are passed in the appropriate buffers.

CHAPTER 5

The ARIAN II ASSEMBLER

The assembler of ARIAN II is a very powerful real-time assembler based on the INTEL-standard mnemonics for the 8080 microprocessor. The assembler, however, is not just an 8080 assembler; it is a pseudo-Z80 assembler which gives the user the ability to assemble some of the common Z80 instructions as well as all of the 8080 instructions.

The assembler features its own set of pseudo-ops (most of which are similar to the INTEL-standard pseudo-ops), all the 8080 mnemonics, some arithmetic operations in the operand field, literal character definitions in the operand field, a unique set of Z80 instructions, the ability to explicitly create binary files, and manipulation of the default execution address.

The standard features supported by the assembler include:

1. free-format source input,
2. symbolic addressing, including forward and relative symbolic references,
3. up to 384 six-character symbols,
4. reserved names for the registers which may be redefined,
5. a full set of pseudo-ops, including default execution address control, and
6. automatic generation of a symbol table which may be referenced later.

1 The Assembler in General

The assembler translates the lines contained in the primary file into object code residing in memory. The second character following the line number is the first source code character position. Therefore, the character immediately following the line number should be a space; the APND and ISRT commands place a space here automatically, and the user need only be concerned with this restriction if he enters his own lines using the <lnum> <text> command. Line numbers are not processed by the assembler; they are merely reproduced in the listing.

The assembler will assemble a source program file composed of statements, comments, and pseudo operations on each line. It does this in two passes. During Pass 1, the assembler allocates all storage necessary for the translated program and defines the values of all symbols used by creating a symbol table. The storage allocated for the object code will begin at the byte explicitly or implicitly specified by the ASSM command unless an ORG pseudo-op is present in the program. During Pass 2, all expressions, symbols, and ASCII constants are evaluated and placed in allocated memory in the appropriate locations. The listing, also produced during Pass 2, indicates exactly what data is in each location of memory.

Statements contain either symbolic ARIAN II assembly language instructions or pseudo-ops. These statements are divided into up to four fields: (1) a name field (optional), (2) an operation field, (3) an operand field (optional), and (4) a comment field (optional). If the first valid character in a statement is a semicolon or asterisk, the entire statement is processed as a comment.

The name field, if present, must begin in the first assembler character position in the input line; this is the second character after the line number. The symbol in the name field can contain as many characters as the user wishes, but only the first six characters are used in the symbol table to uniquely define the symbol. All symbols in this field must begin with an alphabetic character and may contain no special characters. Digits are allowed. The name field may or may not be terminated by a colon, at the user's discretion.

The operation field contains either an ARIAN II assembler operation mnemonic or a system pseudo-op. The ARIAN II assembler operation mnemonics and system pseudo-ops are described below. The operation field is separated from the name field by one or more blanks; if no name field is present, it begins in or after the third character position after the line number (at least two spaces in front of it).

The operand field contains parameters pertaining to the operation in the operation field. If two arguments are present, they must be separated by a comma. The operand field is separated from the operation field by at least one space.

The comment field is for explanatory remarks. It is reproduced in the listing without processing. Comment lines must start with either a semicolon or an asterisk; it is recommended that comments at the end of a statement also start with one of these characters, but this is not a restriction (comments after the operand or operation field may or may not start with a semicolon or asterisk).

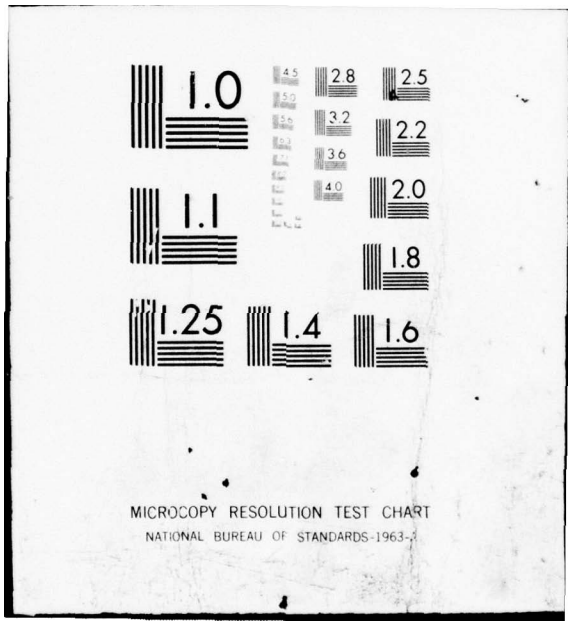
The ARIAN II assembler is completely free-format, as described above, but a "standard" ARIAN II assembler program line format is defined by ARIAN II and is recognized by various options of several commands. This format is as follows:

1. the name field must start in the first assembler line column,
2. if the name field exists, the operation field starts one space after the colon or end of the name field; if the name field does not exist, the operation field starts in the second assembler line column,
3. the operand field starts one space after the operation field,
4. the comment field starts one space after the operand field if there is one or one space after the operation field if there is no operand field, and the comment field must start with a semicolon, and
5. if the entire line is a comment, it must start with either an asterisk or a semicolon (asterisk preferred) in the first assembler line column.

AD-A072 471 ARMY SATELLITE COMMUNICATIONS AGENCY FORT MONMOUTH NJ F/G 9/2
PROJECT ARIES. USER'S MANUALS FOR ARIAN II AND ASSOCIATED SUBSY--ETC(U)
JUL 79 R L CONN

UNCLASSIFIED NL

2 OF 3
AD A072471


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Symbolic names and addressing are also supported by the assembler. To assign a symbolic name to a statement, the name is placed in the name field. To leave off the name field, the user skips two or more spaces after the line number (one or more spaces in block line entry mode) and begins the operation field. If a name is attached to a statement, the assembler assigns it the value of the current location (program) counter. The program counter holds the address of the next byte to be assembled if the instruction is a machine instruction or pseudo-op. The EQU pseudo-op, however, assigns to its label a value which is defined in the operand field. Note: do not confuse the location counter of the assembler with the "$" symbol discussed later; this location counter points to the next instruction to be assembled, while "$" points to the instruction after the current instruction if the current instruction is a normal mnemonic or "$" points to the current instruction if it is a pseudo-op.

Names are defined when they appear in the name, or label, field. All defined names may be used as symbolic arguments in the operand field. The reserved system symbols, however, are defined by the assembler and must not be redefined by the user; a duplicate label error will result if this is done. These reserved system symbols are discussed later.

In addition to the user-defined and system-defined symbols, the assembler has reserved several symbols to represent the registers of the 8080. These symbols, like the system reserved symbols, may only be used in the operand field. These symbols are:

1. A -- the accumulator; value 7,
2. B, C, D, E, H, and L -- the B, C, D, E, H, and L registers; values 0, 1, 2, 3, 4, and 5, resp.,
3. M -- memory (pointed to by H&L); value 6,
4. P, PSW -- the program status word; value 6, and
5. S, SP -- the stack pointer; value 6.

The assembler also supports relative symbolic addressing. If the name of a particular location is known, a nearby location may be specified using the known name and a numeric offset. All defined symbols, including "$", may be used in this relative symbolic addressing scheme.

For example, LDA $+5 loads the accumulator with the value of the byte

located five bytes after the beginning of the next instruction. Also, SSPD LOC-7 stores the value of the stack pointer starting at the byte located seven bytes in front of the memory location pointed to by the symbol "LOC".

The assembler permits the user to write positive and negative numbers directly in a statement. They will be regarded as integer constants, and their binary values will be used appropriately. All unsigned numbers are considered to be positive. Decimal constants can be defined using the suffix "D" after the numeric value, but this is not required since the default is decimal. Hence, 10 and 10D define the constant ten decimal. Hexadecimal constants must start with a digit and end with the suffix "H". Examples of hexadecimal constants are 10H, 0AFH, 000101H, and 00BCH.

ASCII constants may be defined by enclosing the ASCII character within single quotes, i.e., 'C'. Two characters may be enclosed within single quotes for double-word constants.

2 Assembler Pseudo-ops

The following is a list and a description of the pseudo-ops recognized by the assembler:

1. ASC '<string>' -- ASCII string. This pseudo-op loads consecutive memory locations starting at the current value of the location counter with the ASCII values of the characters specified in the string.
2. DB <expression> -- define one byte. This instruction evaluates the specified operand and loads one 8-bit value into the location pointed to by the location counter. If the number is evaluated into a 16-bit quantity, only the low-order byte is loaded.
3. DS <expression> -- define storage. This reserves the specified number of bytes starting at the current value of the location counter.
4. DW <expression> -- define one word. This instruction evaluates the specified operand, producing a 16-bit value which it loads into memory (low order, high order) at the location pointed to by the location counter

and the succeeding location.

5. END -- end the assembly. This statement is not required; assembly will stop when the end of the file is reached or this statement is encountered.

6. <label> EQU <expression> -- the specified label is assigned the computed value of the operand; the computed value is a 16-bit quantity.

7. EXEC <expression> -- the default execution address is set to the value of the expression.

8. LST -- turn on the listing of the assembly of the program on the principal I/O device if it is not already on. This can be used to list only selected portions of a program during the assembly.

9. NLST -- turn off the listing of the assembly of the program.

10. ORG <expression> -- set the origin (location counter) to the specified value. This instruction also resets the assembly limits and the location in memory at which the object code is loaded. If an ORG appears more than one time in the program, the limits set by the last ORG and the end of the assembly are reflected in the assembly limits displayed by the LDIRB command.

All pseudo-ops may be preceeded by a label.

3 System Reserved Labels

Another feature of the ARIAN II assembler is its system reserved labels. These symbols provide easy access to a host of utility subroutines for functions such as I/O, data conversion, and ARIAN entry points, and they also supply some commonly-used buffer and variable addresses. All system reserved symbols start with the letter "Z" and are at most four characters long.

The following is a complete list of the ARIAN II assembler reserved symbols. They are described in detail in Appendix III.

Symbol	Function
ZEOR	the Executive reentry point to ARIAN II
ZLIN	the ARIAN II input line editor
ZINK	polls the principal I/O channel for an <ESC>
ZIN	input one character from principal I/O channel
ZOUT	output one character to principal I/O channel
ZCR	output <CR> <LF> to principal I/O channel
ZARG	the ARIAN II Executive Parser
ZHOT	display the A register as two hexadecimal digits
ZDOT	display the A register as up to three decimal digits, left blank fill
ZBLK	output space to principal I/O channel
ZPRH	print string pointed to by HL ending in <CR> on principal I/O channel
ZPPH	print string pointed to by HL ending in <CR> on printer
ZPRR	print string pointed to by return address ending in <null> (0)
ZPHL	print HL as four hexadecimal digits
ZSHD	compute BC = HL - DE
ZBOF	address of two-byte buffer which contains starting location of primary file
ZEOF	address of two-byte buffer which contains ending location of primary file
ZBBF	address of BBUF
ZIBF	address of IBUF
ZEN	exchange nybbles of the A register
ZCHA	convert low nybble of A to its ASCII hexadecimal equivalent
ZCAH	convert ASCII hexadecimal character in A to its binary equivalent in A
ZCRL	call relative long
ZJRL	jump relative long

4 The Standard 8080 Mnemonics

The ARIAN II assembler recognizes all the standard 8080 mnemonics. For reference, they are:

Mnemonic	Mnemonic	Mnemonic	Mnemonic
ACI	ADC	ADD	ADI
ANA	ANI	CALL	CC
CM	CMA	CMP	CNC
CNZ	CP	CPE	CPI
CPO	CZ	DAA	DAD
DCR	DCX	DI	EI
HLT	IN	INR	INX
JC	JM	JMP	JNC
JNZ	JP	JPE	JPO
JZ	LDA	LDAX	LHLD
LXI	MVI	MOV	NOP
ORA	ORI	OUT	PCHL
POP	PUSH	RAL	RAR
RC	RET	RLC	RM
RNC	RNZ	RP	RPE
RPO	RRC	RST	RZ
SBB	SBI	SHLD	SPHL
STA	STAX	STC	SUB
SUI	XCHG	XRA	XRI
XTHL			

5 The Special Z80 Mnemonics

The following is a list of the special Z80 mnemonics recognized by the ARIAN II assembler and their ZILOG equivalents.

Mnemonic & Operand	ZILOG Equiv	Comments
SSPD <expression>	LD (nn),SP	store SP direct
LSPD <expression>	LD SP,(nn)	load SP direct
SBCD <expression>	LD (nn),BC	store BC direct
LBCD <expression>	LD BC,(nn)	load BC direct
SDED <expression>	LD (nn),DE	store DE direct
LDED <expression>	LD DE,(nn)	load DE direct
EXA	EX AF,AF'	
EXX	EXX	
BR <expression>	JR n	branch relative
BC <expression>	JR C,n	branch relative on carry
BNC <expression>	JR NC,n	branch relative on no carry
BZ <expression>	JR Z,n	branch relative on zero
BNZ <expression>	JR NZ,n	branch relative on no zero
DBJ <expression>	DJNZ n	decrement B and branch relative
LD	LDD	
LDR	LDDR	
LI	LDI	
LIR	LDIR	
CD	CPD	
CDR	CPDR	
CI	CPI	
CIR	CPIR	

```
NEG             NEG

RLD             RLD
RRD             RRD

SHB             SBC HL,BC
SHD             SBC HL,DE
SHS             SBC HL,SP

AHB             ADC HL,BC
AHD             ADC HL,DE
AHS             ADC HL,SP

IM0             IM 0
IM1             IM 1
IM2             IM 2

ID              IND
IDR             INDR
II              INI
IIR             INIR

OD              OUTD
ODR             OTDR
OI              OUTI
OIR             OTIR

CIN             IN A,(C)
COT             OUT (C),A
```

6 Operand Evaluation

Operand evaluation is somewhat limited in ARIAN II, due largely to the size restrictions on the system. Parenthesized expressions are not permitted. Only sixteen-bit addition and subtraction are permitted in infix expressions. Single character strings of the form '<char>' are permitted in expressions and stand-alone.

All numeric arguments are assumed to be decimal unless the suffix "H" is appended to them. Therefore, 100 is 100 decimal and 100H is 100 hexadecimal.

The "$" symbol is used as the value of the location counter for the next instruction. In normal instrucions, "$" points to the first byte of the next instruction; in pseudo-ops, "$" points to the first byte of the pseudo-op. This permits relative addressing to take the form of "BR LABEL-$" and pseudo-ops like "STACK EQU $" to be used.

Finally, if an expression with a value greater than 0FF hexadecimal is loaded into an eight-bit register, like "MVI A,1FFH", only the low-order byte of this value is loaded.

Examples of permitted expressions include:

```
LABEL+3
POINT-'A'+60
POINT3-0AFH+6-2
HERE-$-2
```

7 Assembler Error Messages

The following is a list of the error messages produced by the assembler and their meanings:

Error	Meaning
R	register error. The register name is missing or invalid.
S	syntax error. The instruction syntax is incorrect.
U	undefined symbol. The referenced symbol is incorrect.
V	value error. The computed value cannot be represented as a 16-bit value or the instruction has a syntax error.
M	missing label error. A required label is missing.
A	argument error. The instruction's argument is of the wrong type or generally incorrect.
L	label error. The label of this instruction contains an invalid character.
D	duplicate label error. The label of this instruction has been defined elsewhere.
O	opcode error. The opcode (in the operation field) of this instruction is invalid.

CHAPTER 6

The ARIAN II SUBSYSTEMS -- TERMINAL and UTILITY

ARIAN II supports two subsystems of commands at the Executive level; they are the Terminal subsystem, invoked by the TERM command, and the Utility subsystem, invoked by the UTIL command. Each is a complete subsystem; they contain their own argument parsing schemes and set of commands.

1 The Terminal Subsystem

The TERMinal command invokes the Terminal, or inter-system communication, subsystem of ARIAN II. Under this subsystem the microcomputer becomes relatively transparent to the user, and the user's terminal is made to respond like a timesharing terminal to an external computer system. Each character typed is sent to a modem and acoustic coupler, and each character received by the modem is sent to the user's CRT.

The Terminal subsystem responds to three commands. They are Ctrl-A, Ctrl-L, and Ctrl-R. Ctrl-A invokes the terminal alternate command set. This command set consists of the following commands:

1. M -- transfer to the Monitor Command System (MCS).
2. R <hadr> -- call the subroutine located at the

specified address. The return in the subroutine transfers control to the terminal mode (not the terminal alternate command set).

3. T <hadr> -- transfer the downloaded code to the specified address. This command transmits a <CR> to the external computer, and the object code downloaded is loaded into memory starting at the specified address. The addresses given in the code are ignored, and the entire code is loaded sequentially into memory.

4. X -- exit to terminal mode.

All commands in the terminal alternate command set are parsed like MCS (Monitor Command System) commands. The parser scans the input line until it encounters a non-blank character; this character is interpreted as the command name. The parser then scans until it encounters another non-blank character; it then interprets the string starting at this character as a hexadecimal number and scans until it encounters an invalid hexadecimal character. Only the last four characters are used to interpret the final value. If fewer than four characters are in the number then the number is zero filled from the left.

Ctrl-L is the download command to the terminal mode subsystem. A <CR> is transmitted to the external computer, and the object code, in INTEL-standard format, is loaded into the microcomputer's memory at the addresses specified in the load blocks. Downloading can be interrupted at any time by typing <ESC> on the principal I/O device keyboard. This key transfers control to the terminal mode subsystem.

Ctrl-R simply returns control to the program which called the terminal subsystem. This calling program is usually ARIAN II.

More information is available on the TERM command in Appendix I.

2 The Utility Subsystem

The UTIL command invokes the Utility subsystem. This subsystem gives the user the ability to examine and modify memory directly. In response to this command, the user is prompted with a slash. He may then enter any of the following UTILity commands:

1. C <hadr> <hadr> <hadr> -- copy the block of memory defined by the first two addresses to the location in memory starting at the third address. The original block is unchanged unless the third address resides within this block.
2. D <hadr>? <hval>* -- deposit the specified values into memory.
3. E (<hadr> <hadr>?)? -- examine a specific memory location or block of memory.
4. F <hadr> <hadr> <hval>* -- find all occurrances of the specified byte string in the specified memory block.
5. S <hadr>? -- set/display the default pointer.
6. X -- return to ARIAN Command mode.

These commands are also interpreted like MCS commands. If more than one value is specified in the command, successive values are separated by one or more spaces. <hadr> is a 16-bit quantity, and <hval> is an 8-bit quantity; <hadr> is an address, and <hval> is a value. Refer to Appendix I for more detailed information on the Utility subsystem. The Utility subsystem commands are derived from MCS V1.6; the most complete description of these commands is available in the "Monitor Command System User's Manual".

CHAPTER 7

SUMMARY of the ARIAN II COMMANDS

ARIAN II supports over 30 Executive, or Level 2, commands. Several of these commands have been mentioned so far, and Appendix I covers all of these commands in detail.

The Executive commands are grouped into nine catagories. They are:

1. System Control
2. Primary File Editing
3. Local File Control
4. Disk File Control
5. Local/Disk File Transfer
6. List/Print
7. Program Debugging
8. Assembler
9. Utility

1 System Control

The System Control commands are CMND, CONT, CUST, EXEC, EXIT, RESET, SETC, TABS, and TERM.

The CMND command toggles the Level 3 command mode. It is used to engage and disengage Command Level 3 as well as specify the Command Drive number.

The CONT and EXEC commands are used to explicitly execute a program or continue the execution of a program. EXEC has a large number of options, including assembling and executing the primary file, loading and executing a disk file, and executing a program stored in memory. CONT is used only to execute a program stored in memory; it is generally used to continue from a breakpoint. This command loads the registers from the register save area and then branches to the program in memory.

CUST and TERM are the customized command and the Terminal subsystem described earlier.

EXIT returns to FDOS.

RESET is the same as a warm start. Refer to Appendix I for a detailed description of the effects of a warm start.

SETC is used to redirect either input, output, or both to a program located in memory.

Finally, TABS is used to set and reset the tab stops used by the input line editor.

2 Primary File Editing

The Primary File Editing commands are <lnum>, APND, DEL, EDIT, FIND, ISRT, and RNUM. These were discussed earlier.

3 Local File Control

The Local File Control commands are FCHK, FILE, LDEL, LDIR, LNAME, LSCR, and RCVR. There were also discussed earlier.

4 Disk File Control

The Disk File Control commands are DDEL, DDIR, and DNAME. They were discussed earlier.

5 Local/Disk File Transfer

The Local/Disk File Transfer commands are LOAD and SAVE. They were discussed earlier.

6 List/Print

The LIST command is used to list all or selected portions of the primary file on the principal I/O device; PRINT prints all or selected portions of the primary file on the printer. If the file is in ARIAN II assembler format, LISTF and PRINF will list and print in columnar format.

7 Program Debugging

The Program Debugging command is BREK. It is used to set, reset, and examine breakpoints in memory. A breakpoint takes the form of a one-byte subroutine call which returns control to ARIAN II, preserving the current values of the registers and the Program Counter.

When a breakpoint is set, the byte of the program addressed by the user is saved in the breakpoint save area and it is replaced by a RST 1 instruction. When breakpoints are reset the replaced byte is copied back

to its previous location.

Upon encountering a breakpoint, the breakpoint is automatically reset. The user may then examine the contents of the registers, which is stored in the register save area, and continue program execution by using the CONT command.

8 Assembler

The assembler commands are ASSM and SYMT.

SYMT displays the symbol table from the last assembly on the principal I/O device. The user should employ this command immediately after the assembly; the symbol table is destroyed by disk accesses and the execution of Level 3 commands.

ASSM is used to assemble the primary file. ASSM produces no listing, ASSML produces a normal listing, ASSMP prints the listing on the printer, ASSMF lists the output in formatted mode on the principal I/O device, and ASSMX prints the file in formatted mode on the printer without generating code.

9 Utility

The UTIL command invokes the Utility subsystem; this was discussed earlier.

References

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

Conn, Richard. "ARIAN -- An Implementation of a Microcomputer Operating System." May, 1978; MS Thesis, Graduate College, University of Illinois, Urbana, IL 61801. 187 pp.

Conn, Richard. "ARIAN: A Software Development System for ARIES-1, a Z80-Based Microcomputer." Personal Computing Proceedings, 1979 National Computer Conference, June 4-7, New York, New York; American Federation of Information Processing Societies (AFIPS), AFIPS Press, 210 Summit Avenue, Montvale, New Jersey 07645; p. 363-369.

Conn, Richard. "The Monitor Command System User's Manual." Feb, 1978; Department of Computer Science, University of Illinois, Urbana, IL 61801; Technical Report Number UIUCDCS-R-78-912 and Engineering Report Number UILU-ENG 78 1705. 31 pp.

Conn, Richard. "SDL -- A String Description Language." Project ARIES Report; 24 June 1979; 8 pp.

Conn, Richard. "DEBUG (DBUG) USER'S MANUAL." Project ARIES Report; 1 July 1979; 4 pp.

Conn, Richard. "The USER'S MANUAL for the TEXT FORMATTING SYSTEM." Project ARIES Report; 8 July 1979; 19 pp.

Conn, Richard. "The ARIAN II USER'S MANUAL." Project ARIES Report; 8 July 1979; 150 pp. approximately.

Conn, Richard. "MEMORY TEST USER'S MANUAL." Project ARIES Report; 17 July 1979; 2 pp.

Conn, Richard. "ARIAN DISASSEMBLER USER'S MANUAL." Project ARIES Report; 17 July 1979; 3 pp.

Conn, Richard. "XEDIT USER'S MANUAL." Project ARIES Report; 17 July 1979; 4 pp.

Intel Corp. "8080 Microcomputer Systems User's Manual." Sept, 1975; Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

Intel Corp. "8080 Assembly Language Programming Manual." 1976; Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

Intel Corp. "Interp/80 User's Manual." 1975; Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

Intel Corp. "8008 and 8080 PL/M Programming Manual." 1975; Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051.

Zilog, Inc. "Z80-Assembly Language Programming Manual." 1977; Zilog, 10460 Bubb Road, Cupertino, CA 95014.

MOSTEK. "Z80 Programming Manual." 1977; MOSTEK, 1215 W. Crosby Rd, Carrollton, TX 75006.

North Star Computers, Inc. "The North Star MONITOR." 1977; North Star Computers, 2547 Ninth St, Berkeley, CA 94710.

North Star Computers, Inc. "The North Star Disk Operating System." 1977; North Star Computers, 2547 Ninth St, Berkeley, CA 94710. Version 2, Release 3.

North Star Computers, Inc. "Micro-Disk System MDS-A." 1977; North Star Computers, 2547 Ninth St, Berkeley, CA 94710.

Shugart Associates. "SA400 Minifloppy Diskette Storage Drive." 1977; Shugart Associates, 415 Oakmead Parkway, Sunnyvale, CA 94086. OEM Manual.

Processor Technology Corp. "CUTS User's Manual." 1977; Manual No. 730005; Processor Technology, 7100 Johnson Industrial Way, Pleasanton, CA 94566.

Processor Technology Corp. "ALS-8 User's Manual." 1977; Manual No. 727013; Processor Technology, 7100 Johnson Industrial Way, Pleasanton, CA 94566.

Processor Technology Corp. "SOLOS/CUTER User's Manual." 1977; Processor Technology, 7100 Johnson Industrial Way, Pleasanton, CA 94566.

Digital Research. "An Introduction to CP/M Features and Facilities." 1976; Digital Research, PO Box 579, Pacific Grove, CA 93950.

Digital Research. "CP/M Interface Guide." 1976; Digital Research, PO Box 579, Pacific Grove, CA 93950.

The ARIAN II EXECUTIVE COMMANDS

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

12 July 1979

APPENDIX I

The ARIAN II EXECUTIVE COMMANDS

Contents

Introduction ..... 1 - 4
Commands ..... 6 - 59

Command Name	Page	Command Name	Page
<lnum>	6	ISRT	37
APND	7	LDEL	38
ASSM	9	LDIR	39
BREK	11	LIST	40
CMND	13	LNAME	42
CONT	15	LOAD	43
CUST	16	LSCR	45
DDEL	18	PRINT	46
DDIR	19	RCVR	47
DEL	20	RESET	48
DNAME	21	RNUM	49
EDIT	22	SAVE	50
EXEC	27	SETC	52
EXIT	30	SYMT	53
FCHK	31	TABS	54
FILE	32	TERM	55
FIND	35	UTIL	57

APPENDIX I

The ARIAN II EXECUTIVE Commands

Commands by Functional Group

Name/Group	Page
1. System Control	
CMND	13
CONT	15
CUST	16
EXEC	27
EXIT	30
RESET	48
SETC	52
TABS	54
TERM	55
2. Primary File Editing	
<lnum>	6
APND	7
DEL	20
EDIT	22
FIND	35
ISRT	37
RNUM	49
3. Local File Control	
FCHK	31
FILE	32
LDEL	38
LDIR	39
LNAME	42
LSCR	45
RCVR	47
4. Disk File Control	
DDEL	18
DDIR	19
DNAME	21
5. Local/Disk File Transfer	
LOAD	43
SAVE	50
6. List/Print	
LIST	40
PRINT	46
7. Program Debugging	
BREK	11
8. Assembler	
ASSM	9
SYMT	53
9. Utility	
UTIL	57

ARIAN II EXECUTIVE COMMANDS

Introduction

ARIAN II has an extensive set of Executive, or Level 2, commands which give the user a great deal of control over the microcomputer system and the creation of his programs. This control includes the abilities to:

1. Examine and modify memory directly,
2. Control the system environment,
3. Control, modify, and execute all programs available to the user,
4. Debug user programs through specialized system commands, and
5. Assemble programs in ARIAN II Z80 assembly language.

The purpose of this appendix is to catagorize the Executive commands of ARIAN II and explain them in detail. The two contents pages at the beginning of this appendix are designed to be used for quick reference; they index the commannds alphabetically and by function. The descriptions of the commands start on page 6; the commands are ordered alphabetically throughout this appendix.

The entries for each command consist of the following parts:

1. The name of the command,
2. A brief description of the function of the command,
3. A pseudo-SDL representation of the formats the commands can assume,
4. A more detailed description of the function of the command, and
5. Examples of the use of the command.

The following is a String Description Language (SDL) representation and explanation of the "obvious" tokens used in the format representations.

<blank> : ' '
<digit> : '0' ! ... ! '9'

```
<hexdigit> : <digit> ! 'A' ! ... ! 'F'
<alpha> : 'A' ! ... ! 'Z' ! 'a' ! ... ! 'z'
<alphit> : <alpha> ! <digit>
<quote> : '"'
<text> : "any sequence of ASCII characters whose hexadecimal
        values range from 20H to 7EH"
<string1> : <quote> <text>
<string> : <string1> <quote>
<cmndname> : <alpha> <alphit>*3
<fname> : <alpha> <alphit>*7
<lnum> : <digit> <digit>*3
<inc> : <lnum>
<hadr> : <digit> <hexdigit>*3 ! '0' <hexdigit>*4
<delim> : <blank> ! ','
COMMAND_NAME : <cmndname>
FILE_NAME : <fname>
LINE_NUMBER : <lnum>
HEXVAL : <hadr>
ARIAN_COMMAND : <cmndname> <alphit>* <blank>+
        <fname>? <delim>+
        (<string1> ! <string> <delim>* <string1> !
                <string> <delim>* <string>)
        (<delim>+ (<lnum> ! <hadr>) <delim>*) *3
```

As can be seen by the above description, a command in ARIAN II (ARIAN_COMMAND) can take a large number of forms. It can be thought of as being divided into up to seven free-format fields. These fields are, in order:

1. The name of the ARIAN II command and its affixed options,
2. The name of an ARIAN II file or command [optional],
3. Up to two strings, enclosed in quote marks (") [optional],
4. Up to three line numbers and/or hexadecimal values [optional].

Hence, the following are examples of valid ARIAN II command forms:

```
FILE PROGRAM

LIST 100,2000

LIST 100  2000

TEST  3000  0F0F0  40

TEST2 3000,0F0F0,40

CUSTD TEST

SAVEB1 BINARY1 3400 34FF 06800

FINDF "THIS IS A TEST"

SUBS "THIS IS IT""THIS IS A TEST"

SUBS "THIS IS A TEST", "THIS IS A TEST"

ALLOFIT <fname> "STRING1" "STRING2" 10,20 30

allofit filename "STRING1", "STRING2" 10,20 30
```

The entries in several fields of an ARIAN II command are automatically capitalized internally by the command and argument parser of ARIAN II. Specifically, these fields are the command name and the appended characters (<cmndname> <alphit>*), the file name (<fname>), and the alphabetic hexadecimal digits in the hexadecimal value fields (<hadr>). Hence, the user can type his commands in upper or lower case characters and they will be interpreted in exactly the same way. Also by this token, all command names, options fields, and file names are forced to consist of capitalized characters by the system.

For example, the following forms of commands are equivalent:

Form 1	Form 2
allofit filename "string1"	ALLOFIT FILENAME "string1"
saveb1 binary1 3400 34ff 6800	SAVEB1 BINARY1 3400 34FF 6800
list 100,200	LIST 100 200

In the case of the command "ALLO", the referenced file is named "FILENAME" and the string is "string1". Note that the capitalization rule does not affect strings.

In the case of the command "SAVE", the referenced file is named "BINARY1" and the addresses are 3400, 34FF, and 6800.

In the case of the command "LIST", the lines are 100 and 200.

ARIAN II EXECUTIVE COMMANDS

Command: <lnum>

Brief Description: Any line starting with a line number inserts that line into the primary file at the correct position.

Format: <lnum> <text>

The user may insert a line of text into the primary file or replace a line already in the primary file by simply typing the desired line number followed by a <SP> and the text of the line. This feature of ARIAN provides for very simple line insertion and replacement.

Example: 325 THIS IS LINE 325

Result: The above line (325) is entered into the primary file. If line 325 already exists, it will be replaced by the above line; if it does not exist, line 325 will be inserted between lines 324 and 326 (or the nearest lines to this range) in the primary file.

Command: APND

Brief Description: APND places the following block of lines after the specified or implied line in the file. The block of lines is then typed by the user in block line entry mode.

Format: APND <lnum>?

Format: APNDN <lnum>?

The APND (append) command is one of two block line entry commands of the ARIAN editing subsystem. Block line entry in ARIAN permits the user to type an arbitrary number of lines into the primary file without typing their corresponding line numbers. When the user has finished typing this group of lines, he then types a Ctrl-C followed by a <CR>. These lines will then be entered into the primary file at the appropriate place. If the Ctrl-C is at the end of a line, it will not be included in this line, and this line will be the last line of the block; if Ctrl-C is the first character of a line, the previous line will be the last line of the block.

APND without a line number will enter the block of lines after the last line of the file. APND with a line number will enter the block of lines after the specified line and before the next line in the file.

APND will renumber the file once the block line entry mode is exited, and APNDN will not.

Example: APND

Result: The user is prompted with a "?", after which he types a line of text. If this line is not terminated by a Ctrl-C and <CR>, another prompt appears and he may then type another line of text. This process

continues until he either ends a line with a Ctrl-C <CR> or begins a line with a Ctrl-C <CR>. At this time, the block of lines he has just typed will be appended to the primary file and the primary file will be renumbered.

Example: APND 100

Result: The same procedure is executed as described above, but the block of lines is inserted between line 100 and the next line in the file.

Command: ASSM

Brief Description: ASSM causes ARIAN to assemble the primary file.

Format: ASSM <fname>? (<hadr> <hadr>?)?

Format: ASSMF <fname>? (<hadr> <hadr>?)?

Format: ASSML <fname>? (<hadr> <hadr>?)?

Format: ASSMP <fname>? (<hadr> <hadr>?)?

Format: ASSMX <fname>? (<hadr> <hadr>?)?

The ASSM command instructs ARIAN to assemble the primary file. If <fname> is specified, an entry is made in the binary file directory which specifies the boundaries of the object code generated by the assembler. If no address is given, the object code generated by the assembler is placed immediately after the local text file workspace; if one address is given, the object code is placed starting at this address; if two addresses are given, the object code is assembled to execute at the first address and it is physically placed in memory starting at the second.

ASSM assembles the primary file and lists only the lines with errors in them. ASSML lists the file in paged mode on the user's CRT as it is being assembled, ASSMF lists the file in formatted paged mode on the user's CRT as it is being assembled, ASSMP prints the file on the user's printer as it is being assembled, and ASSMX lists the file in formatted paged mode on the user's printer as it is being assembled and does not place any code into memory.

Example: ASSM T1 5800 6800

Result: The primary file will be assembled to execute at location 5800, and the object code will be placed at location 6800. The local binary file T1, which defines the limits of the assembly in terms of the 5800 address will be produced.

Example: ASSML

Result: The primary file is assembled at the end of the workspace. All lines with their object code are listed on the user's CRT. For example, if the workspace is defined to be 2000 to 5000 hexadecimal, the object code will be placed starting at 5001.

Command: BREK

Brief Description: BREK is used to set, reset, and display breakpoints.

Format: BREK <hadr>

Format: BREKD <hadr>

Format: BREKL

Format: BREKS

The breakpoint (BREK) commands permit the user to set, reset, and display the breakpoints. When a breakpoint is reached during a program's execution, the byte replaced by the breakpoint is restored, the contents of the registers is displayed to the user, the contents of all the registers are saved in the register save area of ARIAN, and control is returned to ARIAN. If the user wishes to resume program execution, he may do so by using the CONTinue command.

BREK will set a breakpoint at the specified address; BREKD will clear the breakpoint at the address specified if there is one and inform the user if there is no breakpoint at this address; BREKL will list the addresses of all breakpoints set by the user that have not yet been cleared; and BREKS will scratch (clear) all user breakpoints that have not yet been cleared.

Note: a breakpoint is cleared when it is encountered during a program's execution.

Example: BREK 4000

Result: A one-byte breakpoint is set at location 4000 hexadecimal. If the user's program later encounters this breakpoint, control will be restored to the user through ARIAN.

Example: BREKL

Result: The addresses of all remaining breakpoints will be listed.

Example: BREKS

Result: All remaining breakpoints will be scratched.

Command: CMND

Brief Description: CMND is used to toggle the level 3 command facility and set the number of the CL3 drive.

Format: CMND <cdrive>?

If <cdrive> is specified, the indicated drive number (1-4) will be made the CL3 drive. If it is not specified, CL3 will simply be toggled; if CL3 is on, it will be turned off, and if CL3 is off, it will be turned on with drive 1 being designated the CL3 drive.

The CL3 drive is the drive whose disk is searched for a level 3 command if the search for the current user command fails at levels 1 and 2.

The first digit of the ARIAN II prompt indicates the status of CL3. If this digit is 0, CL3 is off; otherwise, CL3 is on and this digit indicates the number of the CL3 drive.

Example: 01>CMND

Result: 11>
CL3 was disengaged when CMND was typed, and drive 1 was designated to be the CL3 drive as a result.

Example: 11>CMND 4

Result: 41>
Drive 4 was designated the CL3 drive.

Example: 41>CMND

Result: 01>
CL3 was disengaged.

Example: 01>CMND 2

Result: 21>
Drive 2 was designated as the CL3 drive.

Command: CONT

Brief Description: CONT allows the user to continue from a breakpoint or begin execution of a program with specific register values.

Format: CONT <hadr>?

The CONTinue command is one of two commands which allow the user to explicitly execute a program in the microcomputer's memory (EXEC is the other). CONT may be used to either continue a program's execution after a breakpoint has been encountered or explicitly execute a program at a specified address.

When a breakpoint is encountered, the contents of all the registers of the microprocessor are saved in the register save area in the ARIAN scratchpad RAM, thereby permitting the user to examine and modify these values at his leasure through the REGS level 3 command. CONT with no argument will permit the user to continue execution of the program under test with the registers of the microprocessor loaded with the values stored in the register save area.

CONT with a specified address as an argument also loads the registers of the microprocessor, but execution is begun at the address specified. Hence, dynamic debugging of user subroutines is possible by using this command.

Example: CONT 4000

Result: The values stored in the register save area are loaded into the appropriate registers and the program starting at location 4000 hexadecimal is executed.

Command: CUST

Brief Description: CUSTomize allows the user to create, delete, or rename a customized command. Provision is also made to list and scratch all customized commands.

Format: CUST <cname> <hadr>?

Format: CUSTD <cname>

Format: CUSTL

Format: CUSTN <cname>

Format: CUSTS

CUSTomize is perhaps one of the most powerful commands in ARIAN's repertoire. This command allows the user to add his own set of commands to those already executed by ARIAN. The user may give his additional commands any names he wishes, including the names of the commands already defined by ARIAN. If the user wishes to redefine an ARIAN command in this manner, his customized subroutine replaces the system subroutine normally used to execute that command. This replacement is in effect until the user resets ARIAN or deletes his customized command. Other options under the CUST core include CUSTD to delete a specified customized command, CUSTL to list all the customized commands and their execution addresses, CUSTN to rename a customized command, and CUSTS to scratch (delete) all the customized commands.

The CUST command by itself is used to create or redefine a customized commmand. CUST with no address creates a customized command which will begin execution at the default address; CUST with an address defines the command explicitly to execute at the specified address. If a customized

command of the same name already exists, the user is prompted with the "REPLACE?" message, to which he responds with a "Y" if he wishes to redefine the execution address of that customized command or "N" if he does not.

The functions of CUSTS, CUSTL, and CUSTD are executed without any particular response to the user. CUSTN prompts the user with "NAME?", to which he may respond with the new name of the file or just a <CR> to abort. The new name is terminated with a <CR>.

Example: CUST TEST

Result: The customized command "TEST" is created. It executes at the current value of the default assembly address.

Example: CUSTL

Result: TEST B800

The names of all the customized commands and their execution addresses are displayed.

Command: DDEL

Brief Description: DDEL allows the user to delete the specified disk file.

Format: DDEL <fname>

Format: DDELn <fname>

DDEL deletes the specified disk file from the disk file directory. Only the directory entry is deleted, but the space occupied by the file is released and subject to user compaction and the disk management routines.

As with all disk-related commands, the disk drive number may be specified as a fifth character of the command.

Example: DDEL TEXT1

Result: The entry for the disk file TEXT1 is deleted from the disk directory.

Example: DDEL3 TEXT1

Result: The entry for the disk file TEXT1 residing on disk drive 3 is deleted from that disk directory. Drive 3 becomes the logged-in drive.

Command: DDIR

Brief Description: DDIR displays an organized listing of the directory of the disk on the specified drive.

Format: DDIR

Format: DDIRn

The disk directory command (DDIR) displays the contents of the directory of the disk currently mounted on the specified drive. Each directory entry contains the name of the file, the length of the file as a decimal number of 256-byte blocks, the type of the file, and the execution or load address of the file if it is a binary file. The types of files permitted by ARIAN are general text files (type 0), binary files (type 1), BASIC program files (type 2), and BASIC data files (type 3).

The disk drive number of the desired drive may be specified as the fifth character of the command. If this is done, that drive automatically becomes the logged-in drive.

Example: DDIR2

Result: The disk directory of the disk on drive 2 is displayed to the user. Drive 2 becomes the logged-in drive.

Command: DEL

Brief Description: DEL allows the user to delete either a specified line or block of lines from the primary file.

Format: DEL <lnum> <lnum>?

The function of the DELete command is to delete lines from the primary file. DEL followed by a line number will delete only that line; DEL followed by two line numbers will delete the block of lines enclosed by the specified lines, inclusive. If a specified line number does not exist in the file, the line number of the line which would follow the specified line if it existed will be used; if the specified line number is larger than the largest line number in the file, the last line will be deleted.

Example: DEL 100

Result: Line 100 is deleted from the primary file.

Example: DEL 100 150

Result: Lines 100 to 150, inclusive, are deleted from the primary file.

Command: DNAME

Brief Description: DNAMe allows the user to rename a specified disk file.

Format: DNAM <fname>

Format: DNAMn <fname>

The disk file rename (DNAM) command renames the specified disk file. In response to this command, ARIAN will prompt the user with "NEW NAME?", to which the user may type the new name for the file or just a <CR> to abort the renaming function.

Like the other disk-related commands, the fifth character of DNAM may be a digit from 1 to 4, specifying the number of the disk drive the file resides on. This drive will be the new logged-in drive as the result.

Example: DNAME OLDFILE

Result: ARIAN will respond with "NEW NAME?", to which the user may respond with the new name of the file OLDFILE. A <CR> will abort the process.

Command: EDIT

Brief Description: EDIT allows the user to edit a specified line of the primary file. It is an intra-line editor.

Format: EDIT <lnum>

The EDIT command invokes the ARIAN intra-line editor. The editor allows the user to edit a line that has already been typed without retyping the entire line. The specified line, or the line that would follow the specified line if this line does not exist, will be edited.

The intra-line editor is a dynamic editor which permits the user to see the effects of his editing commands immediately after he types them. When a line is edited, it is copied into the editor's old line buffer and then displayed to the user. The editor then does a <CR> and prompts the user with a "?". As the user edits this line, each character of the new line that is created is placed into the editor's new line buffer; the original line in the old line buffer is not affected. Finally, when editing is finished, the user may type a <CR> to terminate the editing process and replace the original line in the file with the line as it exists in the new line buffer.

The intra-line editor responds to a number of subcommands. The following is a complete list of these commands and their functions.

1. <BS> -- back up the new line pointer and delete the previous character. This command, echoed as a backspace, is used to delete the previous character in the line. Only the new line pointer is affected by it.
2. <CR> -- terminate creation of the new line. This command terminates editing of the line and replaces the original line in the primary file with the line that currently exists in the new line buffer. If <CR> is the first editing character typed, the edit is aborted and no

replacement occurs.

3. <DEL> -- the delete key backs up the new line pointer. The characters backed over are enclosed in "<" and ">" and deleted from the new line. Only the new line pointer is affected by this command.

4. <SP> -- the space bar functions to copy the character pointed to by the old line pointer into the character position pointed to by the new line pointer and advance the old line and new line pointers by one. The space bar, therefore, will simply copy the next character from the old line buffer into the new line buffer. After the copy is done, the copied character will be displayed to the user.

5. A -- abort the editing of the old line. This command may be typed whenever the editor is ready to receive a command (i.e., the editor is not in the middle of an insertion or replacement). It terminates the edit and returns control to ARIAN without affecting the original line.

6. D -- delete the character pointed to by the old line pointer (delete the next character in the old line). The character is deleted by advancing the old line pointer by one character position and not affecting the new line pointer. The deletion is displayed to the user as a backslash ("\") followed by the deleted character. If the next command typed by the user is another D, the next deleted character is displayed (without the backslash). This will continue until the user types some other command, in which case a closing backslash will be printed and that command will be executed. In effect, the deleted characters will be enclosed in backslashes when displayed to the user.

7. E -- skip to the end of the line. The rest of the characters in the old line buffer are copied into the new line buffer and both pointers are advanced to point to the non-existent character after the last character copied. The copied characters are displayed to the user as they are copied.

8. I -- insert a string of characters in front of the character currently pointed to by the old line pointer. In response to the I typed by the user, the editor types a slash ("/"). The user may they type any string of characters he wishes except for an escape or carriage return. These characters will be copied into the new line buffer, the new line pointer will be advanced, and each character will be echoed to the user as he types it.

The escape and carriage return characters are special characters to the ilsert subcommand. <ESC> instructs the insert subcommand to end the insertion. The editor then types another slash to indicate that the insertion is finished and allows the user to continue editing normally. <CR> instructs the editor to terminate creation of the new line, copy the new line into the primary file, and return to ARIAN command mode. The <CR> is echoed as a slash, a carriage return, and a system prompt, indicating that ARIAN is now in command mode.

The backspace character performs its normal function while in this mode.

9. P -- print the new line and edit it. The command will terminate the new line at the current position of the new line pointer, copy the new line buffer into the old line buffer, print the new line, and restart the editing process with this new line instead of the original line. The original line as it exists in the primary file is not affected.

10. R -- replace the characters pointed to by the old line pointer with the following string. Both pointers are advanced and the new characters are echoed to the user. No special character is typed to the user after he types an R, and the <ESC>, <CR>, and <BS> characters respond as in the I command except that no slash is printed. In effect, as the user types his string, each character he types replaces the corresponding character in the old line buffer.

11. S <letter> -- skip to the specified letter. This is the only two-character command in the editor; it consists

of the letter S followed by a single character. When this command is typed, both the old and new line pointers are advanced and the corresponding characters are typed and copied into the new line buffer until the specified character is found or the end of the line is reached. Once the specified character is found, the old line pointer will point to it and this character will not be printed; it will be the next character in the line. The S and the specified letter are not echoed to the user when the command is typed. This command is very useful, particularly when the user wishes to insert, delete, or replace at a specified character; he does not have to space over to that character with this command.

12. X -- exit and reedit the old line. The X command terminates the editing done so far and restarts the edit of the original line. If a P command has been previously typed, the last line placed into the old line buffer is reedited.

The editor has four error messages that it may display. These messages are:

1. ?? -- invalid command. A double question mark indicates that an invalid command has been typed. No recovery is required by the user; he may continue with the desired command.

2. ** -- end of edit line. A double asterisk indicates that the user has tried to go beyond the end of the original line illegally while editing.

3. *EOL* -- end of line buffer. The length of the new line has just reached the limits of the new line buffer, and the user must reedit the original line.

4. <BEL> -- <BS> error. The user has typed a <BS> which attempted to delete a character before the first character in the line.

Example: EDIT 200

Result: Line 200 is printed and the user is prompted with a "?". The user may now edit line 200 using the intra-line editing commands. One useful aspect of this command is that line 200 may be copied as line 201 or any other desired line number by editing only the line number and typing the E (skip to end of line) followed by a <CR>. Line 200 in the primary file will be unchanged and line 201 will be created by this example.

Command: EXEC

Brief Description: EXECute functions to allow the user to execute or assemble and execute a specified or implied file or program.

Format: EXEC (<fname> ! <hadr>)?

Format: EXECn (<fname> ! <hadr>)?

Format: EXECB <fname>?

Format: EXECF <fname>?

Format: EXECL <fname>?

Format: EXECP <fname>?

The EXECute command is one of the most complex of the ARIAN commands. EXEC allows the user to execute a text file, a binary file, or any program or subroutine which resides in memory or on disk. The command also permits a simple, clean return to ARIAN by pushing a return address onto the system stack (which may also be used as the program stack); by keeping the stack stable, the user may return to ARIAN when his program is finished by simply executing a return.

The EXEC command works in many implicit modes. EXEC with no arguments will assemble and execute the primary file. This means of program execution makes software development somewhat easier by permitting the user to execute his primary file, interrrupt the program if it malfunctions, edit the file, and reexecute it with a minimum of effort.

EXEC <fname> will perform the following operations in the order specified:

1. It will first search the local file directory for

the specified file. If this file is found, it will be made primary, assembled, and executed.

2. Secondly, if the search in step 1 fails, ARIAN will search the disk directory of the logged-in disk drive for the specified file. If the file is found, it will be loaded into memory.

3. If step 2 fails, an appropriate error message will be given. If step 2 succeeds, a test will be made to see if the file is binary or text. If it is binary, it will be executed; if it is text, it will be assembled and executed.

4. In all cases except when an error occurs, a binary file of the specified name is created.

If EXEC <hadr> is used, the binary program starting at the specified address is executed.

Finally, if EXECB is used, the specified binary file (as defined in the binary file directory) will be executed. If no file name is specified, execution will begin at the default execution address.

The EXECF, EXECL, and EXECP variants perform functions similar to those of ASSM. EXECF produces a formatted listing on the principal I/O device, EXECL produces a simple listing on the principal I/O device, EXECP produces a simple printing on the printer.

Example: EXEC PROGRAM

Result: If PROGRAM is a local text file, it will be assembled and executed. If PROGRAM is not local and resides on disk, it will be loaded, assembled if it is a text file, and executed.

Example: EXEC 0F000

Result: The machine code beginning at location 0F000 hexadecimal is executed.

Example: EXECB PROGRAMB

Result: The binary file PROGRAMB is executed.

Command: EXIT

Brief Description: EXIT allows the user to transfer control to FDOS.

Format: EXIT

EXIT transfers control to the FDOS disk bootstrap program. FDOS is a modified version of Northstar Corporation's DOS program. Refer to the "FDOS User's Manual" for a detailed description of FDOS.

The system status of ARIAN is not affected by this transfer unless the user explicitly or implicitly changes ARIAN or its system RAM area after the control has been transferred.

Reentry may be made to ARIAN in two ways after the transfer has been accomplished: (1) by branching to the ARIAN warm start address (memory location 4) or (2) by branching to the ARIAN cold start address (memory location 0). Reentry at the warm start address preserves the state of ARIAN at the time of the exit.

Command: FCHK

Brief Description: FCHK functions to check the validity of a local text file. This command checks the specified or implied file as to its correctness in format.

Format: FCHK <fname>?

The file check (FCHK) command is used to determine the validity of the specified local text file. This verification includes checking to see that the character count for each line is correct, each line terminates with a <CR>, the file is properly terminated by an <EOF> mark (binary 1), and the local text file directory limits of the file are correct. ARIAN will respond with "VALID FILE" or "INVLD FILE" when the test is finished.

The FILE, RCVR, and LOAD commands do an implicit file check whenever they are executed, but only the invalid message is displayed by their implicit checks.

When a secondary file is checked for validity it is not made primary. There is no overall affect on the ARIAN system as a result of executing this command.

Command: FILE

Brief Description: The FILE command allows the user to explicitly create a new primary local text file or a binary file or view the directory information on the current primary local text file.

Format: FILE (<fname> <hadr>?)?

Format: FILEB <fname> (<hadr> <hadr>)?

The FILE command is one of the most powerful and complex commands in ARIAN. This command is used to create the primary file or a binary file and display the directory entry for the primary file.

The FILE command with no arguments displays the directory entry for the primary file. This entry, like the entries for the secondary local files, consists of the name of the file and the starting and ending memory addresses of the file.

The FILE <fname> variant creates a primary file of the name specified. If a local file already exists with this name, it is made primary; the memory manager is invoked, the new primary file is moved to the physical end of the old primary file, and the files are compacted within the currently-defined workspace boundaries. If no local file with this name exists, the new primary file of length zero is created at the end of the last primary file and compaction is again done. Text may now be entered into the primary file by typing line numbers or using the APND command.

If an address is specified with the FILE <fname> variant, the new primary file is placed at this address. Compaction is done to the secondary files, but the new primary file is unaffected by this compaction. Also, the workspace boundary parameters are ignored when the primary file is placed, permitting the user to place this file anywhere he wishes. This FILE variant, then, effectively overrides the memory manager; however, if a later command uses the memory manager, the primary file may be moved and

compacted by the memory manager implicitly.

FILEB permits the user to explicitly create a binary file. Binary files are defined solely by their entries in the local binary file directory; they conform to no particular physical structure as they exist in memory. FILEB with no numeric argument creates a binary file with the specified name at the default binary file limits set by the last assembly; FILEB with the two addresses creates the binary file with these addresses as its boundaries.

Example: FILE NEW

Result: If file NEW already exists locally, it is made primary; otherwise, a new text file is created with the name "NEW" at a location determined by the workspace manager. File compaction and workspace compression is then done to all local files.

Example: FILE

Result: The local directory entry for the current primary file is displayed. If this follows the above example and NEW did not previously exist, then an entry like

NEW 3020 3020

would be displayed to the user. This indicates that the current primary file is named NEW, it is a null file, and it begins and ends at location 3020 hexadecimal. If NEW is not null, an entry like

NEW 3020 304C

would be displayed to the user. In this case, NEW resides at locations 3020 to 304C hexadecimal, inclusive.

Example: FILEB BINARY 5000 5010

Result: The local binary file directory entry for the file named BINARY is created. This command defines the file BINARY to reside at locations 5000 to 5010 hexadecimal, inclusive.

Command: FIND

Brief Description: FIND searches the primary file for all occurrances of the specified string.

Format: FIND '"' <string> '"'?

Format: FIND '"' <string> '"' <lnum>

Format: FINDF '"' <string> '"'?

Format: FINDF '"' <string> '"' <lnum>

Format: FINDP '"' <string> '"'?

Format: FINDP '"' <string> '"' <lnum>

The find command performs a search through the primary file for the specified string. If no line number is specified, the search is done over the entire file; if a line number is specified, the search is done starting at the specified line and extending to the end of the file.

The output from the FIND command is a paged listing of the lines which contain the specified string. If the listing is printed on the printer, the output is not paged.

The FIND command by itself prints the lines as they are currently formatted; FINDF prints the lines in assembler format; and FINDP prints the lines as they are currently formatted on the printer.

Example: FIND "test" 40

Result: All lines after line 40, inclusive, containing the string 'test' will be printed.

Example: FINDF "test

Result: All lines in the file containing the string 'test' (assuming no blanks exist after the word 'test' in the command) will be printed in assembler format.

Command: ISRT

Brief Description: Insert the following block of lines before the specified line in the primary file.

Format: ISRT <lnum>

Format: ISRTN <lnum>

The ISRT command is the same as the APND command, except that the ISRT command places the block of lines in front of the specified line while APND places the block after the specified line. ISRT must have a line number, and it is particularly useful in inserting lines before the first line in the file. APND, on the other hand, is useful for appending lines to the end of the file.

ISRT will renumber the file, and ISRTN will not.

Example: ISRT 300

Result: The following block of lines entered in block line entry mode will be placed in front of line 300.

Command: LDEL

Brief Description: LDEL functions to delete the local text file or binary file specified.

Format: LDEL <fname>

Format: LDELB <fname>

The local file delete command (LDEL) is used to delete the directory entry of the specified local binary or text file. This command only deletes the directory entry; the physical file is unaffected by it. LDEL deletes the specified text file, while LDELB deletes the specified binary file.

Command: LDIR

Brief Description: LDIR displays the specified local file directory to the user.

Format: LDIR

Format: LDIRB

The local directory command displays the local text and binary file directories. LDIR displays the local text file directory, and LDIRB displays the local binary file directory. All directory entries consist of the names of the files and their current memory address boundaries. LDIR will display the entry for the current primary file first, and this entry is followed by the entries for the secondary files.

LDIRB displays the local binary file directory and the limits from the last assembly.

Example: LDIR

Result: A list of al the current local text files is displayed to the user. The first file is the primary file.

Example: LDIRB

Result: The local binary file directory is displayed.

Command: LIST

Brief Description: The LIST command is used to list all or selected parts of the primary file on the principal I/O device.

Format: LIST (<lnum> <lnum>?)?

Format: LISTF (<lnum> <lnum>?)?

Format: LISTN (<lnum> <lnum>?)?

The LIST command allows the user to display all or part of the primary file on the pricipal I/O device. All LIST variants are of the same format: if no line number is specified, the entire file is listed; only one line is listed if just one line number is specified; and a block of lines is listed if two lines numbers are given.

LIST displays the requested lines exactly as the user typed them in. LISTF displays the lines in an assembler format in which all labels are aligned in one column, all op codes in another, all operands in a third, and all comments in a fourth. This format assumes that the user entered his lines in an assembler free format in which all labels start in the first assembly column of each line and all op codes not proceeded by a label start in the second assembly column of each line. LISTN displays the requested lines exactly as the user typed them in but without line numbers.

All list displays are paged. This paging varies with the logical I/O device currently assigned as the principal I/O device. At the end of a page, the operator is prompted with "?", to which he may respond with "Y" to continue or "N" to stop. Also, the <ESC> key is monitored throughout the creation of the display, and listing may be terminated at any time by simply hitting this key.

Example: LIST 300 400

Result: Lines 300 to 400, inclusive, are listed on the user's principal I/O device.

Example: LIST 450

Result: Line 450 is listed.

Example: LIST

Result: The entire primary file is listed.

Command: LNAME

Brief Description: This command is used to rename the specified local file.

Format: LNAM <fname>

Format: LNAMB <fname>

The local rename command (LNAMe) allows the user to rename any local binary or text file. LNAM renames a local text file, and LNAMB renames a local binary file. In response to this command, ARIAN prompts the user with "NEW NAME?", to which he may respond with the desired new name for the specified file or a <CR> to abort the renaming process.

Example: LNAM FILE1

Result: ARIAN responds with the prompt "NEW NAME?", to which the user may respond with a valid file name or a <CR>. If he responds with a file name, like F1, FILE1 is renamed F1 in the local text file directory; if he responds with a <CR>, the renaming is aborted.

Command: LOAD

Brief Description: LOAD loads a file from disk into memory.

Format: LOAD <fname> <hadr>?

The LOAD command loads a file from disk into memory. Only binary (type 1) and text (type 2) files may be loaded; an error will be given if the file is of some other type.

If a text file is loaded, the disk directory is searched for the specified file name, and, if found, the local text file memory manager is invoked, the local files are compacted, the specified file is loaded into memory, and the new file is made primary. If a file by the same name already exists locally, the user will be asked if he wishes to replace it by the prompt "REPLACE?"; if the user does not respond with a "N", the file is replaced. If the user specifies an address, the new file is loaded into memory at the specified address, but compaction of the secondary files is still done.

If a binary file is loaded, the disk directory is searched for the specified file name, and, if found, the file is loaded at the execution address given by the disk directory. Any address given in the command becomes the load address. An entry is then made in the local binary file directory.

Example: LOAD LASTF

Result: The disk file LASTF is loaded into memory at a location designated by the local memory manager and made primary if it is a text file. If it is binary, it is loaded at its execution address. In both cases, an entry is made in the appropriate local file directory.

Example: LOAD FILEX 4000

Result: The file FILEX is loaded into memory starting at location 4000 hexadecimal. An entry in the appropriate local file directory is made for it.

Command: LSCR

Brief Description: The LSCR command clears (scratches) the specified local file directory.

Format: LSCR

Format: LSCRB

LSCR scratches (deletes) all entries in the specified local file directory. LSCR scratches the local text file directory, and LSCRB scratches the local binary file directory. The files themselves are not affected by this command -- only the directory entries for them.

Command: PRINT

Brief Description: The PRINT command is the same as the LIST command, but the file is displayed on the printer.

Format: PRIN (<lnum> <lnum>?)?

Format: PRINF (<lnum> <lnum>?)?

Format: PRINN (<lnum> <lnum>?)?

The PRINt command displays the selected line or block of lines on the user's printer. Print displays are not paged. PRIN prints the file as it exists, PRINF prints it in assembler format, and PRINN prints it without line numbers.

The display can be interrupted and control returned to ARIAN by typing the <ESC> key.

If only one line number is given, just that line will be printed. If two line numbers are given, that block of lines, inclusive, will be printed. If no line numbers are given, the entire file will be printed.

Command: RCVR

Brief Description: RCVR allows the user to attempt to recover a local text file whose directory entry has been deleted.

Format: RCVR <fname> <hadr>

The recover (RCVR) command is used to recover a text file that has been deleted from the local text file directory. A validity check is done on the file (see FCHK) starting at the address specified, and a local text file directory entry is created with the specified file name and the file boundaries ascertained by the validity check. If the validity check fails, the recovery is aborted with an appropriate error message. If the validity check succeeds, the new file is made primary implicitly through the FILE command.

Example: RCVR OLDFILE 2001

Result: A validity check is started at location 2001 hexadecimal, and if the validity check succeeds a new local text file of the name OLDFILE is entered into the local text file directory.

Command: RESET

Brief Description: The RESET command resets ARIAN. This is roughly equivalent to a warm start.

Format: RESET

RESEt executes a system reset (warm start) of ARIAN. This system reset includes the following operations:

1. The default execution address is set to the system reset entry point.
2. The assembly limits are reset.
3. The system symbol table is cleared.
4. The system tab stops are reset.
5. The default stack pointer is reset for the CONT command.
6. The principal I/O channel is reset to the default port.
7. Control is returned to ARIAN.

Command: RNUM

Brief Description: The RNUM command is used to renumber the lines in the primary file.

Format: RNUM (<lnum> <inc>?)?

RNUM renumbers the primary file. If no arguments are given, the file is numbered to start at line 5 and continue in increments of 5. If just a line number is specified, the file starts at the specified line number and continues in increments of 5; if both line number and increment are specified, these values are used in the renumbering.

Example: RNUM 100 40

Result: The primary file is renumbered, starting with line 100 and incrementing by 40; i.e., the first line will be 100, and succeeding lines will be 140, 180, 220, etc.

Example: RNUM

Result: The primary file is renumbered, starting at 5 and incrementing by 5 (5, 10, 15, ...).

Command: SAVE

Brief Description: The SAVE command is used to save a file onto disk from memory.

Format: SAVE <fname>

Format: SAVEn <fname>

Format: SAVEB <fname> (<hadr> <hadr> <hadr>?)?

Format: SAVEBn <fname> (<hadr> <hadr> <hadr>?)?

Format: SAVET <fname> <hadr>?

Format: SAVETO <fname>

The SAVE command is used to save text and binary files on disk and change the type and execution address of files currently residing on disk.

The SAVE <fname> variant saves the current primary file on disk under the specified name. This name is not required to be the same as the local name of the primary file. Only the primary file is saved by this command.

The SAVEB variant saves a binary file on disk under the specified name. If no address is given, the default assembly limits are used as the file limits. If two address are given, these are used as the file limits, and if a third address is given, it becomes the execution or load address. If no third address is given, the starting address of the file becomes its execution or load address.

SAVET can be used to change the type of a disk file. SAVETO makes the specified file a type 0 (text) file. SAVET makes the specified file a binary file. If <hadr> is specified, its execution address is set to this value; otherwise, its execution address is set to zero.

An optional fifth character for SAVE and sixth character for SAVEB may be specified. This is a digit from 1 to 4; it specifies the number of the disk drive to save the file on. This drive becomes the new logged-in drive.

Example: SAVE NEW

Result: The local primary text file is saved on disk under the name NEW. Note that this name does not necessarily have to be the same as its local name.

Example: SAVE2 NEWF

Result: The same file is saved under the name NEWF on disk drive 2. Drive 2 is now logged in.

Example: SAVEB1 BIN1 2000 2021 5600

Result: The section of memory from 2000 to 2021 hexadecimal, inclusive, is saved on disk as a binary file with execution address 5600 hexadecimal. It is saved on drive 1, and drive 1 becomes the new logged-in drive.

Example: SAVEB BIN2

Result: The section of memory specified by the default assembly limits is saved on the logged-in drive under the name of BIN2.

Command: SETC

Brief Description: This command allows the user to explicitly redirect I/O within ARIAN.

Format: SETC

Format: SETCI <hadr>

Format: SETCO <hadr>

The SETC command allows the user to redirect all ARIAN system I/O through user-defined I/O routines. The parameter to be passed to and from these routines is passed in the A register. The only constraint placed on these routines is that they do not have a net effect on the system stack or any register and they end with a RET (return) instruction.

SETC by itself resets the system I/O to employ the normal system I/O routines. This is the default upon ARIAN initialization.

SETCI (set customized input) allows the user to redefine the system input routine. All input is routed through the subroutine which begins at the specified address immediately after this command is executed.

SETCO (set customized output) allows the user to redefine the system output routine. All output is routed through the subroutine which begins at the specified address immediately after this command is executed.

Example: SETCI 6C00

Result: All input to ARIAN is directed through the subroutine starting at location 6C00 hexadecimal.

Command: SYMT

Brief Description: The symbol table from the last assembly is displayed to the user on the principal I/O device.

Format: SYMT

The SYMT command displays the symbol table produced in the last assembly if the system has not been reset or the table has not been written over. SYMT displays the user program's symbol table. This is a simple listing of the name of the symbol and its value.

Command: TABS

Brief Description: This command allows the user to set, examine, and reset the tab stops.

Format: TABS

Format: TABSL

Format: TABSR

The TABSet command allows the user to set, examine, and reset the ARIAN tab stops. In the set and examine cases, the scale numbering the columns is printed across the page, and the tab stops are denoted by the letter "X".

TABS is used to set the tab stops. After the scale is printed, the user may space or tab (using the previously-defined tab stops) over to the desired tab set location and type an "X". Up to twenty tabs may be set in this way; the process is terminated by typing a carriage return.

TABSL examines (lists) the tab stops as they are currently set. Again, the scale is printed and X's are printed in the columns in which tabs are set.

TABSR resets the tab stops to the standard system definition. Tabs are set at every eighth column.

Command: TERM

Brief Description: This command invokes the terminal subsystem of the UTILITY PROM.

Format: TERM

The TERMinal command invokes the terminal, or inter-system communication, mode of ARIAN. Under this subsystem, the microcomputer becomes relatively transparent to the user and the user's terminal is made to respond like a normal timesharing terminal to an external computer system. Each character typed is sent to a modem and acoustic coupler, and each character received by the modem/coupler is sent to the user's CRT.

The terminal subsystem responds to three subcommands; they are Ctrl-A, Ctrl-L, and Ctrl-R. Ctrl-A invokes the terminal alternate command set; Ctrl-L invokes the download program; Ctrl-R returns control to the calling program (ARIAN).

The terminal alternate command set consists of the following MCS-like commands:

1. C -- redirect I/O to console.
2. F -- disable software echo for full duplex communication.
3. H -- enable software echo for half duplex communication.
4. M -- return to MCS.
5. P -- redirect output to printer.
6. R <hadr> -- run the subroutine located at the specified address. The return in the subroutine transfers control to the terminal mode (not the terminal alternate command set).
7. T <hadr> -- transfer the downloaded code to the specified address. This command transmits a carriage return (<CR>) to the external computer, and the object code downloaded is loaded into memory starting at the specified address; the

addresses given in the code are ignored, and the entire code is loaded sequentially into memory.

8. V -- redirect output to VDM.
9. X -- exit to terminal mode.

Ctrl-L is the download command to the terminal mode subsystem. A carriage return (<CR>) is transmitted to the external computer, and the object code, in INTEL-standard format, is loaded into the microcomputer's memory at the addresses specified in the load blocks.

Ctrl-R simply returns control to the program that called the terminal subsystem. This calling program is usually ARIAN.

Downloading can be interrupted at any time by typing <ESC> on the principal I/O device.

Example: TERM

Result: The terminal subsystem is invoked.

Example: TYPE,OBJECT^L

Result: This the TYPE command to the external computer. OBJECT is an object code file in INTEL-standard format. When the terminal mode reads the Ctrl-L from the user, it transmits a <CR> to the external computer, thereby terminating the TYPE command. The terminal mode then downloads the incoming code into memory and displays it on the CRT.

Command: UTIL

Brief Description: This command invokes the Memory Utility Subsystem.

Format: UTIL

The UTIL command gives the user the ability to examine and modify memory directly. In response to this command, the user is prompted by ARIAN with a slash ("/"). He may then enter the following UTILity subcommands:

1. C <hadr> <hadr> <hadr> -- copy the block of memory defined by the first two addresses to the location of memory starting at the third address. The original block is unchanged unless the third address resides within this block.
2. D <hadr>? <hval>* -- deposit the specified values into memory starting at the given address. If no address is given, the values are deposited starting at the default pointer. When completed, the default pointer points to the next byte to be deposited into.
3. E (<hadr> <hadr>?)? -- examine a specific memory location or a block of memory. The default pointer points to the last location examined. If only one address is given, it and its contents will be displayed followed by a "?", to which the user can respond with a "B" to back up (see the previous address) or "F" to go forward (see the next address). Any other character will exit this mode. If both addresses are specified, this block will be displayed. If no address is specified, the address pointed to by the default pointer will be displayed as though one address was given.
4. F <hadr> <hadr> <hval>* -- find all occurrances of the specified byte string in the memory block bounded by the given addresses.

5. S <hadr>? -- set/display the default pointer. S alone will display the pointer; S with an argument will set the pointer to that value.

6. X -- return to ARIAN command mode.

These commands are a subset of the MCS V1.6 command set, and more details as to their usage may be found in the "MCS User's Manual".

Example: UTIL

Result: The utility subsystem is invoked.

Example: E 0 FF

Result: A block examine is done of locations 0 to FF hexadecimal. The output is paged.

Example: C 0 FF 4000

Result: Block 0 to FF hexadecimal, inclusive, is copied to start at location 4000 hexadecimal.

Example: D 2000 0 1 2 3

Result: The values 0, 1, 2, and 3 are deposited into memory locations 2000 to 2003 hexadecimal. The default pointer is now pointing to location 2004.

Example: D 4

Result: The value 4 is deposited into memory location 2004 hexadecimal, and the default pointer now points to location 2005 hexadecimal.

Example: X

Result: UTIL is exited and control is returned to ARIAN command mode.

The ARIAN II BUFFERS and JUMP TABLE

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

16 July 1979

APPENDIX II

The ARIAN II BUFFERS and JUMP TABLE

Contents

Title	Page
The ARIAN II Buffers	1
Disk Directory	1
Local Text File Directory	2
Local Binary File Directory	2
Customized Command Table	3
Breakpoint Table	4
Assembler Symbol Table	4
Register Save Area	5
Executive Parser Buffers	6
Selected ARIAN II Buffers	7
The ARIAN II Jump Table	9
The ARIAN II Jump Table Functions	9
The ARIAN II Restart Entries	11

The ARIAN II BUFFERS and JUMP TABLE

The ARIAN II Buffers

The following is a description of several of the key buffer areas within the ARIAN II Scratchpad RAM Area. These descriptions include information as to the address of the start of the buffers, the format of the information in the buffers, and the size of the buffers.

Name of Directory/Table: Disk Directory

Entry Format:
16 bytes
- File Name: 8 bytes
- Disk Address (low order, high order): 2 bytes
- Number of Blocks (low order, high order): 2 bytes
- File Type: 1 byte
- Execution Address (low order, high order): 2 bytes
- Unused: 1 byte

Comments:

All elements of an entry except the File Name are in binary. The File Name is in ASCII, with blank fill on the right.

The buffer is large enough to contain 64 entries. The size of the buffer is 1K bytes.

Each entry is of the form described above. If no entry exists at a particular location in the directory, then the first byte of the File Name is a space.

The Starting Address of the Disk Directory is F300.

The ARIAN II BUFFERS and JUMP TABLE

Name of Directory/Table: Local Text File Directory

Entry Format:
16 bytes
- File Name: 8 bytes
- Beginning of File (BOF) Pointer (low order, high order): 2 bytes
- End of File (EOF) Pointer (low order, high order): 2 bytes
- Maximum Line Number (4 ASCII Characters, MSD First): 4 bytes

Comments:

The File Name is in ASCII, with blank fill on the right. The BOF and EOF Pointers are in binary, and the Maximum Line Number is in ASCII.

The buffer is large enough to contain 10 entries. The size of the buffer is 160 bytes.

If the first byte of a File Name is a binary 0, then there is no directory entry at this particular location.

The Starting Address of the Local Text File Directory is F090.

Name of Directory/Table: Local Binary File Directory

Entry Format:
12 bytes
- File Name: 8 bytes
- Starting Address (low order, high order): 2 bytes
- Ending Address (low order, high order): 2 bytes

Comments:

The File Name is in ASCII, with blank fill on the right. The Starting and Ending Addresses are in binary.

The buffer is large enough to contain 10 entries. The size of the buffer is 120 bytes.

If the first byte of an entry in the directory is a space, then there is no entry at this particular location.

The Starting Address of the Local Binary File Directory is F018

Name of Directory/Table: Customized Command Table

Entry Format:

6 bytes

Command Name: 4 bytes

Execution Address (low order, high order): 2 bytes

Comments:

The Command Name is in ASCII, with blank fill on the right. The Execution Address is in binary.

The buffer is large enough to contain 20 entries. The size of the buffer is 121 bytes.

The first byte of the table contains a count of the number of entries, and the rest of the table follows the entry format.

The Starting Address of the Customized Command Table is F130.

The ARIAN II BUFFERS and JUMP TABLE

Name of Directory/Table: Breakpoint Table

Entry Format:
3 bytes
- Breakpoint Address (high order, low order): 2 bytes
- Replaced Byte: 1 byte

Comments:
- All values are in binary.
- Note the reversed order of the address.
- The buffer is large enough to contain 8 entries. The size of the buffer is 24 bytes.
- The presence of an entry in the Breakpoint Table is determined by examining the address at an entry location. If this address is binary 0, then no entry exists here. Note that this prohibits setting a breakpoint at location 0.
- The Starting Address of the Breakpoint Table is F1A9.

Name of Directory/Table: Assembler Symbol Table

Entry Format:
8 bytes
- Symbol Name: 6 bytes
- Symbol Address (high order, low order): 2 bytes

Comments:

The Symbol Name is in ASCII, with zero fill on the right. The Symbol Address is in binary.

The buffer is large enough to contain at least 384 entries. The size of the buffer is flexible, with room for upward growth, but the allocated size is 3172 bytes.

Note the reversed order of the address.

The number of labels in the Assembler Symbol Table is contained in the buffer NOLA (low order, high order). This buffer is 2 bytes long, and its Starting Address is F275.

The Starting Address of the Assembler Symbol Table is F300.

Name of Directory/Table: Register Save Area

Entry Format:
1 or 2 bytes

The contents of the registers are stored in sequential bytes in this buffer.

Comments:

The sequence of storage of the registers in this area is E, D, C, B, Flags, A, IX (low order, high order), IY (low order, high order), L′, H′, E′, D′, C′, B′, Flags′, A′, L, H, SP (low order, high order), and PC (low order, high order).

The Starting Address of the Register Save Area is F1C1.

The ARIAN II BUFFERS and JUMP TABLE

Name of Directory/Table: Executive Parser Buffers

Entry Format:

Buffer	Length	Address
ABUF	16 bytes	F1FD
BBUF	8 bytes	F20D
CBUF	8 bytes	F1F5
FBUF	8 bytes	F1E9
SBUF1	40 bytes	F213
SBUF2	40 bytes	F23B

Comments:

These are the buffers whose contents are filled by the ARIAN II Executive Parser.

ABUF, CBUF, and FBUF are ASCII buffers, with blank fill on the right. SBUF1 and SBUF2 are ASCII buffers; if they contain valid information, the first character in SBUF1 or SBUF2 is a double quote, and the last character is a carriage return. BBUF contains binary values.

ABUF and BBUF may contain up to three entries. For ABUF, each entry is four bytes long; for BBUF, each entry is two bytes long (low order, high order).

The special characters appended to the command name are stored in CBUF in the 5th, 6th, and 7th bytes (SPCHR1, SPCHR2, and SPCHR3, resp.). The address of SPCHR1 is F1F9.

The ARIAN II BUFFERS and JUMP TABLE

Selected ARIAN II Buffers

Name	Address	Size	Use/Function
1. Workspace Buffers			
WSPE	F002	2	Workspace Pointer (End)
WSPS	F000	2	Workspace Pointer (Start)
2. Disk Drive Buffers			
CDRIV	F1DA	1	Command Drive Number Buffer
DRIVEN	F00A	1	Logged-In Drive Number Buffer
3. Default Execution Address			
EXADR	F00B	2	Default Execution Address
4. Paging Buffers			
LPCNT	F015	1	Line Paging Count (Number of lines left on page)
NL	F265	1	Number of Lines on a Page
5. File Search Buffers			
FEF	F1F4	1	Free Entry Found Flag (use w/XFSEA)
FREAD	F1F2	2	Free Entry in Directory Pointer (use w/XFSEA)
6. Input/Output Buffers			
IBUF	FE00	160	Input Line Buffer (referenced by ZIBF)
OBUF	FDEC	141	Output Line Buffer (used by Assembler)
7. Assembler Buffers			
APND	F1E3	2	Assembler Line Pointer (Pts to current line)
ASMEN	F26C	2	Assembly End Address

The ARIAN II BUFFERS and JUMP TABLE

ASMRET	F26F	2	Assembler Return Address
ASMST	F26A	2	Assembly Start Address
ASPC	F268	2	Assembler Location Counter
NOLA	F275	2	Number of Labels in Symbol Table
PASI	F271	1	Assembler Pass Indicator (1 or 2)

The ARIAN II BUFFERS and JUMP TABLE

The ARIAN II Jump Table

The following is a listing of the entry addresses of the routines accessable via the ARIAN II Jump Table.

The ARIAN II Jump Table Functions

Name	Address	Regs Affected	Function
XCKA11	40	A,HL	Check for the presence of the 1st argument in BBUF/ABUF. Return w/zero set if no argument, not zero if argument.
XCKA21	43	A,HL	Check for the presence of the 2nd argument in BBUF/ABUF. Return w/zero set if no argument, not zero if argument.
XCKFN1	46	A,HL	Check for the presence of an argument in FBUF. Return w/zero set if no argument, not zero if argument.
XDCOM	69	-All-	Execute the ARIAN II Disk Communication routine; this is the same as FDOS' DCOM, but control is returned to ARIAN II upon error. Input register parameters are: HL = Starting Disk Address DE = Starting RAM Address B = Command (0=write, 1=read, 2=verify) C = Unit Number A = Number of Blocks
XDSCAN	49	-All-	Load and scan directory of Loggen-In Drive

			for file whose name is in FBUF. Return w/ zero set if found, not zero if not found. If found, HL points to 1st byte of entry in directory for this file.
XFILE	4C	-All-	Execute FILE command. This is useful in creating a new primary file by loading FBUF with the new file name.
XFIND	4F	-All-	Find a line in the primary file whose number is greater than or equal to the line number (stored as normalized ASCII) in the first element of ABUF. On return, HL points to the 1st byte (character count) of the line found.
XFSEA	5E	-All-	Search the local text file directory for the file whose name is in FBUF. Upon exit, return w/not zero & HL pointing to entry in directory if found. If not found, return w/ zero set. Also, if not found, the Free Entry Flag (FEF) is not zero if there is room in the directory for another file and the FREAD buffer points to an available entry location in the directory.
XINK	72	A	Poll all channels for an <ESC> and return to the Executive if one was typed
XLINE	52	-All-	Enter line contained in IBUF into the primary file. Line must be of ARIAN II format. The assembler-defined routine ZLIN creates it properly. A line number (not necessarily normalized) must begin the line.
XLOAD	55	-All-	Execute the LOAD command. This is useful for loading and making primary a disk file on the Logged-In Drive. The file name is in FBUF.

The ARIAN II BUFFERS and JUMP TABLE

XMESS	5B	-All-	Print string ending in <CR> pointed to by HL on the principal I/O device. Return to the ARIAN II Executive when done. Prefix the string (as printed) w/<CR> '$' and a blank.
XPARSE	6C	-All-	Enter the Executive Parser and parse the line contained in IBUF.
XREAD	6F	HL,A	Enter the input line editor and permit the user to enter a line into IBUF. On exit, HL point to the first char of the line (after the char count).
XRNUM	61	-All-	Execute the RNUM command. The Primary File is renumbered. ABUF contains the non-default parameters as normalized ASCII characters.
XSAVE	58	-All-	Execute the SAVE command. This is useful for creating binary disk files or saving the Primary File (text). Instructions to this routine are passed in the appropriate Executive Parser Buffers, especially SPCHR1.

The ARIAN II Restart Entries

Name	Address	Restart	Function
XARIAN	0	0	ARIAN II Cold Start
XBRKP	8	1	Breakpoint Entry Address
XEOR	66	-	Executive Entry Address
XRESET	4	-	ARIAN II Warm Start
XTRAP	38	7	Memory Trap and Executive Entry

SUMMARY of the ARIAN II ASSEMBLER

by
Richard L. Conn

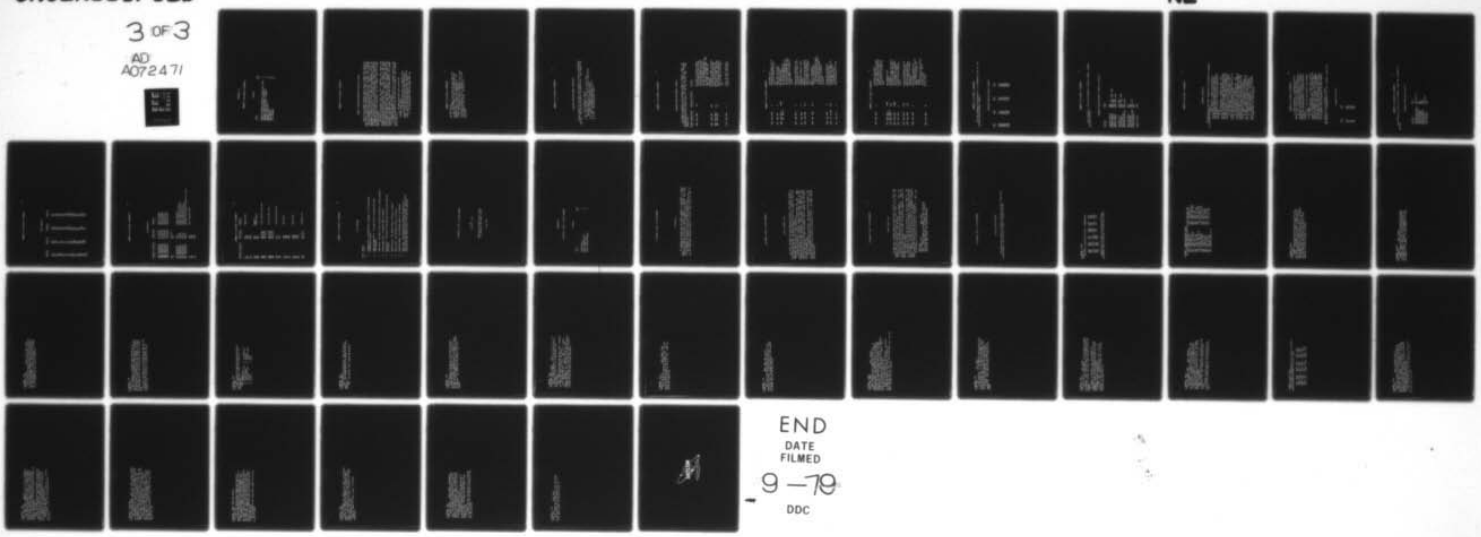
USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

15 July 1979

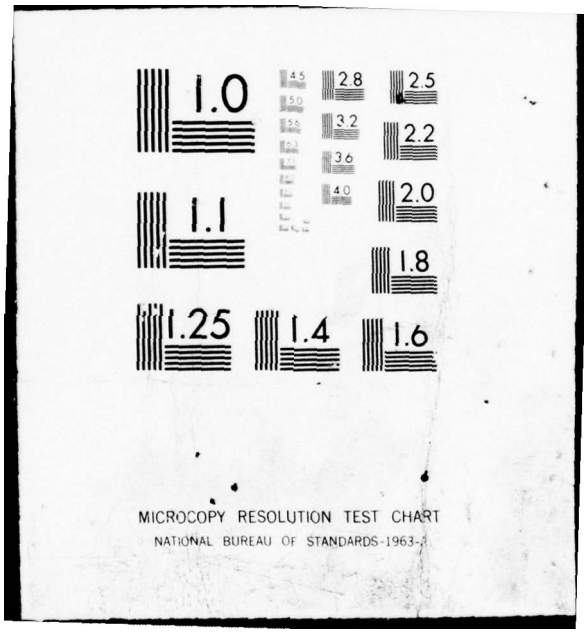
AD-A072 471 ARMY SATELLITE COMMUNICATIONS AGENCY FORT MONMOUTH NJ F/G 9/2
PROJECT ARIES. USER'S MANUALS FOR ARIAN II AND ASSOCIATED SUBSY--ETC(U)
JUL 79 R L CONN

UNCLASSIFIED NL

3 OF 3
AD
A072471



END
DATE
FILMED
9-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX III

SUMMARY of the ARIAN II ASSEMBLER

Contents

Title	Page
ARIAN II Standard Assembler Program Line Format	1
Assembler Reserved Symbols (Registers)	3
Assembler Reserved Symbols (Z-Names)	4
Assembler Pseudo Ops	9
8080 Mnemonics	12
Z80 Mnemonics	13
Error Messages	15

SUMMARY of the ARIAN II ASSEMBLER

ARIAN II Standard Assembler Program Line Format

Each line of assembly language source code is divided into from one to four fields. These fields are the name (or label) field, the operation field, the operand field, and the comment field. The name field contains the label assigned to that particular line; this label is a string of alphanumeric characters, the first of which must be alphabetic, and it may optionally end with a colon (:). The operation field contains the mnemonic or pseudo-op which defines the function. The operand field contains the operands required by the instruction, and the comment field is simply descriptive text.

The source line starts with the first assembler line column and extends to the terminating carriage return. This column is the sixth legible character of the line; it is the character following the space which follows the four-digit line number. Hence, an input line consists of a four-digit line number, a space, and the assembly language source code followed by a carriage return. Blank lines are permitted, as are comment lines (lines which are only comment).

The ARIAN II assembler is free format, but a "standard" format for the source code is defined to work with the F and X options on some commands. This format is:

1. The name field must start in the first assembler line column.
2. If the name field exists, the operation field starts one space after the colon or end of the name field. If the name field does not exist, the operation field starts in the second assembler line column.
3. The operand field starts one space after the

operation field.

4. The comment field starts one space after the operand field if there is one or one space after the operation field if there is no operand field. The comment field must start with a semicolon.

5. If the entire line is a comment, it must start with either an asterisk (*) or a semicolon (;) in the first assembler line column. The asterisk is preferred.

Assembler Reserved Symbols (Registers)

Certain symbols are reserved by the assembler to refer to the 8080/Z80 registers. They may only be used in the operand field. These symbols are:

1. A -- the accumulator; value 7
2. B, C, D, E, H, and L -- the B, C, D, E, H, and L registers; values are 0, 1, 2, 3, 4, and 5, respectively
3. M -- memory (pointed to by H&L); value 6
4. P, PSW -- the program status word; value 6
5. S, SP -- the stack pointer; value 6

SUMMARY of the ARIAN II ASSEMBLER

Assembler Reserved Symbols (Z-Names)

The ARIAN II assembler also reserves a number of symbolic names, all of which begin with the letter "Z". These symbols are used to reference subroutines and buffers within ARIAN II and the UTILITY PROM. These symbols are:

Symbol	Sample Usage	Registers Affected	Function
ZARG	CALL ZARG	-ALL-	the ARIAN II Executive Parser which parses the line in IBUF as described before; it may be used in conjunction with ZLIN to input a command and parse it for the user's program; ABUF, BBUF, CBUF, S1BUF, and S2BUF are generated
ZBBF	LHLD ZBBF	N/A	the address of BBUF, the binary buffer created by the Parser
ZBLK	CALL ZBLK	A	output <SP> to principal I/O
ZBOF	LHLD ZBOF	N/A	the address of BOFP, the buffer which contains the address of the first byte of the primary file
ZCAH	CALL ZCAH	A	convert the ASCII hexadecimal character in A to its binary

SUMMARY of the ARIAN II ASSEMBLER

				equivalent in A
ZCHA	CALL	ZCHA	A	convert the low nybble of A to its ASCII hexadecimal equiv in A
ZCR	CALL	ZCR	A	output <CR> <LF> to the principal I/O channel
ZCRL	CALL	ZCRL	-NONE-	Call Relative Long
	DW	ADR-$-2		The two bytes pointed to by the return address make up the relative displacement from the next instruction. The range of displacement is -32768 to +32767.
ZDOT	CALL	ZDOT	A	display the A register as up to three decimal digits, left space fill
ZEN	CALL	ZEN	A	exchange the nybbles of the A register
ZEOF	LHLD	ZEOF	N/A	address of the two-byte buffer which contains the address of the last byte of the primary file
ZEOR	JMP	ZEOR	N/A	address of the ARIAN II Executive reentry point -- this entry point can be used to make a clean return to ARIAN II and preserve the environment at the time of program execution
ZHOT	CALL	ZHOT	A	display the A register as two hexadecimal digits
ZIBF	LXI	H,ZIBF	N/A	the address of the input line buffer IBUF; this buffer is generated by ZLIN
ZIN	CALL	ZIN	A	input one character from the

```
                                          principal I/O channel
ZINK  CALL ZINK       A                   poll all channels for an <ESC>;
                                           if typed, control returned to
                                           ARIAN II Executive; return if
                                           no character or character not
                                           <ESC>

ZJRL  CALL ZJRL       -NONE-              Jump Relative Long
      DW   ADR-$-2                         (see ZCRL)
ZLIN  CALL ZLIN       A,HL                the ARIAN II input line editor;
                                           HL points to IBUF upon exit
ZOUT  CALL ZOUT       -NONE-              output character in A register
                                           on principal I/O channel
ZPHL  CALL ZPHL       A                   print HL as four hexadecimal
                                           digits

ZPPH  CALL ZPPH       A,HL                print string pointed to by HL
                                           ending in <CR> on printer
ZPRH  CALL ZPRH       A,HL                print string pointed to by HL
                                           ending in <CR> on principal
                                           I/O channel
ZPRR  CALL ZPRR       A                   print string pointed to by
                                           return address ending in <null>
                                           (binary 0) on principal I/O
                                           channel
                                            This string would be specified
                                           as "ASC 'string'   DB 0"
                                           after the call.
ZSHD  CALL ZSHD       BC                  BC = HL - DE
```

SUMMARY of the ARIAN II ASSEMBLER

For quick reference, the following is an alphabetized list of the Assembler Reserved Symbols (Z-Names).

Assembler Reserved Symbols (Z-Names)

Name	Name	Name	Name
ZARG	ZCR	ZHOT	ZOUT
ZBBF	ZCRL	ZIBF	ZPHL
ZBLK	ZDOT	ZIN	ZPPH
ZBOF	ZEN	ZINK	ZPRH
ZCAH	ZEOF	ZJRL	ZPRR
ZCHA	ZEOR	ZLIN	ZSHD

SUMMARY of the ARIAN II ASSEMBLER

The following is a list of the Assembler Reserved Symbols (Z-Names) grouped by function.

Assembler Reserved Symbols (Z-Names) by Function

Function	Names
Executive Parser	ZARG
Internal Buffers	ZBBF, ZBOF, ZEOF, ZIBF
Specific Output	ZBLK, ZCR
Register Output	ZDOT, ZHOT, ZPHL
Character I/O	ZIN, ZOUT
String Output	ZPPH, ZPRH, ZPRR
Input Line Editor	ZLIN
ASCII/HEX Conversion	ZCAH, ZCHA
Nybble Exchange	ZEN
Interrupt Poll	ZINK
Branch Relative Long	ZCRL, ZJRL
16-bit Arithmetic	ZSHD
ARIAN II Entry	ZEOR

SUMMARY of the ARIAN II ASSEMBLER

Assembler Pseudo Ops

The following is a list and a description of the pseudo-ops recognized by the ARIAN II Assembler.

1. ASC '<string>' -- ASCII string definition. This pseudo-op loads consecutive memory locations starting at the current value of the location counter with the ASCII values of the characters specified in the string.
2. DB <expression> -- define one byte. This instruction evaluates the specified operand and loads one 8-bit value into the memory location pointed to by the location counter. If the number is evaluated into a 16-bit quantity, only the low-order byte is loaded.
3. DS <expression> -- define storage. This reserves the specified number of bytes starting at the current value of the location counter.
4. DW <expression> -- define one word. This instruction evaluates the specified operand, producing a 16-bit value which it loads into memory (low order, high order) at the memory location pointed to by the location counter and the next memory location.
5. END -- end the assembly. This statement is not required; assembly will stop when the end of the file is reached or this statement is encountered.
6. <label> EQU <expression> -- the specified label is assigned the computed value of the operand; the computed value is a 16-bit quantity.

7. EXEC <expression> -- the default execution address is set to the value of the expression.

8. LST -- turn on the listing of the assembly of the program on the principal I/O device if it is not already on. This can be used in conjunction with NLST to list only selected portions of a program during an assembly.

9. NLST -- turn off the listing of the assembly of the program.

10. ORG <expression> -- set the origin (location counter) to the specified value. This instruction also resets the assembly limits and the location in memory at which the object code is loaded. If an ORG appears more than one time in the program, the limits set by the last ORG and the end of the assembly are reflected in the assembly limits displayed by the LDIRB command.

All pseudo-ops may be preceeded by a label.
The following is an alphabetized list of the Assembler Pseudo Ops.

Assembler Pseudo Ops

Name	Name
ASC	EQU
DB	EXEC
DS	LST
DW	NLST
END	ORG

SUMMARY of the ARIAN II ASSEMBLER

The following is a list of the Assembler Pseudo Ops organized by function.

Assembler Pseudo Ops by Function

Function	Name
Constant Definition	ASC, DB, DW
Storage Reservation	DS
Label Assignment	EQU
Location Counter	ORG
Execution Address	EXEC
List Control	LST, NLST
End of Assembly	END

SUMMARY of the ARIAN II ASSEMBLER

8080 Mnemonics

Mnemonic	Mnemonic	Mnemonic	Mnemonic
ACI	ADC	ADD	ADI
ANA	ANI	CALL	CC
CM	CMA	CMP	CNC
CNZ	CP	CPE	CPI
CPO	CZ	DAA	DAD
DCR	DCX	DI	EI
HLT	IN	INR	INX
JC	JM	JMP	JNC
JNZ	JP	JPE	JPO
JZ	LDA	LDAX	LHLD
LXI	MVI	MOV	NOP
ORA	ORI	OUT	PCHL
POP	PUSH	RAL	RAR
RC	RET	RLC	RM
RNC	RNZ	RP	RPE
RPO	RRC	RST	RZ
SBB	SBI	SHLD	SPHL
STA	STAX	STC	SUB
SUI	XCHG	XRA	XRI
XTHL			

SUMMARY of the ARIAN II ASSEMBLER

Z80 Mnemonics

Mnemonic & Operand	ZILOG Equiv	Comments
SSPD <expression>	LD (nn),SP	store SP direct
LSPD <expression>	LD SP,(nn)	load SP direct
SBCD <expression>	LD (nn),BC	store BC direct
LBCD <expression>	LD BC,(nn)	load BC direct
SDED <expression>	LD (nn),DE	store DE direct
LDED <expression>	LD DE,(nn)	load DE direct
EXA	EX AF,AF'	
EXX	EXX	
BR <expression>	JR n	branch relative
BC <expression>	JR C,n	branch relative on carry
BNC <expression>	JR NC,n	branch relative on no carry
BZ <expression>	JR Z,n	branch relative on zero
BNZ <expression>	JR NZ,n	branch relative on no zero
DBJ <expression>	DJNZ n	decrement B and branch relative on no zero
LD	LDD	Block Transfer Group
LDR	LDDR	
LI	LDI	
LIR	LDIR	

SUMMARY of the ARIAN II ASSEMBLER

Mnemonic & Operand	ZILOG Equiv	Comments
CD	CPD	Compare Group
CDR	CPDR	
CI	CPI	
CIR	CPIR	
NEG	NEG	Negate A
RLD	RLD	Nybble Rotate
RRD	RRD	
SHB	SBC HL,BC	16-bit subtract with carry
SHD	SBC HL,DE	
SHS	SBC HL,SP	
AHB	ADC HL,BC	16-bit add with carry
AHD	ADC HL,DE	
AHS	ADC HL,SP	
IM0	IM 0	Interrupt Mode Control
IM1	IM 1	
IM2	IM 2	
ID	IND	Input Group
IDR	INDR	
II	INI	
IIR	INIR	
OD	OUTD	Output Group
ODR	OTDR	
OI	OUTI	
OIR	OTIR	
CIN	IN A,(C)	C-Register I/O
COT	OUT (C),A	

SUMMARY of the ARIAN II ASSEMBLER

Error Messages

Error	Meaning
A	Argument error. The instruction's argument is of the wrong type or generally incorrect.
D	Duplicate label error. The label of this instruction has been used elsewhere.
L	Label error. The label of this instrucion contains an invalid character.
M	Missing label error. A required label is missing.
O	Opcode error. The opcode in the operation field of this instrucion is invalid
R	Register error. The register name is missing or invalid.
S	Syntax error. The instruction syntax is incorrect.
U	Undefined symbol. The referenced symbol is not defined.
V	Value error. The computed value cannot be represented as a 16-bit value or the instruction has a syntax error. Also, an 8-bit value may have been required and a 16-bit value was generated.

The ARIAN II LEVEL 3 SYSTEM COMMANDS

by
Richard L. Conn

USA Satellite Communications Agency
Fort Monmouth, New Jersey 07703

15 July 1979

APPENDIX IV

The ARIAN II LEVEL 3 SYSTEM COMMANDS

Contents

Title	Page
Introduction	1
Level 3 Command Files	2
Level 3 HELP Files	3
Level 3 System Commands	4

The ARIAN II LEVEL 3 SYSTEM COMMANDS

Introduction

This appendix is perhaps the most dynamic appendix in the ARIAN II User's Manual. The appendix on the Level 3 System Commands is designed to present introductory material on the Level 3 System Commands in the form of their HELP files and general information on Level 3 commands and HELP files.

This appendix is organized as follows: the introductory information is on the first three pages and the listings of the help files start on page 4.

Level 3 Command Files

All Level 3 Command Files are structured similarly. They reside as Type 1, or binary, files on disk. As noted before, such files are defined solely by their directory entries. All Level 3 Command Files have names of the form 'NAME.CMD'; they all end with a '.CMD' extension.

The Execution Address of a Level 3 File is determined by the user. Most system files execute in the transient program area, which starts at BC00 hexadecimal and ends at BFFF hexadecimal. Some of these files, however, are too large to reside in this area, and they reside elsewhere. The user must take care that the execution of these "non-standard" Level 3 Commands will not damage the memory-resident files.

The Level 3 Command Files are binary. The only restriction on these files is that their execution must start at their load address. A return can be effected to ARIAN II by a simple RET instruction or one of the other methods discussed earlier.

Level 3 HELP Files

The HELP Level 3 Command supports a special form of type 0, or text, file called a HELP File. HELP Files are text files whose names are of the form 'NAME.HLP'; they all end with a '.HLP' extension.

All HELP Files are simple text files created under ARIAN II. Their only restriction is that they may be no larger than 1K bytes (4 blocks). They are intended to provide a summary of the use of a particular command or feature of the user's ARIAN II system.

The contents of a HELP File are formatted in a particular fashion. The first line of the file gives the name of the command. The second line contains the word "USAGE: " followed by the various formats of the command and the word "ARGS: " followed by a description of the forms of the permissible arguments. The following lines contain a brief description of the command.

The following is an example of the text file structure of a HELP File.

```
0005 XEDIT
0010 USAGE: XEDIT                ARGS: <LNUM>
0015    'XEDIT' INVOKES THE EXTENDED CRT INTRA-LINE EDITOR
0020 ON THE SPECIFIED LINE OF THE PRIMARY FILE.  REFER TO
0025 "XEDIT USER'S MANUAL" FOR FURTHER INFORMATION.
```

The ARIAN II LEVEL 3 SYSTEM COMMANDS

Level 3 System Commands

The following are listings of the HELP Files which describe all the Level 3 System Commands. These files are arranged in alphabetical order.

```
++ COMMAND:  ARIAN
 USAGE: N/A
    THE ARIAN II EXECUTIVE, OR LEVEL 2, COMMANDS ARE:

        APND    ASSM    BREK    CMND    CONT    CUST
        DDEL    DDIR    DEL     DNAME   EDIT    EXEC
        EXIT    FCHK    FILE    FIND    INS     LDEL
        LDIR    LIST    LNAME   LOAD    LSCR    PRINT
        RCVR    RESET   RNUM    SAVE    SETC    SYMT
        TABS    TERM    UTIL

    INFORMATION ON THE ARIAN II OPERATING AND SOFTWARE
 DEVELOPMENT SYSTEM IS AVAILABLE IN "ARIAN USER'S MANUAL".
```

```
++ COMMAND:  ASSEMBLER-DEFINED SYMBOLS
    THE 24 ASSEMBLER-DEFINED SYMBOLS AND THEIR FUNCTIONS ARE:
        ZEOR - ARIAN EXEC RET           ZINK - INTERRUPT RET
        ZIN - INPUT CHAR IN A           ZOUT - OUTPUT CHAR IN A
        ZLIN - INPUT LINE EDITOR        ZARG - INPUT LINE PARSE
        ZCR - PRINT <CR> <LF>           ZBLK - PRINT <SP>
        ZDOT - PRINT A AS DEC           ZHOT - PRINT A AS HEX
        ZPRR - PRINT LINE PTED TO       ZPRH - PRINT LINE PTED
          BY RET ADR; ENDS IN 0           TO BY HL; ENDS IN CR
        ZPPH - ZPRH ON PRINTER          ZPHL - PRINT HL AS HEX
        ZBOF - START OF FILE            ZEOF - END OF FILE
        ZBBF - BINARY BUFFER            ZIBF - INPUT BUFFER
        ZCHA - HEX TO ASCII             ZCAH - ASCII TO HEX
        ZJRL - JUMP REL LONG            ZCRL - CALL REL LONG
        ZEN - EXCHANGE NYBBLES OF A
```

```
++ COMMAND:  BACKUP
 USAGE: BACKUP
     'BACKUP' COPIES ONE DISK ONTO ANOTHER USING ONLY ONE
 DISK DRIVE.  IN RESPONSE TO THE 'BACKUP' COMMAND, THE USER
 WILL BE PROMPTED WITH "PRIMARY DISK?".  AT THIS POINT HE
 SHOULD LOAD THE DISK TO BE COPIED INTO DRIVE 1.  IN RESPONSE
 TO "SECONDARY DISK?" HE SHOULD LOAD THE DISK TO COPY TO.
 AFTER HE LOADS EACH DISK, HE MAY STRIKE ANY KEY ON THE
 PRINCIPAL I/O DEVICE TO ENGAGE THE PROCESS.
```

```
++ COMMAND:  COMPACT
 USAGE: COMPACT          ARGS: <DNUM>?
    'COMPACT' COMPACTS THE FILES ON THE SPECIFIED DISK DRIVE.
 IF NO DRIVE NUMBER IS SPECIFIED, THE FILES ARE COMPACTED ON
 THE LOGGED-IN DRIVE.  THE NUMBER OF THE LOGGED-IN DRIVE IS
 NOT CHANGED BY THIS COMMAND.  THE LOCAL WORKSPACE IS CLEARED.
```

++ COMMAND: COPY
USAGE: COPY, COPYB ARGS: <LNUM> <LNUM> <LNUM>
'COPY' COPIES THE BLOCK OF LINES, INCLUSIVE, SPECIFIED BY THE FIRST TWO LINE NUMBERS AFTER THE LINE SPECIFIED BY THE THIRD LINE NUMBER. 'COPYB' COPIES THE BLOCK BEFORE THE LINE SPECIFIED BY THE THIRD LINE NUMBER. THE PRIMARY FILE IS RENUMBERED AFTER THE COPY IS COMPLETED.

++ COMMAND: CREF
USAGE: CREF
'CREF' PRODUCES A LINE-NUMBER CROSS REFERENCE OF ALL LABELS IN THE ASSEMBLY-LANGUAGE PRIMARY FILE ON THE PRINTER. THE PRINTER MUST BE ENGAGED TO USE THIS COMMAND. THE OUTPUT IS PAGED, AND THE USER IS PROMPTED WITH "HEADER:", TO WHICH HE MAY RESPOND WITH A TITLE TO BE PLACED AT THE TOP OF EACH PAGE.

THE FIRST LINE NUMBER IN THE LISTING IS THE LINE THE LABEL IS DEFINED IN, AND THE REST OF THE LINE NUMBERS ARE THE REFERENCE LINES. THE LISTING IS ALPHABETIZED BY LABEL.

```
++ COMMAND:  DBUG
 USAGE: DBUG
     'DBUG' INVOKES THE 8080/Z80 SIMULATION SUBSYSTEM.
 REFER TO "DBUG USER'S MANUAL" FOR DETAILS OF THE
 SUBCOMMAND SET.
     DBUG COMMANDS ARE:
        A - ACTIVATE I/O PORT    B - SET BREAKPOINT
        C - CLEAR & TRACE        E - EXECUTE
        I - SET PREDEF IPORT     R - REGISTER TOGGLE
        S - STATUS               T - TRACE
        U - UTILITY              X - EXIT
```

```
++ COMMAND:  DCPY
 USAGE: DCPY, DCPYV
    'DCPY' COPIES THE DISK ON DRIVE 1 ONTO THE DISK ON
 DRIVE 2.  THIS COMMAND DESTROYS THE CONTENT OF THE ARIAN
 WORKSPACE.
     'DCPY' JUST COPIES; 'DCPYV' COPIES AND VERIFIES.
```

++ COMMAND: DLOG
USAGE: DLOG
'DLOG' PRODUCES A PRINTED LOG OF THE DISK DIRECTORIES ON EITHER JUST DRIVE 1 OR DRIVES 1 AND 2. THIS LOG IS SUITABLE FOR EXTERNAL DOCUMENTATION. IN RESPONSE TO 'DLOG', THE USER IS ASKED A NUMBER OF QUESTIONS, TO WHICH HE RESPONDS AS INDICATED. THE PRINTER MUST BE ENGAGED TO USE THIS COMMAND.

```
++ COMMAND:  DSET
 USAGE: DSET, DSETI, DSETO      ARGS: ( V ! M ! P ! C ! D )?
    'DSET' IS USED TO REDIRECT I/O TO THE SPECIFIED DEVICE.
 IF NO DEVICE IS SPECIFIED, I/O IS REDIRECTED TO THE
 DEFAULT DEVICE.
    'DSET' REDIRECTS BOTH INPUT AND OUTPUT, 'DSETI' REDIRECTS
 JUST INPUT, AND 'DSETO' REDIRECTS JUST OUTPUT.
    VALID DEVICE NAMES ARE:  (1) V -- VDM, (2) M -- MODEM,
 (3) P -- PRINTER, (4) C -- CONSOLE, AND (5) D -- DUAL
 [CONSOLE AND MODEM].  ONLY THE FIRST LETTER NEED BE SPECIFIED.
 V AFFECTS ONLY OUTPUT; M, P, AND C AFFECT INPUT OR OUTPUT;
 D AFFECTS EITHER JUST OUTPUT OR INPUT AND OUTPUT.
```

++ COMMAND: FCPY
USAGE: FCPY, FCPYB ARGS: <FNAME> <LNUM>
'FCPY' COPIES THE SPECIFIED SECONDARY LOCAL TEXT FILE AFTER THE SPECIFIED LINE IN THE PRIMARY FILE. 'FCPYB' COPIES THIS FILE BEFORE THE SPECIFIED LINE. <FNAME> MAY NOT BE THE PRIMARY FILE.

```
++ COMMAND:  FILL
 USAGE: FILL              ARGS: <ADR> <ADR> <BYTE>?
     'FILL' FILLS THE SPECIFIED AREA OF MEMORY WITH THE
 SPECIFIED BYTE VALUE.  IF NO BYTE VALUE IS GIVEN, 0 (ZERO)
 IS THE FILL BYTE.  ANY ADDRESS RANGE EXCEPT THAT STARTING
 WITH MEMORY LOCATION ZERO MAY BE GIVEN.
```

++ COMMAND: FORMAT
 USAGE: FORM, FORMV   ARGS: <PAGENUM>?
    'FORM' INVOKES THE TEXT FORMATTING SYSTEM (TFS). IT
 FORMATS THE PRIMARY FILE ACCORDING TO THE TFS COMMANDS
 CONTAINED WITHIN IT AND PRINTS THE RESULT ON THE PRINTER.
 REFER TO "TEXT FORMATTING SYSTEM USER'S MANUAL" FOR MORE
 INFORMATION ON TFS. IF <PAGENUM> IS SPECIFIED, THE FORMATTING
 STARTS AT THE SPECIFIED PAGE; OTHERWISE, IT STARTS AT PAGE 1
 OR THE PAGE SPECIFIED WITHIN THE FILE.
    'FORMV' IS A VISUAL OPTION OF THE FORMATTER. IT DISPLAYS
 THE FORMATTED OUTPUT ON THE PRINCIPAL I/O DEVICE FOR
 INSPECTION ONE LINE AT A TIME. IT PAUSES AFTER EACH LINE; <ESC> WILL
 ABORT, ANYTHING ELSE WILL CONTINUE.

++ COMMAND: FREE
USAGE: FREE ARGS: <DNUM>?

'FREE' DISPLAYS: (1) THE NUMBER OF FREE AND DEAD BLOCKS ON THE SPECIFIED DISK, (2) THE SIZE OF THE PRIMARY FILE IN BYTES AND BLOCKS, AND (3) THE AMOUNT OF FREE MEMORY SPACE AVAILABLE IN THE ARIAN WORKSPACE IN BYTES AND BLOCKS.

IF <DNUM> IS NOT SPECIFIED, THE LOGGED-IN DRIVE IS REPORTED ON. 'FREE' DOES NOT CHANGE THE LOGGED-IN DRIVE NUMBER.

++ COMMAND: HELP
 USAGE: HELP, HELPP   ARGS: <CNAME>?
    'HELP' IS THE HELP FILE PROGRAM. YOU ARE CURRENTLY RUNNING
 THIS PROGRAM. 'HELP' PRINTS THE HELP FILE INFORMATION ON THE
 PRINCIPAL I/O DEVICE, AND 'HELPP' PRINTS THE HELP FILE ON THE
 PRINTER.
    IF <CNAME> IS SPECIFIED, 'HELP' ONLY PRESENTS THE
 INFORMATION ON THE SPECIFIED COMMAND AND RETURNS TO ARIAN;
 OTHERWISE, A MENU IS GIVEN FROM WHICH THE USER CAN SELECT
 THE COMMANDS HE WANTS TO KNOW ABOUT.
    THE 'HELP' LEVEL 3 COMMAND DOES NOT AFFECT THE ARIAN
 WORKSPACE. IT ONLY AFFECTS THE DYNAMIC AREA IN THE ARIAN
 SCRATCHPAD RAM ABOVE 0F300 HEX.

++ COMMAND: PAGER
USAGE: PAGE, PAGEN, PAGEF   ARGS: <PAGENUM>?
   'PAGE' PRODUCES A PAGED LISTING OF THE PRIMARY FILE
ON THE PRINTER. IN RESPONSE TO THIS COMMAND, THE USER
IS PROMPTED WITH "HEADER:", TO WHICH HE RESPONDS WITH
A TITLE TO BE PLACED AT THE TOP OF EACH PAGE. THE PAGES
ARE NUMBERED AND SPACED APPROPRIATELY WITH TOP AND BOTTOM
MARGINS BY THIS COMMAND.
   'PAGEN' PRODUCES A LISTING WITHOUT LINE NUMBERS IN FRONT
OF EACH LINE, AND 'PAGEF' PRODUCES AN ASSEMBLER-FORMAT
LISTING.
   IF <PAGENUM> IS SPECIFIED, THE LISTING BEGINS ON THE
SPECIFIED PAGE NUMBER; OTHERWISE, THE LISTING BEGINS WITH
PAGE 1.

```
++ COMMAND:  UTILITY PROM
    THE FOLLOWING IS AN ABBREVIATED LIST OF SOME OF THE MORE
 COMMONLY-ACCESSED ROUTINES IN THE UTILITY PROM:

        GETD = D0DB     GETIN = D0B1    GETV = D0B4
        INP1 = D0CC     INP2 = D027     INP3 = D033
        LEDOUT = D0FC   OUT1 = D0CF     OUT2 = D02A
        OUT3 = D036     OUT12 = D0FF    OUT32 = D102
        OUTAB = D0E7    PADC = D069     PA2DC = D066
        PHL5D = D072    VDMOUT = D0E4   VDMSET = D0E1
```

```
++ COMMAND:  REGS
 USAGE: REGS            ARGS: (<REGNAME> <VALUE>)?
    'REGS' CAN BE USED TO DISPLAY OR ALTER THE VALUES OF
 THE REGISTERS STORED IN THE ARIAN REGISTER SAVE AREA FOR
 USE WITH THE 'CONT' COMMAND.  IF NO ARGUMENT IS SPECIFIED,
 THE REGISTER VALUES ARE DISPLAYED TO THE USER.  IF THE DUAL
 ARGUMENTS ARE SPECIFIED, THE SPECIFIC REGISTER OR REGISTER
 PAIR IS ASSIGNED THE SPECIFIED VALUE.
    VALID REGISTER/REGISTER PAIR NAMES ARE: A, F, B, C, D, E,
 H, L, A', F', B', C', D', E', H', L', AF, BC, DE, HL, AF',
 BC', DE', HL', IX, IY, SP
```

```
++ COMMAND:  SEND
 USAGE:  SEND, SENDR    ARGUMENTS: (C ! M ! P)? " <MESSAGE> "?
     THE 'SEND' LEVEL 3 COMMAND IS USED TO TRANSMIT MESSAGES
 FROM THE CURRENT REDIRECTED I/O DEVICE TO A LOGICAL ARIES-1
 SYSTEM DEVICE (CONSOLE, MODEM, OR PRINTER).  IF NO LOGICAL
 DEVICE IS SPECIFIED, THE MESSAGE IS SENT TO THE OPERATOR
 (CONSOLE), WHICH DISPLAYS THE MESSAGE FOLLOWED BY A PROMPT FOR
 ACTION BY THE OPERATOR.
     'SEND' JUST SENDS THE SPECIFIED MESSAGE TO THE SPECIFIED
 LOGICAL DEVICE (OR OPERATOR); 'SENDR' SENDS THE MESSAGE AND
 WAITS FOR A REPLY, WHICH IS TRANSMITTED BACK TO THE USER.
 NOTE:  SENDR AND SEND TO OPERATOR REQUIRE THE PRESENCE OF
 AN INDIVIDUAL AT THE RECEIVING DEVICE.
     VALID DEVICE NAMES ARE:  (1) V -- VDM, (2) C -- CONSOLE,
 (3) P -- PRINTER.  ONLY THE FIRST CHARACTER NEED BE
 SPECIFIED.
```

++ COMMAND: SUBSTITUTE

USAGE: SUBS, SUBSV, SUBSQ ARGS: "STR" "REP" (<LNUM> <LNUM>?)?

'SUBS' SUBSTITUTES THE STRING "REP" (EXCLUDING QUOTES) FOR ALL OCCURRANCES OF THE STRING "STR" IN THE PRIMARY FILE OVER THE RANGE SPECIFIED. IF NO LINE NUMBERS ARE GIVEN, THE SUBSTITUTION IS DONE OVER THE ENTIRE FILE. IF JUST ONE LINE NUMBER IS GIVEN, THE SUBSTITUTION IS DONE OVER JUST THAT LINE.

'SUBSV' DOES A VISUAL SUBSTITUTION -- IT DISPLAYS EACH LINE IT PERFORMS A SUBSTITUTION ON AS IT GOES. 'SUBSQ' DISPLAYS EACH LINE AFTER A SUBSTITUTION IS DONE AND PROMPTS THE USER WITH A '?', TO WHICH HE RESPONDS WITH "Y" TO HAVE THE SUBSTITUTION DONE AND "N" TO NOT HAVE THE SUBSTITUTION DONE ON THAT LINE.

```
++ COMMAND:  SYMBOL TABLE SORT ROUTINE
 USAGE: EXEC SYMT
     'SYMT' IS AN INITIALIZATION PROGRAM WHICH LOADS THE
 SYMBOL TABLE SORT/PRINT ROUTINE, CLEARS THE LOCAL TEXT FILE
 DIRECTORY, AND ESTABLISHES THE CUSTOMIZED COMMAND 'SYMT'.
     THE CUSTOMIZED COMMAND 'SYMT' HAS THE FORMS OF 'SYMT'
 AND 'SYMTP', WHERE 'SYMT' SORTS AND DISPLAYS ON THE PRINCIPAL
 I/O CHANNEL AND 'SYMTP' SORTS AND PRINTS ON THE PRINTER.
 ALSO, THE 'SYMT' COMMAND MAY TAKE THE FORM OF 'SYMT <LABEL>',
 IN WHICH CASE THE SPECIFIED LABEL WILL BE SEARCHED FOR AND
 ITS VALUE DISPLAYED.
     SEE THE SECTION IN THE "ARIAN USER'S MANUAL" ON THE SYMT
 EXECUTIVE COMMAND.
```

++ COMMAND: TMEM
USAGE: TMEM, AMEM

'TMEM' IS A MEMORY TEST PROGRAM, AND 'AMEM' IS AN ALTERNATE VERSION OF THIS PROGRAM FOR RUNNING IN A DIFFERENT MEMORY BLOCK. 'TMEM' CAN BE SET TO TEST EITHER SPECIFIC BLOCKS OF MEMORY OR ALL (EXCEPT FOR THE BLOCK IT RESIDES IN) OF MEMORY. A BLOCK CONTAINS 4K BYTES.

REFER TO "MEMORY TEST USER'S MANUAL" FOR FURTHER INFORMATION.

++ COMMAND: XDIR
USAGE: XDIR, XDIRX ARGS: <FNAME>? <DRIVENUMBER>?

'XDIR' DISPLAYS AN EXTENDED DIRECTORY TO THE USER. 'XDIR' JUST ALPHABETIZES AND NAMES THE FILES ON THE LOGGED-IN OR SPECIFIED DISK; 'XDIRX' ALPHABETIZES AND DISPLAYS ALL THE DIRECTORY ATTRIBUTES OF THE FILES ON THE LOGGED-IN OR SPECIFIED DISK.

IF <DRIVENUMBER> IS GIVEN, XDIR OPERATES ON THE SPECIFIED DRIVE; OTHERWISE, XDIR OPERATES ON THE LOGGED-IN DRIVE. THE LOGGED-IN DRIVE NUMBER IS NOT AFFECTED BY XDIR.

IF <FNAME> IS GIVEN, THE SPECIFIED FILE IS SEARCHED FOR, AND, IF FOUND, ALL THE DIRECTORY ATTRIBUTES OF THIS FILE ARE DISPLAYED.

```
++ COMMAND:  XEDIT
 USAGE: XEDIT           ARGS: <LNUM>
    'XEDIT' INVOKES THE EXTENDED CRT INTRA-LINE EDITOR
 ON THE SPECIFIED LINE OF THE PRIMARY FILE.  REFER TO
 "XEDIT USER'S MANUAL" FOR FURTHER INFORMATION.
```

SATCOM
U.S. ARMY SATELLITE
COMMUNICATIONS AGENCY