NAVAL SEA SYSTEMS COMMAND

NAVSEA 0967-LP-598-9040

CMS-2 LANGUAGE

REFERENCE BOOKLET

DISPLAYS PERIPHERALS SUPPORT SOFTWARE COMPUTERS

PMS 408

TACTICAL EMBEDDED COMPUTERS

SEPTEMBER 1988

The U.S. Government possesses the unlimited rights throughout the world for Government purposes, to publish, translate, reproduce, deliver, perform, and dispose of the technical data, computer software or computer firmware contained herein and to authorize others to do so.

NAVSEA 0967-LP-598-9040

CMS-2 LANGUAGE

This booklet is a quick reference for programmers and operators using the machine-transferable CMS-2 compiler (Revision 02) for the AN/UYK-7(V), AN/UYK-43(V), AN/UYK-20(V), AN/AYK-14, AN/AYK-14EIOP, AN/AYK-14SCP, AN/UYK-44(V), and MIL-STD-1750A
CMS-2 is a component of the MTASS system which generates object code for all of the above mentioned object computers.

Version	Computer	Operating System
01	Unisys 1100 Compatible	Unisys 1100 Time-sharing EXEC OS 1100 (Level 36 or later)
03	IBM 370 Compatible	IBM Operating System OS/VS2 (MVS) R3.7
10	IBM 370 Compatible	VM/370-CMS VM/SP Release 3
11	DEC VAX Compatible	VAX/VMS Operating System (Version 4.2 or later)

Reference Documents

User Handbook for CMS-2 Compiler, NAVSEA 0967-LP-598-8020

Program Performance Specification for CMS-2 Compiler
NAVSEA 0967-LP-598-9020

AN/AYK-14 Programmers Reference Manual
AN/AYK-14 Programmers Reference Card
AN/UYK-7 Programmers Reference Card
AN/UYK-43 Abbreviated Reference Manual
AN/UYK-44 Technical Description
AN/UYK-20 Technical Description
AN/UYK-7 Technical Description
AN/UYK-43 Technical Description

To obtain further information and to order additional copies of this booklet, please contact:

NAVAL SEA SYSTEMS COMMAND
PMS-412
Washington, DC 20362-5101
Telephone: (202)692-8204

CHARACTER SETS

CHARACTER	ASCII		FIELD DATA	EBCDIC	DISPLAY	BCD
	OCT.	HEX.	OCT.	HEX.	OCT.	OCT
NUL	000	00	–	00	–	–
SOH	001	01	–	01	–	–
STX	002	02	–	02	–	–
ETX	003	03	–	03	–	–
EOT	004	04	–	37	–	–
ENQ	005	05	–	2D	–	–
ACK	006	06	–	2E	–	–
BEL	007	07	–	2F	–	–
BS	010	08	–	16	–	–
HT	011	09	–	05	–	–
LF	012	0A	–	25	–	–
VT	013	0B	–	0B	–	–
FF	014	0C	–	0C	–	–
CR	015	0D	–	0D	–	–
SO	016	0E	–	0E	–	–
SI	017	0F	–	0F	–	–
DLE	020	10	–	10	–	–
DC1	021	11	–	11	–	–
DC2	022	12	–	12	–	–
DC3	023	13	–	–	–	–
DC4	024	14	–	3C	–	–
NAK	025	15	–	3D	–	–
SYN	026	16	–	32	–	–
ETB	027	17	–	26	–	–
CAN	030	18	–	18	–	–
EM	031	19	–	19	–	–
SUB	032	1A	–	3F	–	–
ESC	033	1B	–	27	–	–
FS	034	1C	–	22	–	–
GS	035	1D	–	–	–	–
RS	036	1E	–	35	–	–
US	037	1F	–	–	–	–
SP (Space)	040	20	05	40	55	20
! (Exclamation)	041	21	55	5A	–	77
″ (Quotes)	042	22	–	7F	–	76
# (Number)	043	23	03	7B	–	13
$ (Dollar sign)	044	24	47	5B	53	53
% (Percent)	045	25	52	6C	63	74
& (Ampersand)	046	26	–	50	–	32
′ (Apostrophe)	047	27	72	7D	–	57
((Left Parenthesis)	050	28	51	4D	51	35
) (Right Parenthesis)	051	29	40	5D	52	55
* (Asterisk)	052	2A	50	5C	47	54
+ (Plus)	053	2B	42	4F	45	60
, (Comma)	054	2C	56	6B	56	73

CHARACTER SETS (continued)

CHARACTER	ASCII		FIELD DATA	EBCDIC	DISPLAY	BCD
	OCT.	HEX.	OCT.	HEX.	OCT.	OCT
– (Minus)	055	2D	41	60	46	52
. (Period)	056	2E	75	4B	57	33
/ (Slant)	057	2F	74	61	50	61
0	060	30	60	F0	33	00
1	061	31	61	F1	34	01
2	062	32	62	F2	35	02
3	063	33	63	F3	36	03
4	064	34	64	F4	37	04
5	065	35	65	F5	40	05
6	066	36	66	F6	41	06
7	067	37	67	F7	42	07
8	070	38	70	F8	43	10
9	071	39	71	F9	44	11
: (Colon)	072	3A	53	7A	00	15
; (Semicolon)	073	3B	73	5E	77	56
< (Less than)	074	3C	43	4C	72	36
= (Equals)	075	3D	44	7E	54	75
> (Greater than)	076	3E	45	6E	73	16
? (Question Mark)	077	3F	54	6F	–	17
@ (At)	100	40	00	7C	–	14
A	101	41	06	C1	01	21
B	102	42	07	C2	02	22
C	103	43	10	C3	03	23
D	104	44	11	C4	04	24
E	105	45	12	C5	05	25
F	106	46	13	C6	06	26
G	107	47	14	C7	07	27
H	110	48	15	C8	10	30
I	111	49	16	C9	11	31
J	112	4A	17	D1	12	41
K	113	4B	20	D2	13	42
L	114	4C	21	D3	14	43
M	115	4D	22	D4	15	44
N	116	4E	23	D5	16	45
O	117	4F	24	D6	17	46
P	120	50	25	D7	20	47
Q	121	51	26	D8	21	50
R	122	52	27	D9	22	51
S	123	53	30	E2	23	62
T	124	54	31	E3	24	63
U	125	55	32	E4	25	64
V	126	56	33	E5	26	65
W	127	57	34	E6	27	66
X	130	58	35	E7	30	67
Y	131	59	36	E8	31	70
Z	132	5A	37	E9	32	71

CHARACTER SETS (continued)

CHARACTER	ASCII		FIELD DATA	EBCDIC	DISPLAY	BCD
	OCT.	HEX.	OCT.	HEX.	OCT.	OCT
[(Left Bracket)	133	5B	01	AD	61	12
\ (Reverse Slant)	134	5C	57	E0	–	37
] (Right Bracket)	135	5D	02	BD	62	34
^ (Circumflex)	136	5E	–	5F	–	40
_ (Underline)	137	5F	–	6D	–	72
‘ (Grave Accent)	140	60	–	79	–	–
a	141	61	–	81	–	–
b	142	62	–	82	–	–
c	143	63	–	83	–	–
d	144	64	–	84	–	–
e	145	65	–	85	–	–
f	146	66	–	86	–	–
g	147	67	–	87	–	–
h	150	68	–	88	–	–
i	151	69	–	89	–	–
j	152	6A	–	91	–	–
k	153	6B	–	92	–	–
l	154	6C	–	93	–	–
m	155	6D	–	94	–	–
n	156	6E	–	95	–	–
o	157	6F	–	96	–	–
p	160	70	–	97	–	–
q	161	71	–	98	–	–
r	162	72	–	99	–	–
s	163	73	–	A2	–	–
t	164	74	–	A3	–	–
u	165	75	–	A4	–	–
v	166	76	–	A5	–	–
w	167	77	–	A6	–	–
x	170	78	–	A7	–	–
y	171	79	–	A8	–	–
z	172	7A	–	A9	–	–
{ (Left Brace)	173	7B	–	8B	–	–
| (Vertical Line)	174	7C	–	6A	–	–
} (Right Brace)	175	7D	–	9B	–	–
~ (Tilde)	176	7E	–	A1	–	–
DEL	177	7F	–	07	–	–

NOTATION OF STATEMENTS AND OPERATIONS

Each description of a statement or an operation in this reference booklet uses a uniform system of notation to define the structure of the statement. This notation is not a part of CMS-2, but is a standardized notation that may be used to describe the syntax (construction) of the CMS-2 language. It provides a brief but precise means of explaining the general patterns that the language permits. It does not describe the meaning of the statement or operations; it merely describes structure; that is, it indicates the order in which the operands must appear, the punctuation required, and the options allowed.

The following rules explain this standard notation:

1) A word written in lowercase letters represents the type of entry to be made by the programmer. This word may be hyphenated.

name	denotes an entry of a name.
data-unit-name	denotes an entry of a data unit name.

2) A word written in uppercase letters or special characters denotes an actual occurrence of that word or character in the language.

name EQUALS tag-expression	allows a symbolic name denoted by name to be associated with the value defined by a tag expression.

3) A vertical stack of units under an underlined term donotes a choice. At least one of the units in the stack must occur in the statement.

<u>connector</u> OR XOR AND	indicates that either OR, XOR or AND must appear in the statement in place of <u>connector</u>.

4) Square brackets [ ] denote options. A single unit enclosed in brackets is optional; it may or may not appear. A list of units enclosed in brackets denotes a choice of one or none from that list. Generally, no more than one unit from the list may appear.

[name]	indicates that a name may appear in the statement format. However, this unit is not required.

5) The use of • ••• denotes that the type of entry indicated by the word preceding ••• may appear one or more times in succession, where each entry is delimited by the word following •. This does not imply that all entries should be identical. It does imply, however, that all entries should be the same type of entry indicated by the word preceding the three dots. Where there are two or more entries, they are separated by commas (,).

•data-unit-name•••	indicates that one or more data unit names may occur in succession as entries, separated by commas. Thus, the following would be a legal entry: ALPHA, BETA, GAMMA.

6) A word written in lowercase letters and underlined represents a descriptive term.

```
computer

UYK7                  Indicates that this parameter
UYK43                 identifies the target computer.
UYK43EMR
UYK20                 The trailing M  or (MATH)
UYK20M                attached to the target computer
UYK20A[(MATH)]        name indicates the computer
UYK44[(UG1)]          has the MATHPAC optional
UYK44M[(UG1)]         hardware.
AYK14
AYK14E                The trailing (UG1) indicates the
AYK14EIO              UYK44 target computer
AYK14SCP              contains the User Growth One
MS1750A               instructions.
MS1750NP
```

When a descriptive term has been defined once with a list of alternatives, the alternatives are not listed in subsequent appearances of the underlined descriptive term. For example, the first time the term type is used, the possible alternatives for type are given. In subsequent references just the descriptive term type is used.

7) Editorial Comment - For statements that are physically too long to be completed on one line, the lines following the first are indented to signify continuation. The dollar sign ($) character signifies the end of a CMS-2 statement and is not part of the standard notation.

CMS-2 STATEMENTS

The statements within each of the sections for CMS-2 are given in alphabetical order by using the statement symbol.

STATEMENT FORMAT

CMS-2 source cards consist of a card identification field in columns 1 through 10 (columns 71 through 80 for OPTION COL1) and a statement field in columns 11 through 80 (columns 1 through 70 for OPTION COL1) as shown in the following:

```
CC         CC  CC                          CC
 1         10  11                          80
+-------------+-----------------------------+
|    CARD     |                             |
|IDENTIFICATION| STATEMENT$STATEMENT$...    |
+-------------+-----------------------------+
```

If COL1 is present in the OPTIONS statement, then the CMS-2 source cards consist of a card identification field in columns 71 through 80 and a statement field in columns 1 through 10 as shown in the following:

```
CC                          CC  CC         CC
 1                          70  71         80
+-----------------------------+-------------+
|                             |    CARD     |
|  STATEMENT$STATEMENT$...    |IDENTIFICATION|
+-----------------------------+-------------+
```

The identification field may be used for program identification and sequence numbers and has no effect on program compilation.

The statement field has a free format. Each CMS-2 statement is terminated by a dollar sign ($). There may be more than one statement on a card or a statement may require more than one card. A statement will continue in columns 11-80 (columns 1-70 for OPTION COL1) of each card until a dollar sign is encountered. If a symbol or string of characters is to span two cards, the first part must end in column 80 (column 70 for OPTION COL1) of the first card and the second part must start in column 11 (column 1 for OPTION COL1) of the second card.

Names, compiler keywords, and constants must be separated from each other by a blank character or a delimiter. When a delimiter is used as a separator, blank characters are not necessary but may be used if desired. A blank character may not be used within a name, compiler keyword, constant, or between the name and the period character in a statement label.

An embedded comment may be used anywhere a blank is allowed. An embedded comment consists of 2 consecutive single primes (') followed by comment text and terminated by 2 consecutive single primes ('). The comment text may not contain a dollar sign. The embedded comment is replaced by a single blank character during statement processing.

BASIC DEFINITIONS

```
conditional-expression
(conditional-expression)
expression               relational-operator      expression
                               EQ
                               NOT
                               GT
                               LTEQ
                               GTEQ
                               LT
Boolean-function-call
Boolean-data-unit
Boolean-constant
COMP conditional-expression
conditional-expression       connector      conditional-expression
                                AND
                                OR
data-unit         type
                  VALID
                  INVALID
data-unit         type
                  ODDP
                  EVENP

character-constant
H(character-string)

data-unit
name
name (name)
name (•numeric-expression•••)
name (•numeric-expression•••, name)

expression
numeric-expression
Boolean-expression
status-expression
character-expression
bit-string-expression
```

```
status-expression
status-constant
status-data-unit
status-function-call
status-expression

status constant
'character-string'

character-expression
character-constant
character-data-unit
character-function-call
(character-expression)
character-expression CAT character-expression

bit-string-expression
expression   connector    expression
                OR
                XOR
                AND

COMP expression

numeric-constant
O(octal-integer [.[octal-integer]] [E [add-op] octal-integer])
                                         +
                                         -

O(.octal-integer [E [add-op] octal-integer])

decimal-constant
X(hexadecimal-number  [.[hexadecimal-number]])
X(.hexadecimal-number)

decimal-constant
D(decimal-constant)
decimal-constant D

numeric-expression
(numeric-expression) [scaling specifier]
numeric-function-call [scaling specifier]
intrinsic-function-call [scaling specifier]
numeric-data-unit [scaling specifier]
numeric-tag
add-op numeric-expression

numeric-expression    operator      numeric-expression
                         +
                         -
                         *
                         /
                         **

where:   numeric-tag
         name
         add-op numeric-constant
```

PROGRAM STRUCTURE STATEMENTS

```
         COMMENT            [comment-text]          $
                            character-string
                            ((EJECT
                            ((SKIPn
                            ((LINE*
         CSWITCH name $
         END-CSWITCH name $
         END-CSWITCHS $
         CSWITCH-ON •name••• $
         CSWITCH-OFF •name••• $
  name   SYSTEM $
         END-SYSTEM name $
```

HEADER DECLARATIVE STATEMENTS

```
  ACSEPARATION  $

  CMODE  [constant-mode]    $
            O
            D

  CMODE   conversion-mode     $
          SINGLE
          DOUBLE
          FLOAT
          QUAD
          FLOAT, QUAD
          QUAD, FLOAT

  CSWITCH-DEL $

  DEBUG   •debug-parameter•••        $
             SNAP
             DISPLAY
             TRACE
             PTRACE
             RANGE

  name     EQUALS tag-expression $

  EXECUTIVE $

  [name] HEAD $

  END-HEAD [name] $

  LOAD-VRBL  variable list type  P numeric-constant-expression $
               name
               (•name•••)

  MODE FIELD [type] $

  MODE VRBL [type] [P preset-tag] $

  NITEMS (name) EQUALS tag-expression $
```

```
OPTIONS   computer           [,•option•••] $
          UYK7               SOURCE
          UYK43              OBJECT
                                [(•object-specification•••)]
          UYK43EMR           LEVEL (level-specification)
          UYK20              MONITOR
          UYK20M             STRUCTURED
          UYK20A[(MATH)]     NONRT
          UYK44[(UG1)]       MSCALE
          UYK44M[(UG1)]      CLASS (security[,security])
          AYK14              LINE (lines-per-page)
          AYK14E             OPTIMIZE (optimize-level)
          AYK14EIO           INDEPENDENT
          AYK14SCP           HEX
          MS1750A            COL1
          MS1750NP           FARMODE
```

```
   where:  object-specification
           CMP [(name)]
           CR
           CRG
           CRL
           SA
           SM
           CNV
           SCR
           SCRG
           SCRL
           LEVEL (level-specification)
           SADUMP

   where:  level-specification
                0
                1
                C
                W
                F

   where:  security
                U
                C
                S
                T
                UW
                CW
                SW
                TW

   where:  optimize-level
           octal-integer (valid octal values are 0, 1, 3, 5, 7, 11, 13, 15, 17)
```

```
PASSAGE-SPEC   passage-type [namelist] $
               DIRECT
               REGISTER[,CALLING ONLY]
```

```
[name]   pooling-type           [([ ref-type ], [tag])] [tag] $
         LOCDDPOOL                     T
         TABLEPOOL                     F
         DATAPOOL
         BASE
         LOCDDPOOLR
         LOCDDPOOLW
         TEMPSPOOL
         CONSTPOOL
         FARIWSPOOL

SINGLE $

SPILL $

name substitution-type [character-string] $
         MEANS
         EXCHANGE

SYS-INDEX •register-number name••• $
```

SYSTEM DATA DECLARATIVE STATEMENTS

```
CMS-2 $

[data-unit-name]  DATA     data-entry $
                           character-constant
                           tag [,scaling] [tag[,scaling]]
                           tag [,scaling] CORAD (name)
                           CORAD (name) [tag[,scaling]]

      where:  scaling
               tag

DIRECT $

FIELD name   [type] [starting-word starting-bit] [P •preset-item•••] $
      where:  type
              F [(floating-point-attribute)]
              B
              A number-of-bits  sign  number-of-fractional-bits
                                 S
                                 U
              I number-of-bits  sign
                                 S
                                 U
              H number-of-characters
              S•status-constant•••
      where:  floating-point-attribute
                   T
                   R
                   S
                   D
      where:  preset-item
              numeric-tag
              character-constant
              CORAD (data-unit)
              FCORAD (data-unit)
              repeat-count (numeric-tag)
              repeat-count (character-constant)
              repeat-count (CORAD (data-unit))
              repeat-count (FCORAD (data-unit))
```

```
[external]     FILE nonstandard-file-name file-specification (continued)
(EXTREF)       nonstandard-hardware-name (continued)
(EXTDEF)       [•status-constant•••] [WITHLBL] $
(TRANSREF)

     where:   file-specification
              file-type numeric-constant-expression (continued)
              file-structure  numeric-constant-expression

     where:   file-type
              H
              B

     where:   file-structure
              R
              V
              S

     where:   nonstandard-hardware-name
                    MT1
                    MT2
                    MT3
                    MT4
                    MT5
                    MT6
                    MT7
                    MT8
                    MT9
                    MT10
                    MT11
                    MT12
                    MT13
                    MT14
                    MT15
                    MT16
                    PPTR
                    PPTP
                    name

[external]     FILE standard-file-name file-specification (continued)
(EXTREF)       standard-hardware-name [•status-constant•••] $
(EXTDEF)
(TRANSREF)

     where:   standard-hardware-name
                    PRINT
                    PUNCH
                    READ
                    OCM

[external]     FORMAT name •format-item••• $
(EXTREF)
(EXTDEF)
(TRANSREF)

     where:   format-item
              [repeat-count] format-descriptor
              format-positioner
              [repeat-count] character-constant
              repeat-count (format-list)
              [format-item]/[format-item]
```

```
     where:   repeat-count
                 tag

     where:   format-descriptor
              numeric-editing-code tag [.tag]
              character-editing-code  tag

     where    numeric-editing-code
                    I
                    O
                    F
                    E

     where:   character-editing-code
                    A
                    L

     where:   format-positioner
                 tag
                 T tag

[external]     FUNCTION function-name (continued)
(EXTREF)       (•formal-input-parameter•••)[type]  $
(TRANSREF)
(FARREF)

     where:   formal-input-parameter
              name
              CORAD (name)
              FCORAD (name)

[external]     INPUTLIST    inputlist-name      •inputlist-item••• $
(EXTREF)                                        input-receptacle
(EXTDEF)                                        name
(TRANSREF)                                      *data-unit

     where:   input-receptacle
              data-unit
              CORAD(name)

[external]     ITEM-AREA •name••• $
(EXTREF)
(EXTDEF)
(FARREF)
(TRANSREF)

[external]     LIKE-TABLE name [number-of-items] (continued)
(EXTREF)       [major-index-name] $
(EXTDEF)
(FARREF)
(TRANSREF)

     where:   number-of-items
              numeric-constant-expression
              LTAG
              status-type

[external]     OUTPUTLIST  outputlist-name      •outputlist-item••• $
(EXTREF)                                        expression
(EXTDEF)                                        name
(TRANSREF)                                      data-unit
                                                *data-unit
```

```
field-name OVERLAY   •field-overlay-sibling•••
                     field-name
                     numeric-constant-expression

data-unit-name OVERLAY        •overlay-sibling••• $
                              data-unit-name
                              numeric-constant-expression

[external]   PARAMETER parameter-name [type] (continued)
(EXTREF)     [P preset-tag], numeric-constant-expression $
(EXTDEF)
(TRANSREF)
(FARREF)

[external]      PROCEDURE procedure-name (continued)
(EXTREF)        [INPUT •formal-input-parameter•••] (continued)
(FARREF)        [OUTPUT •name•••] [EXIT •name•••] $
(TRANSREF)

[external]   P-SWITCH pindex-switch-name  [INPUT (continued)
(EXTREF)     •formal-input-parameter•••] [OUTPUT •name•••] $
(EXTDEF)
(TRANSREF)
(FARREF)

             pindex-list
             [P] procedure-name $
             pindex-list [P] pindex-switch-name $

             end-pswitch-declaration
             END-SWITCH pindex-switch-name $
             END-P-SW   pindex-switch-name $

[external]      P-SWITCH pitem-switch-name (continued)
(EXTREF)        (variable-name) (continued)
(EXTDEF)        [INPUT •formal-input-parameter•••] (continued)
(TRANSREF)      [OUTPUT •name•••] $
(FARREF)

                pitem-list
                switch-value, procedure-name $
                pitem-list switch-value, procedure-name $
                end-pswitch-declaration

data-unit-name  RANGE     upper-range . . . [lower-range] $

    where:  upper-range
            numeric-constant-expression

    where:  lower-range
            numeric-constant-expression

[external]      STRINGFORM stringform-name •stringform-item••• $
(EXTREF)
(EXTDEF)
(TRANSREF)

    where:  stringform-item
            [repeat-count]  stringform-descriptor
            stringform-positioner
            [repeat-count]  character-constant
            [repeat-count]  (•stringform-item•••)
```

```
    where:  stringform-descriptor
            D  tag.tag[.tag]
            I  tag
            B  tag
            O  tag
            X  tag
            C  tag
            E  tag

    where:  stringform-positioner
            Z  tag
            T  [direction] tag

    where:  direction
            +
            -

[external]      SUB-TABLE sub-table-name (continued)
(EXTREF)        starting-item-number number-of-items (continued)
(EXTDEF)        [major-index-name] $
(TRANSREF)
(FARREF)

    where:  starting-item-number
            numeric-constant-expression
            status-constant

sys-dd-name     SYS-DD $

                END-SYS-DD sys-dd-name $

[external]   TABLE name A    packing (continued)
(EXTREF)                     words-per-item
(EXTDEF)                     NONE
(TRANSREF)                   MEDIUM
(FARREF)                     DENSE
                             (type)

             [indirect-indicator] •dimension••• $
             INDIRECT
             FINDIRECT

[external]   TABLE name  [form] packing (continued)
(EXTREF)                  V
(EXTDEF)                  H
(TRANSREF)
(FARREF)     [indirect-indicator] number-of-items (continued)
             INDIRECT
             FINDIRECT

             [major-index-name] $

             END-TABLE name $

[(EXTDEF)]   TYPE name type $

[(EXTDEF)]   TYPE name packing $
             END-TYPE name $

[external]   VRBL variable-list [type] [P     preset-tag]  $
(EXTREF)        name                          numeric-tag
(EXTDEF)        (•name•••)                    character-constant
(TRANSREF)                                    CORAD (data-unit)
(FARREF)                                      FCORAD (data-unit)
```

SYSTEM PROCEDURE STATEMENTS

```
auto-dd-name      AUTO-DD $

                  END-AUTO-DD auto-dd-name $

[(EXTDEF)]        EXEC-PROC exec-proc-name
                  [INPUT •formal-input-parameter•••] $

                  END-PROC exec-proc-name $

[(LOCREF)]        EXEC-PROC exec-proc-name
                  [INPUT •formal-input-parameter•••] $

[(EXTDEF)]        FUNCTION function-name ([•formal-input- (continued)
                  parameter•••]) [type] $
                  RETURN (expression) $
                  END-FUNCTION function-name $

[(LOCREF)]        FUNCTION function-name ([•formal-input- (continued)
                  parameter•••]) [type] $

[loc-dd-name]     LOC-DD      [access] $
                                R
                                W

                  END-LOC-DD [loc-dd-name] $

[(EXTDEF)]        PROCEDURE procedure-name (continued)
                  [INPUT •formal-input-parameter•••] (continued)
                  [OUTPUT •name•••] [EXIT •name•••] $

                  END-PROC procedure-name $

(LOCREF)          PROCEDURE procedure-name (continued)
                  [INPUT •formal-input-parameter•••] (continued)
                  [OUTPUT •name•••] [EXIT •name•••] $

SWITCH            index-switch-name, index-switch-name $

                  double-switch-list
                  [S] statement-label [,statement-label] $
                  double-switch-list [S] statement-label [,statement-label] $

END-SWITCH        index-switch-name, index-switch-name $

SWITCH            item-switch-name (variable-name) $

                  item-switch-list
                  switch-value, statement-label $
                  item-switch-list switch-value, statement-label $

END-SWITCH        item-switch-name $

SWITCH            switch-name •statement-label••• $

SWITCH switch-name $

                  switch-list
                  [S] statement-label $
                  switch-list [S] statement-label $

END-SWITCH switch-name $

sys-proc-name     SYS-PROC $

                  END-SYS-PROC sys-proc-name $
```

```
sys-proc-name     SYS-PROC-REN $

                  END-SYS-PROC sys-proc-name $
```

PROCEDURE BODY STATEMENTS

```
[statement-labels]                BEGIN    [•for-value•••] $
     name.                                  constant-numeric-expression
     statement-labels name.                 character-constant
                                            status-constant

[statement-labels] CHECKID user-defined-file-name label-definition $

[statement-labels] CLOSE user-defined-file-name $

[statement-labels] DECODE data-unit •input••• format-name $

[statement-labels] DEFID   user-defined-file-name    label-definition $
                                                     STANDARD
                                                     (character-string)

[statement-labels] DISPLAY          •display-item••• $
                                    data-unit [preset-magnitude]
                                    REGS

                  ELSE    simple-statement
                          begin-block
                          debug-phrase
                          direct-code-block
                          exec-phrase
                          exit-phrase
                          for-block
                          goto-phrase
                          input/output-phrase
                          null-phrase
                          procedure-call-phrase
                          procedure-switch-call-phrase
                          resume-phrase
                          return-phrase
                          set-phrase
                          shift-phrase
                          stop-phrase
                          swap-phrase
                          vary-block

          where:   input/output-phrase
                   open-phrase
                   close-phrase
                   endfile-phrase
                   define-label-phrase
                   check-label-phrase
                   file-positioning-phrase
                   record-positioning-phrase
                   output-phrase
                   input-phrase
                   encode-phrase
                   decode-phrase

                ELSIF  conditional-expression THEN simple-statement $

[statement-labels]    ENCODE    data-unit •output••• format-name $
```

```
[statement-labels] END      [statement-label]

[statement-labels] ENDFILE user-defined-file-name $

[statement-labels] EXEC     numeric-constant-expression  (continued)
                            [,numeric-expression]  $

[statement-labels] EXIT  [statement-label] $

FIND find-condition [varying-clause] $

     where:  find-condition
             find-relational-expression
             [binary-connector  conditional-expression]

     where:  find-relational-expression
             data-unit relational-operator expression

     where:  binary-connector
             AND
             OR

     where:  varying-clause
             VARYING  index-clause

     if-data-clause  simple-statement

     [else-clause] $

     where:  if-data-clause
             IF DATA FOUND THEN
             IF DATA NOTFOUND THEN

[statement-labels] FOR      expression [,(type)] [ELSE simple-statement] $

value-block-list END [statement-label]

     where:  value-block-list
             value-block
             value-block-list  value-block

     where:  value-block
             [statement-labels]      BEGIN ●value●●● $

                                     [●statement●●●]

                                     END [statement-label] $

     where:  value
             numeric-constant-expression
             character-constant
             status-constant

[statement-labels]      GOTO    index-switch-name (continued)
                        numeric-expression  (continued)
                        [INVALID statement-label]  [special-condition] $

[statement-labels]      GOTO    item-switch-name [INVALID (continued)
                        statement-label] [special-condition] $

[statement-labels]      GOTO    statement-label [special-condition] $
```

```
     where:  special-condition
             KEY1
             KEY2
             KEY3
             STOP
             STOP5
             STOP6
             STOP7

[statement-labels] IF   conditional-expression THEN simple-statement $

[statement-labels] INPUT    input-file-name ●input-list●●● [format-name]  $

     where:  input-file-name
             name
             READ
             OCM

     where:  input-list
             input-item
             (input-items)

     where:  input-item
             data-unit
             multiple-subscript-data-unit

     where:  input-items
             input-item
             input-items,input-item

     LOC-INDEX ●name●●● $

[statement-labels] OPEN user-defined-file-name i/o-capability $

where:  i/o-capability
        INPUT
        OUTPUT
        SCRATCH

[statement-labels] OUTPUT output-file-name [●output-list●●●] (continued)
                          format-name $

     where:  output-file-name
             name
             PRINT
             PUNCH
             OCM

     where:  output-list
             output-item
             (output-items)

     where:  output-items
             output-item
             output-items,output-item

     where:  output-item
             data-unit
             multiple-subscript-data-unit
             numeric-constant
             character-constant

     where:  multiple-subscript-data-unit
             name ([subscript,] ●multiple-field-list●●●)
             name (item-range [,●multiple-field-list●●●])
```

```
   where:  multiple-field-list
           name
           numeric-expression
           ●multiple-field-list●●● , name
           ●multiple-field-list●●● , numeric-expression

   where:  item-range
           (●subscript-list●●●)...(●subscript-list●●●)

[statement-labels]      pindex-switch-name USING numeric- (continued)
                        expression [INVALID statement-label] (continued)
                        [INPUT ●expression●●●] (continued)
                        [OUTPUT ●receptacle●●●] $

[statement-labels]      pitem-switch-name [INVALID statement- (continued)
                        label] [INPUT ●expression●●●] (continued)
                        [OUTPUT ●receptacle●●●] $

[statement-labels]      procedure-name [INPUT ●expression●●●] (continued)
                        [OUTPUT ●receptacle●●●] (continued)
                        [EXIT ●statement-label●●●] $

[statement-labels] RESUME [statement-label] $

[statement-labels] RETURN  (expression) $

[statement-labels] RETURN  [name] [special-condition] $

[statement-labels] SET       ●receptacle●●● TO expression (continued)
                             [remainder-phrase] [overflow-phrase]  $

   where:  receptacle
           data-unit [scaling-specifier]
           CORAD (name)
           FCORAD (name)
           CHAR (starting-character[,count]) (data-unit)
           BIT (starting-character [,count]) (data-unit)

   where:  remainder-phrase
           SAVING data-unit

   where:  overflow-phrase
           OVERFLOW statement-label

[statement-labels] SET FIL (user-defined-file-name) TO numeric-expression $

[statement-labels] SET POS     (user-defined-file-name) TO (continued)
                               numeric-expression $

[statement-labels] SHIFT data-unit shift-type [-] shift-count (continued)
                   [INTO receptacle] $

   where:  shift-type
           CIRC
           ALG
           LOG

[statement-labels] SNAP data-unit [preset-magnitude] $

[statement-labels] STOP [stop-special-condition] $
                         KEY1
                         KEY2
                         KEY3
                         STOP5
                         STOP6
                         STOP7
```

```
[name]  SUB-DD $

        END-SUB-DD [name] $

[statement-labels] SWAP   swap-operands
                          receptacle , receptacle
                          receptacle AND receptacle

[statement-labels]      TRACE $

                        END-TRACE $

[statement-labels]      VARY     [data-unit] [FROM index-value] (continued)
                                 [THRU loop-value] [WITHIN (continued)
                                 name] [BY [-] numeric- (continued)
                                 expression] [WHILE test-value] (continued)
                                 [UNTIL conditional-expression] $

                                 [simple-statements]

                        END      [statement-label] $

   where:  index-value
           numeric-expression
           status-expression

   where:  loop-value
           numeric-expression
           status-expression

   where:  test-value
           conditional-expression
           data-unit
```

FUNCTION CALLS

User Function Call:

```
   user-function-name ([●expression●●●]) $
```

Intrinsic Function Calls:

```
   ABS (numeric-expression)

   ANDF (expression, expression)

   BIT (numeric-expression[,numeric-expression]) (data-unit)

   CHAR (numeric-expression[,numeric-expression]) (data-unit)

   CNT     (expression)
           numeric
           Boolean
           status
           character
           bit-string

   COMPF (expression)

   CONF (numeric-type,numeric-expression)

   CORAD  (address-operand)
          data-unit
          statement-label-name
```

FCORAD (address-operand)

FIL (name)

FIRST (status-type)

LAST (status-type)

LENGTH (file-name)

ORF (expression,expression)

POS (file-name)

PRED (status-expression)

REM (numeric-expression)

SCALF (numeric-constant-expression,numeric-expression)

SHIFTAL (numeric-expression,numeric-expression)
SHIFTAR (numeric-expression,numeric-expression)

SHIFTCL (numeric-expression,numeric-expression)
SHIFTCR (numeric-expression,numeric-expression)

SHIFTLL (numeric-expression,numeric-expression)
SHIFTLR (numeric-expression,numeric-expression)

SUCC (status-expression)

TDEF (numeric-type,expression)

XORF (expression,expression)

SUPPLIED PROCEDURE CALLS

<u>supplied-procedure</u>	INPUT numeric-expression, *(continued)*
VECTOR	numeric-expression [,numeric- *(continued)*
VECTORP	expression] OUTPUT [data-unit], *(continued)*
VECTORH	[data-unit] **$**
VECTORHP	
ROTATE	
ROTATEP	
ROTATEH	
ROTATEHP	

Fixed Point Arithmetic Function Calls

BAMS	(numeric-expression)
HLN	(numeric-expression)
ICOS	(numeric-expression)
IEXP	(numeric-expression)
ISIN	(numeric-expression)
ISQRT	(numeric-expression)
LN	(numeric-expression)
RAD	(numeric-expression)

Floating Point Arithmetic Function Calls

SIN	(numeric-expression)
COS	(numeric-expression)
TAN	(numeric-expression)
ASIN	(numeric-expression)
ACOS	(numeric-expression)
ATAN	(numeric-expression)
EXP	(numeric-expression)
ALOG	(numeric-expression)
SQRT	(numeric-expression)
ASIN2	(numeric-expression,numeric-expression)
ACOS2	(numeric-expression,numeric-expression)
ATAN2	(numeric-expression,numeric-expression)

DIRECT CODE STATEMENTS

Direct Code Statement Format

The format of direct code statements is consistent with CMS-2 source cards in that columns 1 through 10 (columns 71-80 if COL1 OPTION is present) are considered to be the card identification field, which is ignored by the compiler.

The direct code format consists of four fields separated by at least one blank as follows:

[label] operation operand [.comment]

The label must always start in column 11 (column 1 if COL1 OPTION is present). Labels having an x subscript below may externally define a symbol by suffixing it with an asterisk (*). The operation may be a machine-instruction mnemonic or a direct code directive. The operand field may contain subfields separated by commas as specified for the operation code. The operand field may contain the dollar sign ($) to signify the current value of the location counter. A period (.) followed by a blank signifies the end of the statement and the remainder of the line may contain a comment.

Basic Direct Code Definitions

<u>character-constant</u>
'character-string'

<u>direct-code-constant</u>
single-word-integer-constant
double-word-integer-constant
character-constant
floating-point-numbers
scaled-decimal-numbers
scaled-octal-numbers
scaled-hexadecimal-numbers

<u>double-word-integer-constant</u>
decimal-integer D
hexadecimal-number D
octal-integer D

<u>instruction-expression</u>
name [± single-word-integer-constant]
$ [± single-word-integer-constant]
[± single-word-integer-constant]
literal

<u>literal</u>
(direct-code-constant)

<u>single-word-integer-constant</u>
decimal-integer
hexadecimal-number
octal-integer

Direct Code Directives

[label$_x$] ABS name [± single-word-integer-constant]

[label$_x$] ABSD name [± single-word-integer-constant]

BYTE number-of-characters, size-of-character-field

CHAR c1, e1, c2, e2,...,cn, en

where: cn
octal-code *(000 through 377)*

where: en
expression

[label] DO single-word-integer-constant,direct-code-constant

EVEN

form-label FORM •direct-code-constant•••

[label$_x$] form-label •instruction-expression•••

ODD

ORIG name [± single-word-integer-constant]

REORIG

[label$_x$] RES instruction-expression

built-in-function-label BIFD n, b1 [P v1], b2 [P v2],...bi [P vi]

where: n
number-of-words

where: bi
number-of-bits-in-the-field

where: P
preset-value-keyword-indicator

where: vi
preset-value

[label$_x$] PAGE mp, relocatable name ± constant

LINKAGE AND PARAMETER PASSING

All procedures and functions are called using the following conventions: JLR R4,NAME for 16-bit ISA, JS R4,NAME for 1750A ISA, and LBJ B6,NAME for 32-bit ISA. NAME is the called procedure or function.

Parameters are passed in registers or memory according to the Parameter Passage Declaration as follows:

- DIRECT – parameters passed in memory and all code affecting the passing of values is to be generated in the calling program.
- REGISTER – parameters will be passed in registers. Code to load the parameters into registers will be generated by the calling program. Code to store actual parameter into the formal parameter will be generated in the called program.
- REGISTER[,CALLING ONLY] – parameters will be passed in registers. Code to load the parameters into registers will be generated by the calling program. No code will be generated in the called program.

Input parameters passed in registers will be assigned as follows:

- 16-bit ISA and 1750A ISA – R5, R3-R0, R15-R12
- 32-bit ISA – A0-A7

Output parameters passed in registers will be assigned as follows:

- 16-bit ISA and 1750A ISA – R5-R0 and R15-R12
- 32-bit ISA – A0-A7

The parameters are assigned to registers from left to right as they appear in the procedure or function call. When all registers have been used, remaining parameters are passed directly in memory.

REGISTER SAVING CONVENTIONS

The calling program is responsible for saving and restoring the contents of any registers in the group R0-R5 or R12-R15 for 16-bit ISA and 1750A ISA targets and A0-A7 or B6-B7 for 32-bit ISA targets that contain data that must be preserved across a procedure or function call. The calling program is also responsible for loading and storing the contents of these registers when used for parameter passing before and after a procedure or function call.

The called program is responsible for saving and restoring the contents of R6-R11 for 16-bit ISA and 1750A ISA targets and B1-B5 for 32-bit ISA targets when used as compiler work registers or as local indices. Registers declared as system indices will never be saved and restored.

ADDRESS COUNTER USAGE

Address counter usage can be controlled by using the pooling declarations or by using ACSEPARATION. The CMS-2 compiler uses address counters in the following ways:

- When pooling declarations are present, the following specifies the compiler action:

DECLARATION	CS-NAME	AC-NAME	AC-NUMBER
LOCDDPOOL	LOCDD or user-specified	LOC-DD name	2
TABLEPOOL	TABLE or user-specified	unnamed	3
DATAPOOL	SYSDD or user-specified	SYS-DD name	1
BASE	SYSP or user-specified	SYS-PROC name	0
LOCDDPOOLR	CONST or user-specified	LOC-DD R name	5
LOCDDPOOLW	AUTODD or user-specified	LOC-DD W name	4
TEMPSPOOL	TEMP or user-specified	unnamed	6
CONSTPOOL	CONST or user-specified	LOC-DD R name or unnamed	5
FARIWSPOOL	FARIWS or user-specified	unnamed	8

- When ACSEPARATION is specified, the following specifies the CMS-2 compiler action:

Instructions	AC 0
SYS-DD	AC 1

LOC-DD	AC 2
Auto-DD and LOC-DD W	AC 4
Constants and LOC-DD R	AC 5
Temps	AC 6
Inputlist/Outputlist	AC 7
Variable length table	AC 8-31

- When no pooling declaration or ACSEPARATION directive is present, the following specifies the CMS-2 compiler default action:

ENTITY	CS-NAME	AC-NAME	AC-NUMBER
SYS-DD	SYSDD	SYS-DD name	1
SYS-PROC	SYSP	SYS-PROC name	0
Auto data and temporary cells of SYS-PROC-REN	AUTODD	AUTO-DD name	4
LOC-DD R	CONST	LOC-DD R name	5
LOC-DD W	AUTODD	LOC-DD W name	4

= INCLUDE CONTROL CARD

The = INCLUDE Control Card shall cause input of all CMS-2 source card images from the specified element file. The = INCLUDE Control Card and all source card images from the included element file shall be included in the source listing if a source listing is specified.

Note that the = INCLUDE Control Card is not a feature of the CMS-2 language, but rather a part of the interface between compiler and operating system on the host machines.

The = INCLUDE Control Card format is:

= INCLUDE <internal file name>.<element name>

where: " = " (equals sign character) must be in column 11 (column 1 for COL1 option) followed by INCLUDE. No space is allowed between the " = " and INCLUDE.

<internal file name> is a 1 to 8 alphanumeric character internal file name that is associated with an element directory name when CMS-2 is invoked. The first character must be a letter.

<element name> is a 1 to 8 alphanumeric character name of the source element file in the specified element directory that contains the source to be included. The first character must be a letter.

= COMPOOL CONTROL CARD

The = COMPOOL Control Card specifies what Compool Elements are input to the compilation. A maximum of 127 = COMPOOL Control Cards are allowed. The cards must appear immediately following the OPTIONS declarations and prior to the declaration of any name other than the CMS-2 SYSTEM name.

Compool information is retrieved from the Compool Element at the point of each card. The Compool Element must have been the compool output of a previous CMS-2 compool compilation for the same family of target computers (32-bit ISA, 16-bit ISA, or 1750 ISA) as the target computer for the current compilation. If it is not, a fatal error message shall be given and the Compool Element shall be ignored.

The = COMPOOL Control Card format is:

= COMPOOL <internal file name>.<element name>

where: " = " (equals sign character) must be in column 11 (column 1 for COL1 option) followed by COMPOOL. No space is allowed between the " = " and COMPOOL.

<internal name> is a 1 to 8 alphanumeric character internal file name that is associated with an element directory name when CMS-2 is invoked. The first character must be a letter.

<element name> is a 1 to 8 alphanumeric character name of the compool element in the specified element directory that contains the compool element to be input. The first character must be a letter.

= TITLE CONTROL CARD

The TITLE Control Card shall cause a character string to be associated with a system element (SYS-DD or SYS-PROC) in its compiler listings and object element file. The = TITLE card can be situated in the major header following the OPTIONS statement or within a minor header. The major header = TITLE card shall designate the default character string and shall be used in the absence of minor header = TITLE cards. If no major header = TITLE card is present, the default shall be blank.

The Title Control Card format is:

= TITLE <character string>

where: " = " (equals sign character) must be in column 11 (column 1 for COL1 option) followed by TITLE. No space is allowed between the " = " and TITLE.

<character string> defines a character string of length 60 and is delimited by the " character (quotes). The " character may be included in the string by coding two " characters in sequence; the pair will be treated as a single " character. Strings longer than 60 will be truncated and strings shorter than 60 will be blank filled on the right. The allowable characters are the 96-character ASCII subset.

Note that the Title Control Card is not a feature of the CMS-2 language, but rather a part of the interface to the CMS-2 compiler.

= NOTES CONTROL CARD

= NOTES <character string>

where: " = " (equals sign character) must be in column 11 followed by NOTES. No space is allowed between the " = " and NOTES.

<character string> defines a character string of length 60 and is delimited by the " character (quotes). The " character may be included in the string by coding two " characters in sequence; the pair will be treated as a single " character. Strings longer than 60 will be truncated and strings shorter than 60 will be blank filled on the right. The allowable characters are the 96-character ASCII subset

CMS-2 RESERVED WORDS

Certain symbols that are language keywords in CMS-2 are reserved words and may not be used as names to identify entities in a CMS-2 program. With the exception of single letter reserved words (D, H, O or X), if any of these reserved words are used in a CMS-2 source program, a fatal error message will be given. Single letter reserved words will be allowed as names except for tables, item-areas, and functions.

CMS-2 RESERVED WORDS

ABS	DATAPOOL	FILE	MEANS	READ	USING
ALG	DEBUG	FIND	MEDIUM	REGS	VALID
AND	DECODE	FOR	MODE	RESUME	VARY
BASE	DEFID	FORMAT	NITEMS	RETURN	VARYING
BEGIN	DENSE	FROM	NONE	SAVING	VRBL
BIT	DEP	FUNCTION	NOT	SET	WHILE
BY	DIRECT	GOTO	O	SHIFT	WITH
CAT	DISPLAY	GT	OCM	SNAP	WITHIN
CHAR	ELSE	GTEQ	ODDP	SPILL	X
CHECKID	ELSIF	H	OPEN	STOP	XOR
CIRC	ENCODE	HEAD	OPTIONS	SWAP	
CLOSE	END	IF	OR	SWITCH	
CMODE	ENDFILE	INDIRECT	OUTPUT	SYSTEM	
COMMENT	EQ	INPUT	OVERFLOW	TABLE	
COMP	EQUALS	INTO	OVERLAY	THEN	
CORAD	EVENP	INVALID	PACK	THRU	
CORRECT	EXCHANGE	LIBS	PRINT	TO	
CSWITCH	EXEC	LOG	PTRACE	TRACE	
D	EXIT	LT	PUNCH	TYPE	
DATA	FIELD	LTEQ	RANGE	UNTIL	

The single-letter symbols A, B, F, I, P, S, U, and V are used as terminal symbols of the language in certain contexts but are not reserved words of the language. When these symbols are used in the context in which they are defined as terminal symbols, the terminal symbol meaning is used. In all other contexts, these symbols are considered to be names and can be used wherever names are allowed.The contexts in which these symbols are terminal symbols are as follows:

A, B, F, I, S, U - type descriptor
A, V - table declaration
P - preset indicator

Scope of Names:

There are four levels of scope in CMS-2: universal, global, local, and procedure (subprogram). The universal scope is a scope that contains every CMS-2 program. Universal scope names represent predefined compiler functions and procedures. These names can be used to reference the predefined specified functions or procedures if not overridden by a user declaration. If a user program contains a declaration for any of these names, the predefined meaning is overridden from the point of the declaration to the end of the scope of the declaration and the user defined attributes will be used from the point of the user declaration to the end of the scope of the user declaration.

The predefined universal scope names are:

*ACOS, *ACOS2, *ALOG, ANDF, *ASIN, *ASIN2, *ATAN, *ATAN2, *BAMS, CNT, COMPF, CONF, *COS, *EXP, *FIL, FIRST, *HLN, *ICOS, *IEXP, *ISIN, *ISQRT, LAST, *LENGTH, *LN, ORF, *POS, PRED, *RAD, *ROTATE, *ROTATEH, *ROTATEHP, *ROTATEP, REM, SCALF, SUCC, *SIN, *SQRT, *TAN, TDEF, *VECTOR, *VECTORH, *VECTORHP, *VECTORP, and XORF

*Names marked with an asterisk are predefined only for certain target computers.

RUN-TIME LIBRARY ROUTINES

16-BIT RUN-TIME ROUTINES

16-BIT NUMBER CONVERSION ROUTINES

LINKING CONVENTION

```
LK      R5,PACKET
JLR     R4,ROUTINE
```

	15	7	3	0
PACKET	SCALE FACTOR		IN	OUT

PACKET is the packet address
SCALE FACTOR is one byte long
IN and OUT are register numbers

ROUTINE DESCRIPTION	INPUT	OUTPUT
CSS$F Converts scaled single-length fixed-point number to floating-point number	R(IN)	R(OUT) R(OUT) + 1
CSD$F Converts scaled double-length fixed-point number to floating-point format	R(IN) R(IN) + 1	R(OUT) R(OUT) + 1
CSQ$F Converts scaled quad-length fixed-point number to floating-point format	R(IN) R(IN) + 1 R(IN) + 2 R(IN) + 3	R(OUT) R(OUT) + 1
CF$SS Converts floating-point format to scaled single-length fixed-point number	R(IN) R(IN) + 1	R(OUT)
CF$SD Converts floating-point format to scaled double-length fixed-point number	R(IN) R(IN) + 1	R(OUT) R(OUT) + 1

CF$SQ
Converts floating-point format to scaled quad-length fixed-point number
	R(IN)
	R(IN + 1)
		R(OUT)
		R(OUT) + 1
		R(OUT) + 2
		R(OUT) + 3

16–BIT MATH RUN–TIME ROUTINES

LINKING CONVENTION

```
LK      R5,PACKET
JLR     R4,ROUTINE
```

	15	7	3	1	0
PACKET	OUT		OP1	OP2	
	OPERAND 1				
	OPERAND 2				

PACKET is the packet address
OUT is one byte and indicates output register
OP1 and OP2 control bits are two bits long

Control bit meaning:

0 – Indirect address
1 – Direct address
2 – Register number
3 – Number (constant)

The packet indicates the location of input and output. Consecutive locations are used if more than one word. These routines are for computers without MATHPAC. The result is normalized.

ROUTINE DESCRIPTION

F$ADD
Adds a floating-point number to a floating-point number with the result a floating-point number

F$SUB
Subtracts a floating-point number from a floating-point number with the result a floating-point number

F$MUL
Multiplies a floating-point number with a floating-point number with the result a floating-point number

F$DIV
Divides a floating-point number by a floating-point number with the result a floating-point number

F$COM
Compares a floating point-number with a floating-point-number setting the condition code

P$II
Calculates X^y where X and Y are single-length fixed-point numbers with the result a single-length fixed-number

P$RI
For computers without ALOG and EXP instructions. Calculates X^y where X is a floating-point number and Y is a single-length fixed-point number with the result a floating-point number

ROUTINE DESCRIPTION

P$RIM
For computers with ALOG and EXP instructions. Calculates X^y where X is a floating-point number and Y is a single-length fixed-point number with the result a floating-point number

P$RR
For computers without ALOG and EXP instructions. Calculates X^y where X and Y are floating-point numbers with the result a floating-point number

P$RRM
For computers with ALOG and EXP instructions. Calculates X^y where X and Y are floating-point numbers with the result a floating-point number

16–BIT ROUTINES FOR FLOATING–POINT INTRINSICS

LINKING CONVENTION

```
LK      R5,PACKET
JLR     R4,ROUTINE
```

		7	3	0
PACKET		IN	OUT	

PACKET is the packet address
IN and OUT are register numbers
These routines are for computers other than AN/UYK–44 with MATHPAC and AN/AYK–SCP

ROUTINE DESCRIPTION

F$SINM
Calculates the sine of a floating-point input in radians with the result a floating-point number

F$COSM
Calculates the cosine of a floating-point input in radians with the result a floating-point number

ROUTINE DESCRIPTION

F$TANM
Calculates the tangent of a floating–point input in radians with the result a floating–point number

F$ASINM
Calculates the arc sine of a floating–point input with the result a floating–point number in radians

F$ACOSM
Calculates the arc cosine of a floating–point input with the result a floating–point number in radians

F$ATANM
Calculates the arc tangent of a floating–point input with the result a floating–point number in radians

F$EXPM
Calculates E^X where input and output are floating–point numbers

F$ALOGM
Calculates LN(X) where input and output are floating–point numbers

16–BIT CHARACTER AND BIT LOAD AND STORE ROUTINES

LINKING CONVENTION

```
LK      R5,PACKET
JLR     R4,ROUTINE
```

	15	12	5	3	1
PACKET	REG		N	S	A
	ADDRESS				
	START CHARACTER				
	NUMBER OF CHARACTERS				

REG is four bits long. N, S and A are two bits

REG	–	Register to load into or store from
N	–	Control bits for number of characters
S	–	Control bits for start character
A	–	Control bits for characters

Control bit meaning

A

0 – Indirect address
1 – Register address is in
2 – Direct address

S and N

0 – Address that contains value
1 – Register value is in
2 – Value

ROUTINE DESCRIPTION

LD$BIT
Loads bits from memory into a register

LD$BITD
Loads bits from memory into a double register

ST$BIT
Stores bits from a register into memory

ST$BITD
Stores bits from a double register into memory

LD$CHAR
Loads characters from memory into a register

LD$CHRD
Loads characters from memory into a double register

ST$CHAR
Stores characters from a register into memory

ST$CHRD
Stores characters from a double register into memory

16-BIT RUN-TIME ROUTINES

16-BIT MOVE, SWAP AND COMPARE CHARACTERS

LINKING CONVENTION

```
LK      R5,PACKET
JLR     R4,ROUTINE
```

	15	13	11	9	7	5	3	1
PACKET	A		B	C	D	E	F	G
	OP1 BASE ADDRESS							
	OP1 START CHARACTER							
	OP1 NUMBER OF CHARACTERS							
	OP2 BASE ADDRESS							
	OP2 START CHARACTER							
	OP2 NUMBER OF CHARACTERS							

A

0 - Use shortest operand length for number of characters in this operation
1 - OP1 is a constant
OP2 is longer
2 - OP2 is a constant
OP1 is longer

B, C, E & F

0 - Memory address
1 - Register
2 - Value

G	OP1 BASE
F	OP1 START
E	OP1 NUMBER
D	OP2 BASE
C	OP2 START
B	OP2 NUMBER

D & G

0 - Indirect address
1 - Register address is in
2 - Direct address

ROUTINE DESCRIPTION

MOV$CHR
Moves a character string from one memory location to another

COM$CHR
Compares one character string with another setting the condition code to indicate results of compare

SWP$CHR
Swaps character strings in memory

MS1750A RUN-TIME ROUTINES

MS1750A NUMBER CONVERSION ROUTINES

LINKING CONVENTION

```
LIM     R4,PACKET+1
SJS     R4,ROUTINE
```

	15	7	3 0
PACKET	RETURN ADDRESS		
	SCALE	IN	OUT

IN and OUT are register numbers

ROUTINE DESCRIPTION

CFL$SC
Converts a floating-point number to a single-length scaled fixed-point number

CFL$SCD
Converts a floating-point number to a double-length scaled fixed-point number

CSC$FL
Converts a single-length scaled fixed-point number to a floating-point number

CSC$FLD
Converts a double-length scaled fixed-point number to a floating-point number

MS1750A EXPONENTIATION ROUTINES

LINKING CONVENTION

```
        LIM     R4,PACKET+1
        SJS     R4,ROUTINE
```

	15	11	7	3
PACKET	RETURN ADDRESS			
	NIBBLE1	NIBBLE2	NIBBLE3	NIBBLE4

ROUTINE DESCRIPTION

POW$II

Computes X^y where X and Y are single length.
X is in R(NIBBLE3) Y is in R(NIBBLE2)
Result is single length in R(NIBBLE4)

POW$RI

Computes X^y where X is a floating-point number in R(NIBBLE3) and R(NIBBLE3+1) and Y is a single-length number in R(NIBBLE2). Result is a floating-point number in R(NIBBLE4).

POW$RR

Computes X^y where X is a floating-point number in R(NIBBLE3) and R(NIBBLE3+1) and Y a floating-point number in R(NIBBLE2) and R(NIBBLE2+1). Result is a floating-point number in R(NIBBLE4) and R(NIBBLE4+1).

1750A INTRINSIC ROUTINES

LINKING CONVENTION

```
        LIM     R4,PACKET+1
        SJS     R4,ROUTINE
```

	15	7	3 0
PACKET	RETURN ADDRESS		
		IN	OUT

Note: If IN/OUT is 15, then IN+1/OUT+1 is 0.

ROUTINE DESCRIPTION

F$SINA

Calculates the sine of an angle in radians. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$COSA

Calculates the cosine of an angle in radians. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$TANA

Calculates the tangent of an angle in radians. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$ALOGA

Calculates LN(X). Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$EXPA

Calculates EXP(X). Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$ASINA

Calculates arc sine. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$ACOSA

Calculates arc cosine. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$ATANA

Calculates arc tangent. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

F$SQRTA

Calculates square root. Floating-point input in R(IN) and R(IN+1). Floating-point output in R(OUT) and R(OUT+1).

MS1750A LOAD AND STORE CHARACTER ROUTINES

LINKING CONVENTION

```
    LIM     R4,PACKET+1
    SJS     R4,ROUTINE
```

	15 12		5	3	1
PACKET	RETURN ADDRESS				
	REG		N	S	A
	ADDRESS				
	START				
	NUMBER				

REG is four bits long. N, S and A are two bits.

REG	–	Register to load into or store from
N	–	Control bits for number of characters
S	–	Control bits for start character
A	–	Control bits for characters

Control bit meaning

A

0 – Indirect address
1 – Register address is in
2 – Direct address

S and N

0 – Address that contains value
1 – Register value is in
2 – Value

ROUTINE DESCRIPTION

LOD$BIT
Loads bits from memory into a register

LOD$BITD
Loads bits from memory into a double register

STO$BIT
Stores bits from a register into memory

STO$BITD
Stores bits from a double register into memory

LOD$CHAR
Loads characters from memory into a register

LOD$CHRD
Loads characters from memory into a double register

STO$CHAR
Stores characters from a register into memory

STO$CHRD
Stores characters from a double register into memory

1750A CHARACTER MOVE, COMPARE AND SWAP ROUTINES

LINKING CONVENTION

```
    LIM     R4,VALUE+1
    SJS     R4,ROUTINE
```

	15	13	11	9	7	5	3	1
PACKET	A		B	C	D	E	F	G
	OP1 BASE ADDRESS							
	OP1 START CHARACTER							
	OP1 NUMBER OF CHARACTERS							
	OP2 BASE ADDRESS							
	OP2 START CHARACTER							
	OP2 NUMBER OF CHARACTERS							

A

0 - Use shortest operand length for number of characters in this operation
1 - OP1 is a constant
OP2 is longer
2 - OP2 is a constant
OP1 is longer

B, C, E & F

0 – Memory address
1 – Register
2 – Value

G	OP1 BASE
F	OP1 START
E	OP1 NUMBER
D	OP2 BASE
C	OP2 START
B	OP2 NUMBER

D & G

0 – Indirect address
1 – Register address is in
2 – Direct address

ROUTINE DESCRIPTION

MOV$CHAR
Moves a character string from one memory location to another

COM$CHAR
Compares one character string with another setting the condition code to indicate results of compare

SWP$CHAR
Swaps character strings in memory

MS1750A COUNT BITS AND PARITY ROUTINES

LINKING CONVENTION

```
        LIM        R4,PACKET+1
        SJS        R4,ROUTINE
```

	15	7	3 0
PACKET	RETURN ADDRESS		
		IN	OUT

ROUTINE DESCRIPTION

CNT$

Counts the number of bits in R(IN) and puts the count in R(OUT)

ODD$P

Checks parity in R(IN). If odd result is 1 else 0. Result is placed in R(OUT)

EVEN$P

Checks parity in R(IN). If even result is 1 else 0. Result is placed in R(OUT)