CONTROL DATA CORPORATION
Interoffice Memorandum

DATE: June 3, 1983

TO: NOS Section

FROM: G. A. Kersten

SUBJECT: NOS Coding Standards

Attached is the new "NOS COMPASS Programming Standard" and the new "NOS SYMPL Coding Standard" for the NOS Development Section. These versions replace all previous versions of the standards.

Many additions, changes and clarifications have been included from suggestions received from the Design Support Team (DST) and from others during the preparation of these revisions. Change bars in the right hand columns mark the majority of the changes (some minor wording changes are not marked).

Major changes in the COMPASS standard include:

* Documentation of the PP instruction and jump macro usage (MJP, PJP, ADK, LDK, etc.).
* Suggestions for hangs and error processing to assist in debugging.
* Description of the new ISTORE macro and its use for instruction modification.
* Use of BSS vs BSSN.
* Use of "SYSTEM XXX,=" and "EXECUTE XXX,=" macros for cross reference purposes.
* Reservation of tags beginning with "U" for installation use.
* Clarification of the use of REQUIRES line in a modset.
* Additional interface considerations.
* A new Appendix - "DOCUMENTATION/USABILITY GUIDELINES".

Major changes in the SYMPL standard include:

* Deleting the MSF Project Addendum and incorporating many of the items into the standard.

* Addition of the Screen Management Facility (SMF) Project Addendum.

Any questions or comments should be addressed to the Design Support Office (DSO) or members of the DST. Requests for changes to the standards should follow the procedure defined in the NOS Section Procedures Notebook. Several suggestions were made as a result of section code review and resulting changes will be incorporated into the next revision of the standards.

G. A. Kersten

G. A. Kersten
Design Support Team

NOS-DEV/4926G/smb

NCS COMPASS PROGRAMMING STANDARD

TABLE OF CONTENTS

1.0 INTRODUCTION 6
1.1 SCOPE OF DOCUMENT 6
1.1.1 PURPOSE 6
1.1.2 SCOPE OF USE 6
1.2 CONFORMANCE AND ENFORCEMENT 6

2.0 DOCUMENTATION 7
2.1 INTRODUCTION 7
2.1.1 DESIGN OVERVIEW 7
2.1.2 EXTERNAL INTERFACE 7
2.1.3 INTERNAL OPERATION 8
2.1.4 DETAILED CODE ANALYSIS 8
2.2 GENERAL REQUIREMENTS 9
2.2.1 ABBREVIATIONS 9
2.2.2 PUNCTUATION 9
2.2.3 FORMATS 10
2.2.4 FORMAT OF ITEMIZED DOCUMENTATION 11
2.3 PROGRAM LEVEL DOCUMENTATION 11
2.3.1 OVERVIEW 11
2.3.2 EXTERNAL 12
2.3.3 INTERNAL 12
2.4 SUBROUTINE LEVEL DOCUMENTATION 13
2.4.1 ENTRY CONDITIONS (ENTRY) 14
2.4.2 EXIT CONDITIONS (EXIT) 14
2.4.3 ERROR EXIT CONDITIONS (ERROR) 14
2.4.4 REGISTER OR DIRECT CELL USAGE (USES) 15
2.4.5 ROUTINES CALLED (CALLS) 16
2.4.6 COMMON OR SYSTEM DECKS REQUIRED (XREF) 16
2.4.7 MACROS CALLED (MACROS) 16
2.4.8 ALLOCATED REGISTERS OR DIRECT CELLS (DEFINE) . . . 16
2.4.9 TIMING CONSIDERATIONS (TIMING) 16
2.4.10 PROGRAMMING NOTES (NOTES). 16
2.5 CODE LEVEL DOCUMENTATION 17
2.5.1 STAND-ALONE COMMENTS 17
2.5.2 EMBEDDED COMMENTS 17
2.6 MACRO LEVEL DOCUMENTATION 18
2.6.1 ENTRY CONDITIONS (ENTRY) 19
2.6.2 EXIT CONDITIONS (EXIT) 19
2.6.3 REGISTER OR DIRECT CELL USAGE (USES) 20
2.6.4 ROUTINES CALLED (CALLED) 20
2.6.5 MACROS CALLED (MACROS) 20
2.7 DOCUMENTATION EXAMPLES 20
2.7.1 PROGRAM LEVEL 20
2.7.2 SUBROUTINE LEVEL (PP CODE) 21
2.7.3 SUBROUTINE LEVEL (CP CODE) 22
2.7.4 MACRO LEVEL DOCUMENTATION 23

3.0 CODING 24
3.1 CARD/LINE LAYCUTS 24
3.1.1 COMPASS CPERATION CARDS 24
3.1.2 COMPASS COMMENT CARDS 24
3.2 PROGRAM LAYCUT 24
3.2.1 GROUP 1 INSTRUCTIONS 25
3.2.1.1 PERIPHERAL PROCESSCR PROGRAMS 25
3.2.1.2 CENTRAL PRCCESSOR PROGRAMS 25
3.2.1.3 COMMON DECKS 26
3.2.2 PROGRAM LEVEL DOCUMENTATICN 26
3.2.3 MACRO DEFINITIONS 26
3.2.4 INSTALLATICN SYMBOL DEFINITICNS 26
3.2.5 LOCAL SYMBCL DEFINITIONS 26
3.2.6 GLOBAL MEMCRY DEFINITIONS 27
3.2.7 MAIN LOOP 27
3.2.8 PRIMARY SUBRCUTINES 27
3.2.9 SECONDARY SUBROUTINES 27
3.2.10 WORKING STORAGE AND BUFFERS 27
3.2.11 INITIALIZATION CODE 27
3.2.12 PROGRAM TERMINATION 28
3.3 INSTRUCTION USE, FORMAT AND PARAMETERS 28
3.3.1 REGISTER USE AND SPECIFICATICN 28
3.3.1.1 B0 REGISTER USE 28
3.3.1.2 B1 REGISTER USE 28
3.3.1.3 PACK AND NCMINAL SHIFT X REGISTERS 29
3.3.1.4 UNPACK AND NORMALIZE X REGISTERS 29
3.3.2 MULTIPLE LCGICAL TESTS 29
3.3.3 SHIFT INSTRUCTION PARAMETERS 29
3.3.4 BOOLEAN MASK USAGE 31
3.3.5 RELATIVE ADDRESSING 31
3.3.6 JUMP INSTRUCTICN USE 31
3.3.7 SUBROUTINE ENTRY 32
3.3.8 CPU CODE CPTIMIZATICN 33
3.3.9 CLEARING PP MEMORY. 34
3.3.10 INSTRUCTICN MODIFICATICN 34
3.3.11 COMMON DECK REGISTER USAGE 37
3.3.12 PP ADK, LDK, LMK, LPK, SBK MACRC USAGE 37
3.4 DATA USE, FORMAT AND PARAMETERS 37
3.4.1 LITERALS 37
3.4.2 DATA FORMATS 37
3.4.3 TABLE GENERATION 38
3.4.4 DIRECT CELL USE 38
3.4.5 BUFFER DEFINITIONS 39

3.5 DATA/CODE NAMING TERMINOLOGY 39
3.5.1 USE OF CONDITION TERMINOLOGY 40
3.5.2 TAGS WITHIN SUBROUTINES 40
3.5.3 TAGS ON DATA 41
3.5.3.1 DIRECT CELLS 41
3.5.3.2 CODE CONTROL NAMES 41
3.5.3.3 TABLE NAMES 41
3.5.3.4 GLOBAL MEMORY LOCATIONS 42
3.5.4 CONSTANTS USED AS INSTRUCTIONS 42
3.5.5 IF/ELSE/ENDIF SYMBOLS 42
3.5.6 NON-LOCAL MACRO SYMBOLS 42
3.5.7 LOW CORE LOCATION SYMBOLS 43
3.5.8 CONTROL POINT AREA LOCATION SYMBOLS 43
3.5.9 MONITOR FUNCTION SYMBOLS 43
3.5.10 NEGATIVE FIELD LENGTH SYMBOLS 43
3.6 PSEUDO INSTRUCTION USE, FORMAT AND PARAMETERS 43
3.6.1 BASE AND POST RADIX USE 43
3.6.2 EXTERNAL REFERENCES 43
3.6.3 SPACE CARD FORMAT 44
3.6.4 CONDITIONAL CODE 44
3.6.5 MACROS 45
3.6.6 DIS PSEUDO INSTRUCTION 45
3.7 TESTS FOR OVERFLOW 46
3.7.1 CM LOADS 46
3.7.2 TABLE OVERFLOW 46
3.7.3 MASS STORAGE LOADS 47
3.7.4 OVERLAY LOADS 47
3.8 RELOCATABLE CPU CODE 47
3.9 ROUTINE/SUBROUTINE 48

4.0 MISCELLANEOUS 48
4.1 PROGRAM NAMING 48
4.1.1 LENGTH OF PROGRAM NAME 48
4.1.2 RESERVED NAMES 49
4.1.3 COMMON DECK NAMES 49
4.2 CODE TRANSMITTAL RULES 50
4.2.1 GENERAL RULES 50
4.2.2 MODSET FORMAT 50
4.2.2.1 MODSET IDENTIFIER 51
4.2.2.2 MODSET CORRECTION LETTER 52
4.2.2.3 OVERFLOW 52
4.2.2.4 MODSET EXAMPLE 52
4.3 INTERFACE CONSIDERATIONS 53
4.3.1 SYSTEM SUPPLIED INTERFACES 54
4.3.2 PARAMETER VALIDATION 54
4.3.3 MEMORY ACCESS 54
4.3.4 SECURITY 54
4.3.5 RESERVATIONS AND INTERLOCKS 54
4.3.6 DOCUMENTING HARDWARE DEFICIENCIES 55
4.3.7 NEW FUNCTION/LOW CORE IDENTIFIERS 55
4.3.8 DECK INTERDEPENDENCIES 55

4.4 MODULARITY 55
4.4.1 PP OVERLAYS 55
4.4.2 HELPER PP-S 56
4.4.3 COMMON DECKS 56
4.5 DAYFILE MESSAGES 56
4.6 UNHANGABLE CHANNEL CODE 57
4.7 SPECIAL ENTRY POINTS 57
4.8 SCRATCH FILE NAMES 58

APPENDIX A - ABBREVIATIONS 59
A.1 GENERAL ABBREVIATIONS 59
A.2 NETWORK HOST PRODUCTS ABBREVIATIONS 60
A.3 ACRONYMS 61

APPENDIX B - ERROR MESSAGE GUIDELINES 62

APPENDIX C - DOCUMENTATION/USABILITY GUIDELINES 64

1.0 INTRODUCTION

1.1 SCOPE OF DOCUMENT

1.1.1 PURPOSE

This document establishes standard procedures to be used by programmers in the development and maintenance of programs in COMPASS Assembly Language.

This document should be used by programmers as a reference manual of standard programming procedures. The implementation of these procedures will increase the efficiency of program development, improve the reliability and maintainability of the program and aid in the training of persons who will be maintaining or using the program.

1.1.2 SCOPE OF USE

The procedures defined in this document are applicable to the NETWORK OPERATING SYSTEM (NOS) and its related subsystems.

1.2 CONFORMANCE AND ENFORCEMENT

The procedures defined in this document are to be used in all newly developed programs. In existing programs these procedures should be used if they are not inconsistent with the existing procedures. If major changes (such as a rewritten subroutine or a new overlay) are made to a program, these procedures are to be used for the changes.

Programs which do not conform to these requirements will be returned to the programmer for correction.

2.0 DOCUMENTATION

2.1 INTRODUCTION

Documentation is the presentation of information about a program in easily understandable form so that those who need to understand the program do not need to study the program itself. This reduces the time consumed from days or weeks to just minutes or hours.

The program documentation presented in this standard is embedded within the source language of each program. The documentation is designed to be extractable by the standard documentation processing program, DOCMENT. This approach serves to unify the program and its documentation, making it easier and more natural to update the documentation as changes are made to the code.

The effectiveness of documentation is judged by its success in meeting the needs of those who use it. This introduction defines four distinct needs for documentation which arise during the life of a software product. Sections 2.3 through 2.5 define three levels of documentation (Program Level, Subroutine Level and Code Level) which together meet these needs.

Section 2.7 gives examples of proper documentation.

2.1.1 DESIGN OVERVIEW

Anyone wanting to know the structure of the system, or some functional area, does not want excessively detailed information which will make the task more difficult. One should be able to ask the question:

What is the function of this program?

and get an answer that is brief and to the point. It should not contain any information about the input parameters, options, error conditions, or internal workings of the program.

2.1.2 EXTERNAL INTERFACE

Anyone who knows the function of a program and wants to know how to interface to the program needs to know the form of the call, what parameters to supply, what information is returned, and what is accomplished. One should be able to ask the question:

How is this program used?

and get an answer that lists the parameter definitions, formats, and contents, the initial conditions of buffers and devices, any status and condition information, a list of other programs called, and a complete list of error codes, error messages issued and parameters returned. A general description of the actions taken should be included for each function performed that is recognizable by the calling program.

2.1.3 INTERNAL OPERATION

Anyone working on a modification or enhancement to the system needs a general knowledge of the internal operation of a program. This requires finding out where within the program some function is performed and how it is performed. One should be able to ask the question:

How does this program work?

and get an answer that includes a description of the logical flow and structure of the program, the algorithms used and the function performed by each overlay or subroutine in the program.

2.1.4 DETAILED CODE ANALYSIS

Anyone attempting to modify the program or establish a knowledge of the detailed operation of a program uses the listing. Documentation should be provided in the listing to aid in following the flow of the program without reading all of the code. This documentation consists of comments within the code itself. Comments describing the function of logical groups of instructions should be provided, and comments documenting table structures, data areas, and constants should appear on the instructions which define them. One should be able to ask the question:

What should I know when modifying this program?

and get back an answer with all the detail needed to make the modifications without adversely affecting existing program functions.

2.2 GENERAL REQUIREMENTS

2.2.1 ABBREVIATIONS

The abbreviations for technical terms which are to be used in program documentation are listed in Appendix A. All other technical words and phrases are completely spelled out. Routine names and mnemonic names of tables and equipments are not considered abbreviations.

A program whose documentation makes extensive use of terms not in this list may define a list of abbreviations and include it in the first section of the internal documentation for the program. Such abbreviations may not be used in the program overview or external documentation.

The following standard symbols are used in the documentation when expressing logical and arithmetic comparisons:

.NOT.	logical inverse
.XOR.	logical difference (exclusive or)
.AND.	logical product
.OR.	Logical sum (inclusive or)
.EQ.	equal to
.NE.	Not equal to
.LE.	Less than or equal to
.GE.	Greater than or equal to
.LT.	Less than
.GT.	Greater than
=	equal to
()	contents of
(())	contents of the contents of (indirect addressing)

2.2.2 PUNCTUATION

All documentation and comment lines contain complete English sentences with correct punctuation. Exceptions are allowed in subroutine headings (see section 2.4) and in embedded comments (see section 2.5.2). Titles (such as "Assembly Constants.") should end with a period but need not be complete sentences. Each comment (excluding embedded comments) should end with a period even if it is not a sentence.

Correct punctuation means the same punctuation as required in written English. However, the apostrophe presents problems due to character set and print train differences, and plurals of abbreviations are not readable without upper and lower case. Therefore, plurals and possessive forms of abbreviated terms are to be avoided. Authorized abbreviations (see section 2.2.1) are made plural by adding a hyphen and the letter "s" (e.g. PP-S).

If an upper case item is to be indicated in the documentation, it is enclosed within asterisks. Upper case is used for names of files, programs, calling parameters, subroutine tags, table names and any other words that are normally capitalized . Accepted abbreviations, acronyms, and programming language names are not enclosed in asterisks even if they would normally be capitalized (refer to Appendix A and section 2.2.1). Tables defined in CMR do not need asterisks (EJT, QFT, etc.).

2.2.3 FORMATS

Documentation lines contain only asterisks and blanks in the first ten columns and text in columns 11 through 71. The text is written using correct English except where specifically noted. The format for each of the various types of documentation is shown below. Later sections define the usage of each type of documentation.

External - Asterisks in columns 1 through 3 (***). This line indicates that this and the following contiguous comment lines are to be included in the program external documentation.

Internal - Asterisks in columns 1 and 2 (**). This line indicates that this and the following contiguous comment lines are to be included in the program internal documentation.

Internal Bracket - Asterisks in columns 1 through 4 (****). This line and all other lines up to and including the next occurrence of this line are to be included in the program internal documentation.

Other Comment - Asterisk in column 1 (*). This type of comment is used for continuation of the above types of documentation. It is also used for stand-alone comments (see section 2.5.1).

Table - Asterisk in column 1, T in column 2 (*T). This line indicates the beginning of table documentation. The *T lines must appear within consecutive comment lines beginning with an external (***) or internal (**) statement. DOCMENT generates a table diagram from the field widths and descriptions in the *T documentation. Note that the field description size (including blanks) cannot exceed the field width size (to prevent truncation). A DOCMENT listing should be made of a program's external/internal documentation when table structures are added (to verify correctness).

Table Continuation - Asterisk in column 1, T in column 2, comma in column 3 (*T,). This type of comment is used for continuation of table documentation started by a *T line.

Blank Comment Line - A line with an asterisk in column 1 (*) and blanks in columns 2 through 71 is called a "Blank Comment Line or "Blank Comment Card" and is used as a separator to improve readability of documentation.

2.2.4 FORMAT OF ITEMIZED DOCUMENTATION

Documentation which contains several separate items of information (as found in sections 2.3.1, 2.3.2, 2.3.3 and 2.4) contains a Blank Comment Line between the items. Each item ends with a period. If an item is not applicable,it is omitted from the documentation. The items are placed in the order specified and the last item is followed by a SPACE 4,10 line.

2.3 PROGRAM LEVEL DOCUMENTATION

Every program contains comment lines which make up the Program Level documentation (as defined in this section). This level of documentation may be used with the program listing or without it (using extracted documentation produced by a documentation processor). Even without the listing, the Program Level documentation satisfies the Design Overview, External Interface and Internal Operation documentation needs discussed in section 2.1.

2.3.1 OVERVIEW

The overview documentation is placed immediately following the COMPASS Group 1 instructions and before any other documentation, macro definitions or executable code (see section 3.2). It consists of an external documentation line (see section 2.2.3) which contains the program name and a brief description of the program, and additional comment lines which contain the following items of data (see section 2.7 for the layout of these items):

- Name of author and date written (yy/mm/dd).
- Names of authors of major modifications, with dates.
- Text of overview of program.

The text of the overview should follow the general definition of Design Overview documentation in section 2.1. The objective is to describe the function of the program in general terms.

2.3.2 EXTERNAL

The external documentation is placed immediately following the overview documentation and before any internal documentation, macro definitions or executable code (see section 3.2). It consists of an external documentation line (see section 2.2.3) and additional comment lines which together contain the following items of data (see section 2.7 for the layout of these items):

- Detailed description of functions and options.
- Entry conditions, including parameters and initial conditions of buffers and external tables.
- Command format.
- Exit conditions, including status bits and fields returned.
- Errors detected, error codes returned, including subsequent action taken for each.
- System errors detected and subsequent action taken.
- Other programs called.
- Messages issued (including dayfile and operator).

The content of this section follows the general definition of External Interface documentation in section 2.1. The objective is to supply information required by a potential user of the program.

2.3.3 INTERNAL

The internal documentation describes the internal workings of the program. It may be dispersed throughout a program as desired, however a major portion appears immediately following the external documentation. Internal documentation consists of an internal documentation line (see section 2.2.3) and additional comment cards which together contain the following items of data (see section 2.2.4 for the layout of these items):

- System texts required for assembly (other than default).
- Direct cell usage (PP programs).
- Global register assignments (CPU programs).
- Data areas and table formats.
- Memory map (if overlays are used).

Other items to be included if applicable are:

- Techniques or algorithms employed where not obvious.
- Timing considerations.
- Interlock considerations.
- Known limitations to performance or extensibility, such as timing of loops, core size, error-recovery deficiencies.

Any other information which would aid someone in understanding the internal workings of the program is also provided, including logical flow, structure, and pitfalls to redesign.

2.4 SUBROUTINE LEVEL DOCUMENTATION

The heading of any subroutine consists of comment lines giving a brief description of its function, its entry and exit conditions, register or direct cell usage and internal workings. The information contained should be on a level indicated by the complexity of the subroutine. The following items of data should be included (see sections 2.7.2 and 2.7.3 for the layout of these items):

- TITLE line with name as subtitle (primary subroutine)
- SPACE line (see section 3.6.3)
- Internal comment line giving name and title of subroutine
- One or more sentences describing the function of the subroutine (optional but desirable).
- Entry conditions (list)
- Exit conditions (list)
- Error exit conditions (list)
- Register or direct cell usage (list)
- Routines called (list)
- Macros called (list)
- Description of allocated registers
- Timing considerations, if critical
- Design, implementation and general information
- Two blank lines

The title of the subroutine should describe the action performed by the subroutine (for example, Position Mass Storage, Make Queue Entry). This means that titles should always contain a verb. Titles without verbs should be used for groups of subroutines and COMSxxx decks.

Defined formats exist for the list of items in the subroutine heading. A keyword appears in column 11, followed by text in column 18. The text is simply a list, rather than complete sentences. Any list requiring more than one line is continued beginning in column 18 (or beyond) of the next comment line. The formats are shown below. Each list ends with a period. Acceptable keywords include:

ENTRY	Entry conditions.
EXIT	Exit conditions.
ERROR	Error exit conditions.
USES	Register or direct cells destroyed.
CALLS	Routines called.
XREF	Common or system decks required.

MACROS	Macros called.
DEFINE	list of allocatable registers.
TIMING	Description of timing considerations.
NOTES	Programming information.

Documentation for common decks and zero level overlays should include subroutine level documentation where appropriate.

2.4.1 ENTRY CONDITIONS (ENTRY)

Entry conditions include the following items:

Registers, direct cells or memory locations that must be set before the subroutine is called.

Logical status of channels, files, etc.,(i.e. Channels reserved, files set busy, disk postioned, files positioned) that should exist before the subroutine is called.

Only one entry condition may be specified per line. For PP code, the contents of the A register should be described first.

2.4.2 EXIT CONDITIONS (EXIT)

Exit conditions include the following items:

Registers, direct cells or memory locations that may be used by subsequent routines.

Logical status of channels, files, etc., (i.e. Channels reserved, files set busy, disk positioned, files posticned) that exist when the subroutine is exited.

Only one exit condition is listed per line. For PP code, the contents of the A register should be described first.

Branches (not CALLs) to other routines.

2.4.3 ERROR EXIT CONDITIONS (ERROR)

Error exit conditions include all exit conditions that exist when a special termination of the subroutine, such as a jump to an error processor, is taken. Error conditions include the following items:

- The label being jumped to and the conditions that caused the special exit.

- Registers, direct cells or memory locations that may be used by subsequent routines.

- Logical status of channels, files, etc., (i.e. Channels reserved, files set busy, disk positioned, files postioned) that exist when the subroutine is exited.

Only one exit condition is listed per line. The label being jumped to is documented first with all exit conditions pertaining to that exit following. If these are multiple error exits, each with unique conditions, the conditions should be listed under their own exits.

2.4.4 REGISTER OR DIRECT CELL USAGE (USES)

Registers or direct cells used include all registers or direct cells destroyed by that subroutine only. Registers or direct cells destroyed by subroutines or MACROS called by a routine are not listed.

For CPU code, the format of the USES block includes a register type (X - operand register, A - address register, B - index register) followed by a sequence of ascending numbers indicating the registers used. The term ALL may be substituted if all registers are used by a routine.

Example:

```
*         USES   X - 0, 1, 6.
*                A - 1, 6.
*                B - 3, 7.
```

Indicates the following registers are used:

X0, X1, X6, A1, A6, B3, B7.

For PP code, single direct cells are listed in alphabetical order followed by multiple direct cells listed in alphabetical order.

Example:

```
*         USES   T1, T2, T5, CM - CM+4, RI - RI+4.
```

The direct cell at PP memory location 0 (T0) is assumed to be used by every subroutine unless otherwise stated in the ENTRY/EXIT conditions. This is because certain instructions, such as CRM and CWM, destroy location zero.

2.4.5 ROUTINES CALLED (CALLS)

Routines called by a subroutine include all subroutines, common decks and overlays that are explicitly called. Routines called by macros are not to be included. Where the routine called does not return to the calling point, the branch is an EXIT condition rather than a CALL.

2.4.6 COMMON OR SYSTEM DECKS REQUIRED (XREF)

When a routine, macro, or common deck requires a particular common or system deck for assembly or for correct execution, it is valuable to list the deck(s) required.

Example:

```
*          XREF   COMCLOD, CCMCRTN.
```

2.4.7 MACROS CALLED (MACROS)

Macros called include all macros that are explicitly called by a subroutine. The SUBR macro is always implied to be called and need not be listed. PP macros ADK, IDK, IMK, IPK, SBK, MJP, NJP, PJP, UJP, and ZJP need not be listed.

2.4.8 ALLOCATED REGISTERS OR DIRECT CELLS (DEFINE)

Allocated registers are registers (or direct cells for PP code) used for well defined items throughout a particular subroutine or program.

2.4.9 TIMING CONSIDERATIONS (TIMING)

Timing considerations should describe any timing limitations that are imposed on the subroutine because of hardware or performance constraints. Care should be taken to express units of time in a manner independent of machine type. For example, units of time expressed in cycles rather than microseconds is more desirable. This is because the same routine may execute faster or slower than stated depending on the type of the PP or CPU it executes in.

2.4.10 PROGRAMMING NOTES (NOTES)

This section documents design, implementation and general information that may be useful to other analysts. Information in this section pertains to the subroutine only.

2.5 CODE LEVEL DOCUMENTATION

All documentation which is not described in one of the preceding sections of this document falls into the category of Code Level documentation. The requirements for this type of documentation vary widely, so few rules can be stated; however, Code Level documentation is necessary and the lack of explicit requirements must not lead to its neglect.

Code Level documentation, along with the subroutine headings, must satisfy the need for aid in reading the code. Its content follows the guidelines for Detailed Code Analysis in section 2.1.4.

2.5.1 STAND-ALONE COMMENTS

Stand-alone comments are comment lines appearing in-line with code, as opposed to within higher-level documentation previously defined. All stand-alone comments are preceded and followed by one blank line. To reduce the binary size of systems texts, stand-alone comments within macro definitions should be preceeded and followed by a blank comment line.

Stand-alone comments describe the function performed by the subsequent section of in-line code. The comments are complete English sentences with correct punctuation, ending with a period. The comments refer to functions and data in external terms, rather than only in octal numbers and bit positions. These comments follow the general requirements found in section 2.2.

2.5.2 EMBEDDED COMMENTS

Embedded comments are comments in columns 30-71 of a line assembled by COMPASS, not a comment line. The comment need not be a complete sentence and is not terminated with a period. This type of comment is never continued onto another line. If the intended comment is too long to fit on the single line, it is inserted as a stand-alone comment preceding the area of code to which it applies. Care should be taken not to overflow into column 72.

An embedded comment describes the function of the instruction or sequence of instructions on which it appears. (It must be at the beginning of the sequence.) It does not describe the hardware operation being performed, but rather its meaning in the context of the function to be performed by the program.

With the exception of extremely complex code, it should not be necessary to put embedded comments on every line. Frequently, it is advantageous to omit "obvious" or redundent comments, since it then becomes easier for the casual reader to scan the routine.

An embedded comment is required on each jump instruction, to identify the condition being tested (conditional jumps) or the action being taken (unconditional jumps). On jump instructions, the word "JUMP" is superfluous, and is not used. On conditional jumps, the comment must begin with the word "IF" and describes the condition on which the jump will be executed. These comments follow the general requirements found in Section 2.2.

An embedded comment is required on all pseudo tests (ERRNZ, ERRPL, etc.). The comment should state the condition for which the test fails and the word "IF" should not be used.

Example - ERRNG *-BUFAL CODE OVERFLOWS BUFFER AREA

2.6 MACRO LEVEL DOCUMENTATION

The heading of any macro definition consists of comment lines giving a brief description of its function, its entry and exit conditions, register or direct cell usage and internal workings. The information contained should be on a level indicated by the complexity of the macro. The following items of data should be included (see section 2.7.4 for the layout of these items):

- SPACE line (see section 3.6.3)
- Internal comment line giving name and title of macro
- One or more lines of text explaining the purpose and/or function of the macro (optional but desirable).
- Format of macro call
- Entry conditions (list)
- Exit conditions (list)
- Register or direct cell usage (list)
- Routines called (list)
- Macros called (list)
- Two blank lines

Defined formats exist for the list of items in the macro heading. A keyword appears in column 11, followed by text in column 18. The text is simply a list, rather than complete sentences. Any list requiring more than one line is continued beginning in column 18 (or beyond) of the next comment line. The formats are shown below. Each list ends with a period.

ENTRY	Entry conditions.
EXIT	Exit conditions.
USES	Register or direct cells destroyed.
CALLS	Routines called.
MACROS	Macros called.

2.6.1 ENTRY CONDITIONS (ENTRY)

Entry conditions may include the following items:

- Description of macro parameters that are allowed. Complete descriptions of macro parameters include:
  1. valid parameter options
  2. default values of parameters
  3. register optimization, if applicable

- Registers, direct cells or memory locations that must be set before the macro is called. Entry conditions may refer to the entry documentation found in subroutines called by the macro.

- Logical status of channels, files, etc.,(i.e. Channels reserved, files set busy, disk positioned, files positioned) that should exist before the macro is called.

Only one entry condition may be specified per line. For PP code, the contents of the A register should be described first.

2.6.2 EXIT CONDITIONS (EXIT)

Exit conditions include the following items:

- Registers, direct cells or memory locations that may be used by subsequent routines. Exit conditions may refer to exit conditions in subroutines called by the macro.

- Logical status of channels, files, etc., (i.e. Channels reserved, files set busy, disk positioned, files positioned) that exist when the code generated by the macro is exited.

- Special terminations of the macro such as jumps to error processors or to any other routines. The label being jumped to and the conditions that cause the special exit should be documented.

Only one exit condition is listed per line. For PP code, the contents of the A register should be described first.

2.6.3 REGISTER OR DIRECT CELL USAGE (USES)

Registers or direct cells used include all registers or direct cells destroyed (or modified) by that macro only. Refer to section 2.4.4 for the format of the USES block.

2.6.4 ROUTINES CALLED (CALLS)

Routines called by a macro include all subroutines and overlays that are explicitly called. Routines called by macros within the macro definition are not to be included.

2.6.5 MACROS CALLED (MACROS)

Macros called include all macros that are explicitly called by a macro definition.

2.7 DOCUMENTATION EXAMPLES

These examples are statements of the standard and are intended as further clarification of the required procedures.

2.7.1 PROGRAM LEVEL

```
***       LIBEDIT - LIBRARY EDITING PROGRAM.
*
*          A. B. ORIGINAL.    74/01/01.
*          A. B. MODIFIER.    75/01/01.
*          C. D. MODIFIER.    76/01/01.
          SPACE  4,10
***       *LIBEDIT* IS A GENERAL PURPOSE FILE EDITING
*         PROGRAM CAPABLE OF MODIFYING AND GENERATING
*         LIBRARY FILES.
          SPACE  4,10
***       COMMAND FORMAT.
*         .
*         .
          .
          SPACE  4,10
***       DAYFILE MESSAGES.
*         .
*         .
          .
```

```
            SPACE  4,10
***         ACCOUNT FILE MESSAGES.
*           .
*           .
            .
            SPACE  4,10
***         ERROR LOG MESSAGES.
*           .
*           .
            .
            SPACE  4,10
***         OPERATOR MESSAGES.
*           .
*           .
            .
            SPACE  4,10
*           COMMON DECKS.
            .
            .
            .
            SPACE  4,10
*           MACRO  DEFINITIONS.
            .
            .
            .
****        ASSEMBLY CONSTANTS.

BUFL        EQU    1001B          OUTPUT BUFFER LENGTH
****
            SPACE  4,10
*           GLOBAL STORAGE.
            .
            .
            .
```

2.7.2 SUBROUTINE LEVEL (PP CODE)

```
            TITLE  ERROR PROCESSING ROUTINES.
LEM         SPACE  4,30
**          LEM - LIST ERROR MESSAGE.
*
*           *LEM* ISSUES ERROR MESSAGES TO THE JOB AND
*           SYSTEM DAYFILES AND TO THE ERROR LOG.
*
*           ENTRY  (A) = 1 IF SINGLE BIT SECDED ERROR.
*                      = 2 IF STATUS/CONTROL REGISTER
*                          ERROR LIMIT.
```

```
*                (SCRA - SCRA+20) = SCR IMAGE.
*                (NL) = ADDRESS OF NEXT LIST ENTRY.
*
*          EXIT  (NL) = UPDATED LIST POINTER.
*
*          ERROR TO *ERR* IF ERROR ENCOUNTERED.
*
*          USES  T1, T2, CM - CM+4, FN - FN+4.
*
*          CALLS LMC.
*
*          XREF  COMPABZ.
*
*          MACROS MONITOR.
*
*          DEFINE (T2) = FWA OF MESSAGE.
*
*          TIMING A DELAY IS NEEDED TO AVOID FILLING THE
*                 DISK WITH ERROR LOG MESSAGES.

LEM       SUBR               ENTRY/EXIT
          .
          .
          .
          UJN    LEMX        RETURN
```

2.7.3 SUBROUTINE LEVEL (CP CODE)

```
 ACS      SPACE  4,25
**        ACS -  ASSEMBLE CHARACTER STRING.
*
*         *ACS* ASSEMBLES A CHARACTER STRING INTO BUFFER
*         *CBUF*, PACKED 10 CHARACTERS PER CM WORD.
*
*         ENTRY  (B6) = FWA OF CHARACTER STRING.
*                (B7) = LENGTH OF STRING BUFFER.
*
*         EXIT   (CBUF - CBUF+20) = CHARACTER STRING.
*
*         ERROR  TO *ERR* IF INVALID CHARACTER FOUND.
*                (X1) = FWA OF ERROR MESSAGE.
*
*         USES   X - 0, 1, 6.
*                A - 1, 6.
*                B - 6, 7.
*
```

```
*          CALLS  MCI.
*
*          MACROS GETCH.
*
*          DEFINE (XO) = CHARACTER MASK.

 ACS       SUBR               ENTRY/EXIT
           .
           .
           .
           EQ     ACSX        RETURN
```

2.7.4 MACRO LEVEL DOCUMENTATION

```
 ERROR     SPACE  4,10
**         ERROR - ERROR PROCESSING MACRO.
*
*          ERROR  ADDR
*
*          ENTRY  *ADDR* = FWA OF DAYFILE MESSAGE.
*
*          EXIT.  TO *EPR*.
*
*          USES   X - 1.

           PURGMAC ERROR

ERROR      MACRO  A
           SX1    A
           EQ     EPR
ERROR      ENDM
```

3.0 CODING

3.1 LINE LAYOUTS

3.1.1 COMPASS OPERATION LINES

The following list of column numbers represent the beginning of each field in a COMPASS coding line. (An exception is allowed for macro definitions in system texts where space is critical. Refer to section 3.6.5.)

```
Column 1       = Continuation field (comma) if required
Column 2       = Location field
Column 11      = Operation field
Column 18      = Address field
Column 30      = Comment field
Column 73-80   = Reserved
```

If a field is full or overflows into an adjacent field, then two spaces should separate the fields. For readability, a block of comments can be aligned in a column past column 30. Column 72 of the comment field should be blank unless a continuation line is required.

3.1.2 COMPASS COMMENT LINES

The following list of column numbers represent the format of a comment line. A full description of where and how to use comment lines is found in section 2.

```
Column 1      = always contains an asterisk
Column 2-5    = (see section 2.2.3)
Column 6-10   = generally blank
Column 11-71  = contains the text of the comment
Column 72-80  = reserved
```

3.2 PROGRAM LAYOUT

The following sections define the components of a program in the order they appear within the program. It is not expected or required that every program will consist of all components described. In this discussion a "program" is a relocatable program unit (from "IDENT" to "END"), an entire absolute program or a common deck. A subroutine is a routine within a program.

3.2.1 GROUP 1 INSTRUCTIONS

Group 1 instructions appear at the beginning of each program and contain the identification and environment information for the program. The following examples define the layout of the Group 1 instructions for each type of program.

3.2.1.1 PERIPHERAL PROCESSOR PROGRAMS

```
1         11     18          30          (Columns)
+---------+------+-----------+-----

          IDENT  XXX,ORIGIN
          MACHINE             (optional)
          PERIPH
          NOLABEL             (deadstart routines)
          BASE   M
          LIST                (optional)
          SST
          TITLE  XXX - program description.
*COMMENT  deckname - description.
          COMMENT  COPYRIGHT CONTROL DATA CORPORATION, year.
 XXX      SPACE  4,10
```

3.2.1.2 CENTRAL PROCESSOR PROGRAMS

```
          IDENT  XXXXXXX,FWA program description
          ABS                 (optional)
          MACHINE             (optional)
          LCC                 (optional)
          SST                 (optional)
          ENTRY  YYYY         (optional)
          SYSCOM B1
          LIST                (optional)
          TITLE  XXXXXXX - program description.
*COMMENT  deckname - description.
          COMMENT  COPYRIGHT CONTROL DATA CORPORATION, year.
 XXX      SPACE  4,10
```

3.2.1.3 COMMON DECKS

```
1         11     18          30          (Columns)
+---------+------+-----------+-----

          CTEXT  XXXXXXX - common deck description.
          SPACE  4,10
 QUAL$    IF     -DEF,QUAL$
          QUAL   XXXXXXX
 QUAL$    ENDIF
          BASE   B           (B = any legal value)
          CODE   C           (optional)
*         COMMENT  COPYRIGHT CONTROL DATA CORPORATION, year.
 XXX      SPACE  4,10
```

Refer to section 4.4.3 for further information on qualification of common decks.

3.2.2 PROGRAM LEVEL DOCUMENTATION

Program level documentation consists of overview, external and internal documentation as described in section 2.3.

3.2.3 MACRO DEFINITIONS

The macros are in alphabetical order. Common decks which define macros should be included before local macro definitions in alphabetical order.

3.2.4 INSTALLATION SYMBOL DEFINITIONS

Installation symbols are parameters that may be changed by a site when installing a product. These symbols may include buffer lengths, default values, and timing delays. Installation symbols are defined in alphabetical order unless functional order is more meaningful. The installation symbol definition area should be bracketed by internal bracket lines (****).

3.2.5 LOCAL SYMBOL DEFINITIONS

Local symbols are parameters that should not be changed by an installation. These symbols may include code generation symbols (QUAL$, DBI$, etc.) And symbols used for cross reference purposes. Local symbols are defined in alphabetical order, unless functional order is more meaningful.

3.2.6 GLOBAL MEMORY DEFINITIONS

This section of the program is used to define memory that is preset with data. This section may include FETs, tables, and working storage. Global memory definitions are defined in alphabetical order unless functional order is more meaningful.

3.2.7 MAIN LOOP

This section of the program contains the major logic and control flow for the program and internal documentation for that flow (see section 2.4). A TITLE line with an appropriate subtitle precedes the first primary subroutine.

3.2.8 PRIMARY SUBROUTINES

This section of the program contains the subroutines which are of major importance to the program. They should be in alphabetical order unless there is a logically associated set of subroutines which interact together (in which case these subroutines may be grouped together). Each subroutine contains documentation as described in section 2.4. A TITLE line with an appropriate subtitle precedes the first primary subroutine.

3.2.9 SECONDARY SUBROUTINES

This section of the program contains subroutines of minor importance to the program. They should be in alphabetical order unless there is a logically associated set of subroutines which interact together (in which case these subroutines may be grouped together). A TITLE line with an appropriate subtitle precedes the first secondary subroutine. Each subroutine contains documentation as described in section 2.4.

Common decks (except those used for initialization) are after the secondary subroutines. Common decks should be listed in alphabetical order whenever possible.

3.2.10 WORKING STORAGE AND BUFFERS

This section of the program contains working storage and buffer definitions that are not preset with data. (Refer to section 3.4.5) Use of EQU or BSSN is preferred to BSS since additional code is not added to the binary.

3.2.11 INITIALIZATION CODE

Code which may be overlayed after program initialization is included here.

3.2.12 PROGRAM TERMINATION

All programs end with an "END" statement except common decks which end as follows:

```
1         11     18          30          (columns)
+---------+------+-----------+-----

          BASE   *           (if applicable)
          CODE   *           (if applicable)
 QUAL$    IF     -DEF,QUAL$
          QUAL   *
 YYY      ECU    /XXXXXXX/YYY (unqualified entry point)
          .
          .
          .
 QUAL$    ENDIF
 XXX      ENDX
```

If "CODE x" is used at the beginning but "CODE *" is not used at the end of a common deck, it must be explicitly documented in the common deck header.

If the main listing title has been changed by use of an IDENT or TTL line, the main title must be restored with a TTL card just before the END line to provide the correct title on the symbolic reference table.

3.3 INSTRUCTION USE, FORMAT, AND PARAMETERS

3.3.1 REGISTER USE AND SPECIFICATION

3.3.1.1 B0 REGISTER USE

The B0 register should not be specified in instructions which test B registers. The assembler assumes B0 if the requisite number of B registers is not specified.

3.3.1.2 B1 REGISTER USE

The B1 register must always contain the value one (1). The "SYSCOM B1" macro is included in each program to indicate that B1 will contain this value. B1 must be set to 1 immediately upon program entry. B1 is then used by COMPASS in conjunction with the R= psuedo instruction to generate 15 bit instructions rather than 30 bit instructions.

It should be assumed that calls to external entry points which may be loaded from an external source destroy B1. Therefore, B1 should be reset to one after these calls.

3.3.1.3 PACK AND NOMINAL SHIFT X REGISTERS

In the Pack and Nominal Shift instructions, the X register is specified before the B register, as follows:

```
PXi    Xk,Bj

LXi    Xk,Bj
```

3.3.1.4 UNPACK AND NORMALIZE X REGISTERS

In the Unpack and Normalize instructions, the B register is specified in the opcode field immediately following the opcode.

```
UXi,Bj Xk

NXi,Bj Xk
```

3.3.2 MULTIPLE LOGICAL TESTS

When a PP program tests a value in the A-register for equality with several possible values, it may be done with a sequence of logical difference (exclusive "or") operations, as follows:

```
LMC    AA
ZJN    XYZ12        IF TYPE AA
LMC    BB^AA
ZJN    XYZ24        IF TYPE BB
LMC    CC^BB
ZJN    XYZ36        IF TYPE CC
```

The value being tested is specified first in the LMC.

Alternatively, a table look-up may be more efficient.

3.3.3 SHIFT INSTRUCTION PARAMETERS

Shift counts in shift instructions which are used to test bits, are coded in one of the following forms:

```
A-B              (first shift of a word)

A-B-AA+BB+M      (next shift of the word)
```

Where:

A	=	The desired position of a bit in the word.
B	=	The original position of a bit in the word (before any shifts).
AA,BB	=	The A and B parameters from the previous shift of this word.
M	=	Modulus value

Note: The modulus values (60 for CPU and 22B for PP) may have to be added to the shift value if the resulting value is not within the legal limits for the instruction.

Example:

1. To shift bit 47 to bit 59:

```
LXi     59-47
```

2. To shift the result of example 1 so that bit 32 of the original register (before any shifts) is in bit 59 of the result:

```
LXi     59-32-59+47
```

3. To shift the result of example 2 so that bit 58 of the original register (before any shifts) is in bit 59 of the result:

```
LXi     59-58-59+32
```

Example:

1. To shift bit 2 to bit 21B:

```
SHN     21-2
```

2. To shift the result of example 1 so that bit 5 of the original register (before any shift) is in bit 21 of the result:

```
SHN     21-5-21+2+22
```

A modulus of 22B is needed in this case to avoid executing a right shift (ie. The resultant shift would otherwise be negative.)

3.3.4 BOOLEAN MASK USAGE

The mask created for use in boolean instructions depends on whether the field of bits to be extracted is in the left or right hand part of the word. If the field of n bits is in the left hand part of the word, use the following method:

```
MXi    n
BXj    Xi*Xk (Xj contains the extracted field)
```

If the field of n bits is in the right hand part of the word, the following method is used:

```
MXi    -n
BXj    -Xi*Xk (Xj contains the extracted field)
```

If the mask is used in more than one way, the first use determines how it is defined.

3.3.5 RELATIVE ADDRESSING

Relative addressing (such as *+n and *-n, where n is a numeric value) should not be used except:

1. In timing delays (where *-1 is the only acceptable value).
2. For instruction modification (where *-1 or *-2 are the only acceptable values).
3. In PP code to reference bytes within a CPU word. The relative address must be in one of the following forms:

```
tag+n
tag+c*5+n
```

where:

tag = base address
c = CM word within the PP buffer
n = byte within the CM word (0 - 4)

3.3.6 JUMP INSTRUCTION USE

Unconditional jumps in CPU code are coded using the EQ instruction so that the instruction stack is not voided. When it is necessary to void the instruction stack the RJ instruction is used. (The RJ is the only instruction which voids the stack on all central processors.)

PP jump macros MJP, NJP, PJP, UJP, ZJP can be used to assemble short or long jump as needed; however, these macros should be avoided when branching forward since a long jump sequence is always generated if the jump address has not yet been defined on pass 1 of the assembly.

A blank line is inserted after each unconditional jump instruction to indicate a break in the program flow. If the unconditional jump occurs at the end of a subroutine, a SPACE line or TITLE line may be used.

A blank line is also required after an implied unconditional jump. The following are examples of an implied unconditional jump.

Example:

A blank line should be inserted after macro calls that break the flow of execution in a sequence of code.

```
            .
            .
            .
            NZ      X1,tag2     IF comment
            ABORT               TERMINATE
            (blank line)
tag2        SA1     B2
            .
            .
            .
```

Example:

When code occurs before the SUBR, there should be a blank line between the code and the SUBR.

```
tag2        LDN     0           comment
            (blank line)
tag         SUBR                ENTRY/EXIT
            .
tag1        .
            .
            UJN     tagX        RETURN


tagA        BSS     1           comment
```

When storage locations for a subroutine are defined at the end of the routine, there should be 2 blank lines between the code and the first data tag.

3.3.7 SUBROUTINE ENTRY

Each subroutine has one and only one entry point. Exceptions are allowed as follows:

If memory limitations in a PP program make this impractical.

For termination subroutines (such as error processors). Each entry point should be documented within the subroutine.

PP and CPU subroutines which are entered via a return jump contain the following instruction at their entry/exit point:

```
tag     SUBR            ENTRY/EXIT
        .
        .
        .
        UJN     tagX    RETURN
```

or,

```
tag     SUBR            ENTRY/EXIT
        .
        .
        .
        EQ      tagX    RETURN
```

A subroutine may also consist of a block of code that is entered by a jump instruction. In this case, the subroutine entry points should be clearly documented using a BSS pseudo-instruction:

```
TAG     BSS     0       ENTRY
        .
        .
        .
```

Subroutines defined with SUBR should be used for hangs and error processors so the RJ/RJM for CP/PP code leaves a trace of the caller, even though return to the caller is not used. PP's should write the caller's address and other pertinent information in the PP output register or message buffer (space permitting) before issuing a HNGM monitor function.

3.3.8 CPU CODE OPTIMIZATION

An effort should be made to avoid the generation of NO-OPs at the end of a 60-bit word. This may be done by arrangement of code so that each 60-bit word is completely filled with executable code. This is also done for instructions which have an optional "k" parameter by supplying a zero value for "k", thus generating a 30-bit instruction instead of a 15-bit instruction. The way to do this is to append a "+" to the register in the variable field of the instruction, as shown below:

```
        SA4     A1+     (Generates 30-bit instruction)
```

This indicates that the padding was added for optimization purposes and may be removed as necessary when the code is modified.

When initializing an X register tc zerc, a

```
SXi    B0+
```

should be used if a 30 bit instructicn packs better.

If a 15 bit instruction packs better,

```
BXi    Xi-Xi
```

is preferred; but fcr efficiency

```
SXi    B0
MXi    0
```

also may be used interchangeably.

3.3.9 CLEARING PP MEMORY

The following coding sequence is used tc clear 5 consecutive words of PP memory to zeroes:

```
LDN    ZERL
CRD    tag
```

The constant ZERI should not be assumed tc be at address absolute zero in memory.

3.3.10 INSTRUCTICN MCDIFICATION

Instruction modification greatly increases the complexity cf code and is a reliable source of prcgram errcrs. It is a practice to be avcided wherever pcssible. The scle justification fcr instruction modification is cverwhelming space cr time-critical constraint, such as a crowded PP, an in-stack lccp, cr a hardware driver. It is particularly impcrtant that cne rcutine mcdifying the ccntents of another routine be avoided. It is far preferable to emplcy a global variable for communications between routines, even at the expense of some storage.

Where a routine must modify ccde within ancther routine, the modified code must be documented as an EXIT condition from the first routine and an ENTRY conditicn tc the second.

Data locaticns imbedded within a rcutine and referenced by more than one routine should be assigned descriptive, global variable symbols (this is an exception to the standard fcr the naming of data locations within a rcutine). This will scmewhat decrease the chances of error arising from their use.

Instructions which must be modified are to be followed by a comment which shows each alternative form under which the instruction can take. The following examples show the layout used:

Example:

```
     tag      LDC    TRCO          SET READ FUNCTION
    *         LDC    TWTO          (WRITE FUNCTION)
    *         LDC    TFCN          (POSITION FUNCTION)
     tagA     EQU    *-1
```

Example:

```
     tagA     LDC    *             RESTORE (T1)
    *         LDC    (T1)          (CONTENTS OF T1)
              STD    T1
```

The comment in ( ) should describe the conditions under which the instruction is changed.

In CPU code, care must be taken to insure that the instruction being modified is not already in the instruction stack. Since the only way to guarantee this for all mainframes is to perform an RJ instruction, any CPU program that does code modification must have at least one RJ instruction between the modification and the execution of the code. This RJ may be a call to a dummy subroutine, or to a "normal" one; if a call to a normal subroutine is also being used to void the instruction stack, the comment on the RJ should note that fact.

PP short jump instructions which must be modified are tested for range errors. The LCC pseudo-op is used and the jump instruction is actually assembled if the program size is not a critical factor. For example:

```
              LDM    TAGB
              STM    TAGA
              .
     TAGA     MJN    TAG1          IF TIME NOT EXPIRED
    *         UJN    TAG2          (ONE CPU ONLY)
              .
              .
     TAGB     BSS    0
              LOC    TAGA
              UJN    TAG2          comment
              LOC    *O
```

When program size is a constraining factor and the tag to be modified is previously defined, the ISTORE Macro defined in COMPMAC should be used:

```
          ISTORE CADDR,(INSTR)
```

where CADDR is the address of the code to be modified,
INSTR is the instruction (op code and address field) to be stored.

For example:

```
 TAGA     MJN    TAG1        IF TIME NOT EXPIRED
*         UJN    TAG2        (ONE CPU ONLY)
          .
          .
          .
          ISTORE TAGA,(UJN TAG2)
```

Which generates the following sequence of instructions:

```
          LDC    **
          ORG    *-1
          LOC    TAGA
          UJN    TAG2
          LOC    *O
          STM    TAGA
```

When program size is a constraining factor and the tag to be modified has not yet been defined, the jump should be assembled as part of an LDC instruction as follows:

```
          LDC    UJNI+TAG2-TAGA
          STM    TAGA
```

In this case, the ERRNG psuedo instruction must be used to test for range errors as follows:

```
          ERRNG  37+daddr-jaddr  (comment)
    or
          ERRNG  37+jaddr-daddr  (comment)
```

(depending on whether the jump is a backward or forward jump respectively)

Where:

jaddr =address of jump instruction
daddr =destination address of jump

Again, instruction modification should be avoided in PP and CPU code whenever possible.

3.3.11 COMMON DECK REGISTER USAGE

CPU code within common decks avoids using registers A0, A5, X0 and X5 unless absolutely necessary. If these registers must be used, they should be restored before exiting to the calling routine.

3.3.12 PP ADK, IDK, LMK, LPK, SBK Macros Usage

Use of ADK, LDK, LMK, LPK, and SBK macros defined in COMPMAC are encouraged, since the actual instruction assembled will be adjusted to a 0, 1, or 2 byte instruction as needed, depending on the tag values in the operand field. If the operand value reduces to zero, no instruction will be generated (except for LDK). Operands to these macros should not be numerics only (usefulness for tags is recommended). Because of the variability of the code generated by these instructions, this code should not be changed by in-line code modifications.

3.4 DATA USE, FORMAT, AND PARAMETERS

3.4.1 LITERALS

Literals may be used for read-only constants only. Error message text should not be defined as literals, but rather should be defined in data statements (preferably in tables).

3.4.2 DATA FORMATS

Data is specified in its natural form (readable and understandable by humans) using post-radix symbols as required (see section 3.6.1). If conversion considerations make this impossible, the comment field will contain the natural form of the data. Octal values are not used for character data unless the data cannot be specified in any other way. When the VFD is used, it cannot generate more than one CM word of data.

If a data item does not require an initial value preset at assembly time, BSS should be used to reserve space rather than CON.

Only one piece of data is specified on a line of code unless a block of data is being specified for use as a single data item to be referenced by a single name.

3.4.3 TABLE GENERATION

Tables which are generated with entry ordinals relative to the base address of the table, should use the LOC pseudo-op as shown in the following example:

```
TFCN    BSS     0           table entry
        LOC     0
        CON     RNM         first entry
        CON     ACF         second entry
        .
        .
        .
        CON     VSN         last entry
        LOC     *O
TFCNL   EQU     *-TFCN      table length (optional)
```

Where tables are described, they are defined so they can be processed by the "Documentation Table Generator". A description of this format is found in the external documentation for the program DOCMENT.

3.4.4 DIRECT CELL USE

Direct cells are defined using one of the following methods:

1. A single cell:

```
 xx     EQU     n           description
```

2. Multiple cells:

```
 xx     EQU     n - m       description
```

3. Contiguous cells:

```
        LOC     n
xx      BSS     1           description
yy      BSS     5           description
        .
        .
        .
zz      BSS     1           description
        LOC     *0
```

4. Contiguous direct cells or other sequential tag definitions without reserving space:

```
BEGIN     BSSN     n
xx        BSSN     1        description
yy        BSSN     5        description
            .
            .
            .
zz        BSSN     2        description
END       BSSN
```

The BSSN macro is defined in COMCMAC and COMPMAC.

Where:

xx, yy, zz = the tag for the cell
n = location of the cell (or first cell)
m = location of the last cell

Multiple definitions of direct cells should be avoided.

The first few direct cells in the PP should not be used for data which is critical to debugging. The deadstart dump process destroys the contents of these locations:

T0 - T3 and 7774 - 7777

3.4.5 BUFFER DEFINITIONS

Large buffers and working storage areas should be defined using EQU statements (rather than BSS and BSSZ) to avoid unneccessary loading of the buffer areas that do not require initialization. This applies to CPU and PP code.

```
          USE     BUFFERS
 IBUF     EQU     *
 OBUF     EQU     IBUF+IBUFL
 RFL=     EQU     OBUF+OBUFL
```

Small buffers and working storage areas may be allocated via BSSZ, if the program requires that the area be zero on program initiation.

The BSSN macro defined in COMCMAC and COMPMAC may also be used to define buffers.

3.5 DATA/CODE NAMING TERMINOLOGY

3.5.1 USE OF CONDITION TERMINOLOGY

The following terms are used to describe the condition of bits used as flags or switches. The selected terms should be used consistently within a program.

1	0
on	off
true	false
set	clear (reset)
nonzero	zero
up	down

3.5.2 TAGS WITHIN SUBROUTINES

Each subroutine (main loop, primary subroutine or secondary subroutine) has a meaningful three character name which is derived from the title of the subroutine (see section 2.4).

Tags used for branch instructions are of the form:

XXXn Example: GFN1

Tags on code which is added later to the subroutine are of the form:

XXXN.n Example: GFN1.1

Tags that are inserted between the SUBR and the tag XXX1 by corrective code are of the form:

XXX0.n Example: GFN0.1

Tags on storage locations (constants, temporary storage and instruction modification) within a subroutine are of the form:

XXXa

Where:

XXX	=	Subroutine name
XXXN	=	Tag preceding an added one
n	=	Number from 1 to 99 (in consecutive order beginning at the entry point and ending at the exit point)
a	=	Letter from A to Z and AA to ZZ (in alphabetical order and excluding X)

Tags of the form XXXn, XXXn.n, and XXXa should not be referenced outside of subroutine XXX.

If the above rules cannot be followed due to tag unavailability, the entire subroutine will have its tags resequenced.

Tags on storage locations within a subroutine which must be referenced outside of the subroutine should be given appropriate global tags.

3.5.3 TAGS ON DATA

3.5.3.1 DIRECT CELLS

All PP direct cells have two-character names.

The following table defines the preassigned direct cell usage:

Contents	Name	Location
Control Point RA/100	RA	55
Control Point FL/100	FL	56
1	ON	70
100	HN	71
1000	TH	72
3	TR	73
CPA address	CP	74
PP input register address	IA	75
PP output register address	OA	76
PP message buffer address	MA	77

3.5.3.2 CODE CONTROL NAMES

Names used for assembly options, micros and to control code generation are five or more characters long.

3.5.3.3 TABLE NAMES

Tags used on tables have the form:

```
Txxx
```

Tags used for table lengths have the form:

```
TxxxL
```

Tags used for table entry lengths have the form:

```
TxxxE
```

Where:

xxx = 3 character table name

3.5.3.4 GLOBAL MEMORY LOCATIONS

Names used for global memory locations (locations referenced by more than one subroutine) are four characters long. Multiple definitions for global memory locations should be avoided.

3.5.4 CONSTANTS USED AS INSTRUCTIONS

Four letter tag names should end in "I" if the tag name is defining an instruction.

For example:

```
LJMI    EQU    0100B        *LJM* INSTRUCTION
SHNI    EQU    1000B        *SHN* INSTRUCTION
```

PP instructions are defined as constants in common deck COMSPIM.

3.5.5 IF/ELSE/ENDIF SYMBOLS

Symbols used on IF, ELSE, ENDIF and SKIP psuedo instructions may be system symbols or local symbols of the form:

```
.a
```

where:
a = letter from A to Z

Unlabeled, unnumbered IF/ELSE/SKIP/ENDIF sequences should not be used as they may unexpectedly affect other sequences. For very short sequences a line count may be used, for longer sequences labels are preferred.

3.5.6 NON-LOCAL MACRO SYMBOLS

To avoid conflicts with user code, non-local symbols defined within macros are of the form:

```
.n
```

where:
n = number from 1 to 99

3.5.7 LOW CORE LOCATION SYMBOLS

Symbols that are used to define locations in low core (CMR) are of the form:

xxxL

3.5.8 CONTROL POINT AREA LOCATION SYMBOLS

Symbols that are used for defining locations in the control point area are of the form:

xxxW

3.5.9 MONITOR FUNCTION SYMBOLS

Symbols used for monitor function requests are of the form:

xxxM

3.5.10 NEGATIVE FIELD LENGTH SYMBOLS

Symbols used to reference data in negative field length are of the form:

xxxN

3.6 PSEUDO INSTRUCTION USE, FORMAT, AND PARAMETERS

3.6.1 BASE AND POST RADIX USE

The BASE DECIMAL pseudo-op is used in all CPU code. The BASE MIXED pseudo-op is used in all PP code. Post Radix is allowed for data formats other than octal and decimal; in specifying timing loops where decimal values are more meaningful to humans; and where external specifications such as ANSI or corporate standards dictate the use of a particular format.

3.6.2 EXTERNAL REFERENCES

The EXT pseudo-op is not used. All references to external names use the form =Xname. In an absolute assembly, references to locations in other overlays use the form =Xname.

3.6.3 SPACE CARD FORMAT

The format of the SPACE pseudo instruction is:

```
     tag       SPACE  4,n
```

where:

```
       tag      =table, macro or subroutine name
       n        =statement count
```

The statement count is a multiple of 5 that is greater or equal to 10. It should be large enough to avoid breaking documentation across page boundaries.

3.6.4 CONDITIONAL CODE

Numeric skip counts are discouraged with IF, IFC, ELSE, etc., because this makes code difficult to read (especially when the skipped lines are not listed). ENDIF should be used instead. An exception is allowed for very short sequences and for systems texts where space is critical.

Conditional sequences should be bracketed with labels (refer to section 3.5.5) which allows them to be easily spotted and matched in listings.

When either end of a sequence of conditional code occurs at a break in the listing (SPACE, TITLE, or blank lines), the spacing lines should be placed so that spacing will be correct whether the test is true or false. Usually this means moving the spacing outside the conditional code.

Example:

```
            .
            .
            .
            EQ      TAG1            CONTINUE
            (blank line)            (outside conditional code)
    .A      IFC     EQ,*"SYSTEM"*SCOPE*
    TAG3    CONTROL TAGA,R          READ COMMAND
            EQ      TAGX            RETURN
            (blank line)
    TAGA    BSS     8               COMMAND BUFFER
    .A      ELSE
    TAG3    CONTROL  CCDR           READ COMMAND
            EQ      TAGX            RETURN
    .A      ENDIF
    abc     SPACE   4,10            (outside conditional code)
```

3.6.5 MACROS

Macro definitions should include a description of how the macro is called and a description of all formal parameters. (Refer to section 2.5.)

The PURGMAC psuedo instruction should be used to disable any previous macro definitions of the same name.

Non-local symbol definitions should be of the form .n (see section 3.5.6).

To avoid terminating multiple macro definitions, the ENDM instruction should be labeled with the macro name.

The MACREF macro (defined in SYSTEXT) may be included within the macro definition body to provide symbolic reference table listing of the calls of the macro.

The SYSTEM XXX,= macro (defined in CPCOM) should be used for cross referencing of CP programs calling PP programs without a standard interface (Examples: CPUMTR calling 1MA, MAGNET calling 1MT via SPC call).

The EXECUTE XXX,= macro (defined in COMPMAC) should be used for cross referencing of PP programs calling PP programs or overlays without a standard interface.

When space is critical, as in systems texts, the following list of column numbers represent the beginning of each field in a COMPASS coding line to be used in a macro definition.

Column 2 = location field
column 3 = operation field
column 6 = address field
column 73 = reserved

If a field is full or overflows into an adjacent field, then one space should separate the fields. If comments are required, they should appear as stand-alone comments rather than embedded comments.

3.6.6 DIS PSEUDO INSTRUCTION

The DIS pseudo instruction should not be used to generate data since the syntax of the instruction determines the format of the data. The DATA psuedo instruction should be used instead.

3.7 TESTS FOR OVERFLOW

CPU and PP programs should contain assembly checks for certain types of overflow conditions. The following points should be considered when making the checks.

PP programs and overlays are generated by COMPASS in multiples of 5 PP bytes (1 CM word). Therefore, when reading an overlay from central memory to PP memory, more PP bytes may be destroyed than the actual number of bytes of PP code.

Overlays loaded from mass storage to PP memory come in multiplies of 500 PP bytes. At least 5 bytes of the last PRU are required to represent end of record which can increase the size of the overlay by one PRU (500 bytes).

Care should be taken to insure that the literals block has been defined before checking for the overflow conditions. Literals can be flushed by specifying:

```
          USE    (name)
          USE    *
```

The OVERFLOW macro (defined in COMPMAC) may be used to perform these checks.

3.7.1 CM LOADS

All PP programs include a test for the amount of core remaining after a CM load as shown in the following example:

```
          USE    OVERFLOW
xxx       BSS    0
          ERRNG  7772-xxx    PP MEMORY OVERFLOW
```

Where:

xxx =tag for the last location defined

3.7.2 TABLE OVERFLOW

If a PP program uses more storage than it declares, its length is checked as shown in the following example:

```
          USE    OVERFLOW
xxx       BSS    0
xxxE      EQU    xxx+xxxL
          ERRNG  7777-xxxE   PROGRAM OVERFLOW
```

Where:

xxx = tag for the last location defined
xxxL = length of undeclared space
xxxE = end of space used

3.7.3 MASS STORAGE LOADS

A test will be included in each PP program which may reside on mass storage. This test will protect against a load which exceeds the end of memory in the PP causing wrap around. The "OVERFLOW" macro is available in COMPMAC for this operation.

3.7.4 OVERLAY LOADS

Programs calling overlays should test for memory overflow with the following test:

```
            ERRNG  (1wa+1)-(load addr)-len   comment
```

where:

1wa+1 = first byte not to be destroyed by the zero level overlay
load addr = address where the overlay is loaded
len = length of overlay

The length of an overlay is defined to be the number of bytes destroyed by the overlay during loading and execution. The overlay should also contain a test to insure that it does not exceed its defined length. The overlay length can be adjusted to a higher or smaller value as long as none of the tests fail. The "OVERFLOW" macro is available in COMPMAC for this operation.

3.8 RELOCATABLE CPU CODE

The first word of a relocatable CPU program should be of the format:

```
        42/OLDECK, 18/ADDR
```

where:

DECK = deck name
ADDR = entry address of program

This word is used to locate the first word address and entry point of a routine in a CM dump.

The contents of A0 must never be used in a library level routine unless it is saved and restored. A0 is used by FTN as a base register for formal parameters in subroutine linkages.

3.9 ROUTINE/SUBROUTINE COMPLEXITY

In this context, the term "complexity" is used in its formal sense; that is, a sense of the structural incoherence (entropy) of a routine. The more complex a routine is, the more liable it is to be a source of errors, difficult to implement, and worse to modify or correct. There are no hard and fast rules for gauging the complexity of a routine, but it can be said in general that the longer it is, the more decisions it makes (branches), and the more functions it performs, the more complex and unreliable it tends to be.

In order to reduce complexity, the following guidelines are to be followed whenever possible (i.e., not impossible).

1. One routine - one function. Each routine should have one clearly defined function.

2. 10-Tag rule. If there are more than 10 branching locations within a routine, it is most likely attempting to perform too many functions (see 1 above). It should be considered a candidate to be broken up into functional units.

3. Code Modification. Minimize within routines, avoid between routines. If used between routines, document thoroughly.

4. Hidden Variables. Data placed in a register with the hope of being used at some later time often may not survive to its destination. Consider global variables for inter-routine communication, especially when there are one or more routines intervening. In any case, all exit conditions from a routine must be documented.

5. Code for the Future. Always consider the implications of debugging, modification, and maintenance; structure code to make these tasks easier.

4.0 MISCELLANEOUS

4.1 PROGRAM NAMING

4.1.1 LENGTH OF PROGRAM NAME

Peripheral processor program names are 3 characters long.

Central Processor program names are 4 to 7 characters long.

4.1.2 RESERVED NAMES

The following PP program names are reserved or presently defined for use (x means any character legal in a PP program name and n means any number between 0 - 9):

Uxx	Reserved for installations
nUx	Reserved for installations
9AA-9T9	Reserved for system use
9VA-9Z9	Reserved for system use
90A-929	Reserved for diagnostics
93A-939	Reserved for system use

The following names are currently used by NOS:

6xx	Callable mass storage drivers
7xx	Mass storage error processing overlays
9AA-9D9	DSD overlays
9EA-9F9	DIS overlays
9GA-9G9	C26 overlays
0xx	location free overlays
0Cx	Controlware identification processors
0Px	Pack number identification processors
0Tx	Automatic track flaw processors
UxxL	PPCOM symbol - reserved for installations
UxxM	PPCOM symbol - reserved for installations
UxxN	PPCOM symbol - reserved for installations
UxxP	PPCOM symbol - reserved for installations
UxxW	PPCOM symbol - reserved for installations

In general, tags beginning with U should not be used as tags in NOSTEXT, COMSXXX decks, etc. or as macro names in COMCMAC, COMPMAC, etc. These tags should be reserved for installation use.

4.1.3 COMMON DECK NAMES

Common deck names are seven characters in length and in the following form:

COMxaaa

where:

aaa = The name of the routine or a symbolic name if no routine name.

X = One of the following common deck indicators:

C = CPU code
P = PP code
S = Subsystem text symbols, constants etc.
D = Display driver code
T = Tables
M = Mass storage error equivalents
B = Data manager
K = Transaction subsystem
I = Initialization

The following indicators are reserved for SYMPL common decks: A, E, U, Z.

4.2 CODE TRANSMITTAL RULES

Code which is to be integrated into a system build for eventual release to the field is identified and formatted as described in this section.

4.2.1 GENERAL RULES

Each external PSR being answered has a corresponding corrective code identifier (to be described later). Corrective code answering other PSRs is not included in the modification under this identifier. Exceptions are allowed where required by interrelated modifications for several PSRs.

Corrections are placed in ascending numerical order; i. e., the corrections are sorted in the same order that the lines being corrected appear on the program library. If a single modification changes several decks, then the corrections are also sorted in the order that the decks appear on the program library.

Corrections modifying lines with previously modified sequence numbers include the line number of the nearest preceding original line in parenthesis in the comments field of the modify directive.

4.2.2 MODSET FORMAT

4.2.2.1 MODSET IDENTIFIER

Separate modset name lists are maintained for NOS 1 (R5.5) and NOS 2 (R6.0). Processes for naming modsets are identical, except for multiple deck modsets (NS1xxx and NS2xxx), documentation modsets (DOK1xxx and DOK2xxx), and release feature modsets (FN1xxx and FN2xxx). A modset named 1CD4 in NOS 1, for example, has no relationship to modset 1CD4 in NOS2.

You should not sign up for a modset name until after code has been generated and tested, and depending on the size of the modset, even after code review has taken place. This allows other analysts submitting code to get the next available modset name. You should not request modset names for modsets not to be included in the current series of builds; you should not request feature modset names for a future release until feature code for the current release is complete. If you sign up for a modset name and don't use it, notify Code Control so that the modset name can be used by someone else.

The modset name consists of three to five alphanumeric characters which are extracted from the deck name followed by a 1 to 3 digit sequence number. The modset name cannot exceed 7 characters.

1. For common decks that begin with "COM" use the last four characters of the name (example - CMAC4 is a modset in deck COMCMAC).

2. For PP programs use the three character program name (example - CPM2 is a modset in deck CPM).

3. All other decks use the first five characters of the deck name; if the deck name is less than six characters use the entire deck name (example - LIBED2 is a modset in deck LIBEDIT).

4. Modifications which involve multiple decks are given the modset name:

   NS1xxx if NOS 1,
   NS2xxx if NOS 2.

   Example - NS1001 is a multiple deck modset in NOS 1.
   NS2001 is a multiple deck modset in NOS 2.

5. Modifications which only correct documentation within a deck are gathered together for each corrective code release and given the modset name:

   DOK1xxx if NOS 1,
   DOK2xxx if NOS 2.

   Example - DOK1006 is a NOS 1 modset.
   DOK2006 is a NOS 2 modset.

This does not include lines of code which have documentation changes in them.

6. If a modset is adding a new feature, the feature modset name for each deck modified is obtained from NOS Code Control.

7. Upon release of the system, a "compgosite" modset is generated from all feature code which is to be released.

4.2.2.2 MODSET CORRECTION LETTER

When a modset is correcting a previous modset, one alphabetic character (starting with A) will be appended to the sequence number. Whenever a modset correction letter of "B" or above is required, the comments header of the modset must indicate which previous modset is being corrected.

4.2.2.3 OVERFLOW

Whenever a modset identifier using the above conventions exceeds seven characters, truncate the last character(s) of the deck name to reduce the identifier to seven characters.

4.2.2.4 MODSET EXAMPLE

The following format is used for corrective code modsets:

```
1       11        18              30        (column numbers)
+-------+---------+--------------+

ident
*IDENT  ident     initials.       yy/mm/dd.
*/      ****      system.
*/      ****      PSR number.
*/      ****      REQUIRES - modset.
*/      *****     PROBLEM - problem description.
*/      (continuation of problem description).
*/
*/      SOLUTION - solution description.
*/
*/      *****     RESUBMITTAL - yy/mm/dd.
*/      Reason for resubmittal.
*DECK   deckname
*I,sequence number
*D,sequence number
*D,modname.sequence number        (nearest original seq. no.)
*EDIT   deckname                  (if common deck)
*/      END OF MODSET.
```

Where:

ident	= Modset name.
initials	= Initials of the analyst(s) who wrote and/or tested the code. Example - ABC. (one analyst) ABC/ADD. (two analysts)
yy/mm/dd	= Date of last modset change.
system	= Name of system in which the modset will be released. (ie - NOS 1 OS. Or NOS 2 OS.)
PSR number	= This line must always be present. If more than one PSR is answered, list the PSR numbers on separate lines. If no PSR is involved, use the phrase: NO PSR.
modset	= The REQUIRES lines must always be present. If more than one modset is required (direct or indirect dependency), list the modsets on separate lines. If no dependency is involved, use the phrase: NONE.

Since mods to mods in the same release are not allowed for PSR code (physically dependent code must be in the same modset), the REQUIRES should specify modsets with logical or space dependencies. Modsets previously released in a standard system usually should not be included. For initial transmittal of feature code, REQUIRES - NONE should be used. For feature repair code, the REQUIRES should specify those modsets going into the same build that are dependent on each other.

The problem description should describe the problem being fixed by the modset and the impact the problem has on the user. The solution description should describe how the problem was fixed. External interface changes caused by the installation of the modset should also be documented. The problem/solution description may be combined into one paragraph to avoid redundancy.

The *READPL directive is not used.

4.3 INTERFACE CONSIDERATIONS

4.3.1 SYSTEM_SUPPLIED_INTERFACES

All interactions between programs (CPU and PP) and the system use system-supplied macros, linkage labels or common decks. In PP programs the system defined direct cells are only used as defined by the system. (see section 3.5.3.1)

4.3.2 PARAMETER_VALIDATION

Each parameter passed between programs will be validated or processed in a way that protects the program from uncontrolled actions caused by unexpected values.

4.3.3 MEMORY_ACCESS

PP programs which access the field length of a job will insure that no combination of parameters, errors, etc. will cause access to an address outside of that field length. Addresses should be validated prior to using them for a CM read or write to avoid referencing areas of memory outside of the control points field length.

A PP program accessing the field length of a control point should insure the relative address does not exceed 377777B.

4.3.4 SECURITY

Programs that perform privileged functions must insure that the requester of the function has been given permission by the system to use the function. This also applies to the use of special device drivers, which could be called accidentally or maliciously by unauthorized users. Where common decks are available to check security or privileges, they should be used rather than locally written code.

4.3.5 RESERVATIONS_AND_INTERLOCKS

Reservations and interlocks are only used as defined by the system and are released as soon as possible. Non-essential code is not executed while a reservation or interlock is in effect.

In cases where a reservation reject could occur, the program will:

1. Control the rate of reservation re-issue.
2. Detect and respond to error conditions.
3. Protect against storage move lockup.

Programs which use reservations and interlocks will insure that the conditions are released no matter what program path is taken.

When multiple interlocks are required, all programs in the operating system must request the interlocks in the same order. When a reject occurs when attempting to obtain such interlocks, all reservations held must be released and the entire sequence of interlocking must begin again.

Care must be taken to not issue dayfile messages, load overlays, or pause with non-dedicated channel(s) reserved.

A PP program must not have a disk channel reserved when it attempts to do an STBM or AFAM monitor function. If necessary, a ENDMS should be performed to ensure this.

4.3.6 DOCUMENTING HARDWARE DEFICIENCIES

Instructions which are included to compensate for hardware deficiencies are documented with a brief description or identification of the deficiency.

4.3.7 NEW FUNCTION/ICW CORE IDENTIFIERS

New tags in PPCOM, use of previously reserved fields in CMR and CPA, new PP function numbers, new monitor function numbers, etc. must be signed up for via the DSO (Design Support Office).

4.3.8 DECK INTERDEPENDENCIES

Beware of deck interdependencies which may require additional code/modsets, such as PPCOM and COMSXXX changes affecting DSDI, DIS changes that should also be made in XIS, and COMCXXX/COMDXXX/COMPXXX/ COMSXXX/COMTXXX, etc. changes causing CALLCPU/CALLDIS/CALLPPU/ CALLSYS/CALLTAB, etc. to not assemble without lots of errors.

4.4 MODULARITY

4.4.1 PP OVERLAYS

PP Programs use overlays whenever possible to improve the long range performance of the system. Overlays are used for any seldom executed code such as error handling and seldom used features.

4.4.2 HELPER PP-S

Helper PPs are not used unless no other method exists. The availability of PPs when needed should be considered, since the use of helper PP's may lead to deadlocks if no PP's are available.

4.4.3 COMMON DECKS

A common deck containing executable code consists of one or more subroutines (as defined in sections 3.2.6 and 3.2.7) and any associated data storage areas. The purpose of common decks is to increase efficiency in writing code, insure uniformity of code and decrease debugging time. Common decks contain optimized code and external interfaces that are generalized to facilitate their use in future programs. These decks should be used in preference to local code whenever possible.

Common Decks which contain only macros are not qualified. "S" type common decks are not qualified by the QUAL pseudo-op within the common deck. If an "S" type common deck is qualified externally, the qualifier is the three character name of the routine. For example:

CCMSaaa (where aaa is the qualifier)

4.5 DAYFILE MESSAGES

Dayfile messages issued to the user or system Dayfile begin with a blank character and end with a period. Dayfile messages should not exceed 50 characters. Abbreviations should be avoided when possible in dayfile messages.

Informative messages should be issued to the user dayfile only (option 3 on the MESSAGE macro or the CFCN option on the call to DFM). Messages that indicate that the job will abort are the only informative user messages that should also be issued to the System Dayfile. Special system programs (NCDVAL, ISF, Subsystems, PF or Queue Utilities, etc.) are exceptions and may issue informative messages to the System Dayfile when necessary.

4.6 UNHANGABLE CHANNEL CODE

To avoid channel hangs, bit 2**5 is set on some PP channel instructions. This should only be used when undesirable side affects will not result and where it is possible to take corrective action. (For example: disconnecting an inactive channel will not result in undesirable effects.) Bit 2**5 should not be used when unpredictable results may occur. Bit 2**5 is not used with channel 15.

Example:

```
            IJM     tag1,CH       IF CHANNEL DISCONNECTED
            DCN     CH+40         DISCONNECT CHANNEL
            RJM     ERP                PROCESS ERROR
tag1         .
             .
             .
```

4.7 SPECIAL ENTRY POINTS

Special entry points defined by NOS include:

```
ARG=    Inhibit argument processing
CLB=    Command line buffer
DMP=    Allow special system processing
LDR=    Loader processing
MFL=    Minimum field length
RFL=    Running field length
SDM=    Suppress dayfile message
SSJ=    Special system job
SSM=    Secure system memory
VAL=    Validation program
```

To insure proper loading and execution of special entry point programs, special entry points must be declared after normal entry points.

EXAMPLE:

```
IDENT     FWA
ABS
ENTRY     ABC
ENTRY     XYZ
ENTRY     RFI=
ENTRY     SSJ=
SYSCOM    B1
   .
   .
   .
```

4.8 SCRATCH FILE NAMES.

Programs requiring temporary scratch files will use the names ZZZZZG0 - ZZZZZG9 as names for scratch files. Programs which require more than these 10 scratch file names must resolve the required file names with Systems Design. Such scratch files must always be returned at program termination.

APPENDIX A - ABBREVIATIONS

Standard industry abbreviations and programming language names may be used even though they are not included in the following appendix.

A.1 GENERAL ABBREVIATIONS

BML	binary maintenance log
BOI	beginning of information
CLT	common library table
CM	central memory
CME	central memory extension
CMM	common memory manager
CMR	central memory resident
CMU	compare/move unit
CP	control point
CPA	control point area address
CPU	central processing unit
CR	carriage return
CSU	cartridge storage unit
CW	control word
DAT	device access table
DIT	device interlock table
ECS	extended core storage
ESM	extended semi conductor memory
EJT	executing job table
EJTO	executing job table ordinal
EM	extended memory
EOF	end of file
EOI	end of information
EOL	end of line
EOR	end of record
EOS	end of stream
EPD	entry point directory
EST	equipment status table.
FOT	family ordinal table
ETX	end of text
FDX	full duplex
FET	file enviroment table
FL	field length
FLE	field length for extended memory
FLPP	first level PPU
FNT	file name table
FST	file status table
FWA	first word address
HDX	half duplex
ID	identifier or identification
I/O	input/output
JCB	job control block
JSN	job sequence name
LCME	large core memory extended

LFN	logical file name
LWA	last word address
MID	machine identification
MMF	multi-mainframe
MRT	machine recovery table
MS	mass storage
MSA	mass storage adapter
MST	mass storage table
MST	mass storage transport (do not use abbreviation where confusion with Mass Storage Table may result)
MT	magnetic tape
MUX	multiplexer
NFL	negative field length
PF	permanent file
PFC	permanent file catalog
PFN	permanent file name
PLD	peripheral library directory
PP	peripheral processor
PPU	first-level peripheral processor; only on CYBER 176 (also known as FLPP)
PRU	physical record unit
PST	program status table
QFT	queued file table
RA	reference address
RAE	reference address for extended memory
RCL	resident central library
RPL	resident peripheral library
RMS	rotating mass storage
SCP	system control point
SCR	status and control register
SECDED	single error correction, double error detection
SUBCP	subcontrol point
TRT	track reservation table
TTY	teletype
UCP	user control point
UEM	unified extended memory for 8X5
UDT	unit descriptor table
UJN	user job name
VSN	volume serial number

A.2 NETWORK HOST PRODUCTS ABBREVIATIONS

ABH	application block header
ABL	application block limit
ABN	application block number
ABT	application block type
ACK	block acknowledged
ACN	application connection number
ACT	application character type
ADR	address information

ALN	application list number
CLA	communications line adapter
IBU	input block undeliverable
IVT	interactive virtual terminal
LCF	local configuration file
LOP	local operator
NAK	block not acknowledged
NCF	network configuration file
NDL	network definition language
NFE	nc format effectors
NOP	network operator
NPU	network processing unit
PFC	primary function code
SFC	secondary function code
SM	supervisory message
SMP	supervisory message processor
TA	text area
TLC	text length characters
TLMAX	maximum length of data message block text
TNAME	terminal name

A.3 ACRONYMS

AIP	Application Interface Program
BIO	Batchio
CDCS	CYBER Database Control System
CRM	Cyber Record Manager
FSE	Full Screen Editor
IAF	Interactive Facility
LCN	Loosely Coupled Network
MAG	Magnet
MAP	Matrix Array Processor
MCS	Message Control System
MSF	Mass Storage Facility
MSS	Mass Storage Subsystem
NAM	Network Access Method
NOS	Network Operating System
NVF	Network Validation Facility
RBF	Remote Batch Facility
RDF	Remote Diagnostic Facility
RHF	Remote Host Facility
SMF	Screen Management Facility
SSF	Scope Station Facility
STM	Stimulator
TAF	Transaction Facility
TIP	Terminal Interface Program
TVF	Terminal Verification Facility

APPENDIX B - ERROR MESSAGE GUIDELINES

The following general principles are to be observed in designing future error messages for NOS. Existing messages should be improved as opportunities arise. These guidelines will be formalized in a later Usability Design Direction Document.

1. The purpose of an error message is to inform the user how to correct a problem.

   Discussion: It helps to view error messages as prompts: not "this is what you did wrong" but "this is how to do it right." Messages should be phrased positively. The words "ILLEGAL", "INVALID", and "SYNTAX" are specifically not permitted in NOS messages. Of course, there are other ways to phrase unhelpful, negative messages; but these three words are singled out for extinction for being so frequently seen in the company of usability offenders.

2. A single message should diagnose a single error.

   Discussion: For example, if the meaning of message is "more than seven characters or leading non-alphabetic character or null identifier" it should be three messages. Usually, the code must make three separate tests, so it is easy to be precise. An exception is when a common deck returns an error status which could have resulted from several different conditions.

3. An error message is friendly if it is business-like and informative.

   Discussion: Cute, funny, or flippant messages are to be avoided, as they seldom diagnose accurately and always wear quickly. Messages should be directed at the process and not the person.

4. Messages must be written plainly, using terms already known to the user.

   Discussion: Messages should use terms which are either self-defining or natural to the process. All words should be part of the external user interface, like "file name" instead of "LFN" (unless LFN is an external parameter).

5. Messages must be written in English.

   Discussion: Messages should follow normal rules for English grammar and punctuation, although "pidgin English" -- the omission of selected subjects, verbs or objects in the interest of brevity where the meaning is clear - is acceptable. Messages should not be written in octal, or in other forms of scientific notation. Note that the asterisk is not an English punctuator.

6. Messages should be self-contained.

   Discussion: If you need to tell a story, tell the whole story. Avoid references, as they are difficult to keep up-to-date and are often no more helpful then a good one-line message would be.

7. Error messages should point directly to the source of the trouble.

   Discussion: For example, "FILE NOT FOUND" is better put as "FILE XYZ NOT FOUND", "EXPECTING COMMA OR PERIOD AFTER 'ABC'" is much clearer than "SYNTAX ERROR". In general, the technique of echoing back part of the user input as part of the message is better than the use of internal names or parameter keywords which the user may not recognize.

8. Interactive error messages should appear as soon as possible after an error is committed.

   Discussion: Each interactive input should be completely and fully validated as soon as it is received. In no event should a user be led down the garden path to enter a long series of input only to be advised that it is all wrong because the first part was wrong.

9. No messages at all should appear for trivial, correctable errors - nor should they be errors.

   Discussion: Errors such as missing or redundant terminators should not be errors at all. If a reasonable assumption can be made as to the intent of an input, it should be acted upon as though it were "valid". No error diagnostic should be produced for these cases. If it is not perfectly clear what assumption was made, the assumption was probably not reasonable to begin with.

10. An error message must clearly signal that an error has occurred.

    Discussion: An error message must not be phrased in such a way as to be confused with a merely informative message. Also, a message should indicate the gravity and extent of the error, as when an error in a list inhibits processing of the remainder of the list.

APPENDIX C DOCUMENTATION/USABILITY GUIDELINES

1. COMMAND should be used instead of CONTROL STATEMENT or CONTROL CARD.

2. INCORRECT should be used rather than ILLEGAL or INVALID (refer to Appendix B item 1).

3. USER should be used instead of ACCOUNT.

4. USER NAME should be used instead of USER NUMBER or ACCOUNT NUMBER.

5. EXTENDED MEMORY should be used rather than ECS.

6. MONITOR REQUEST should be used rather than RA+1 CALL.

7. EST ORDINAL or DEVICE should be used instead of EQUIPMENT NUMBER or EQUIPMENT.

8. English variables such as file name rather than filenam should be used to remove the shorthand notation of using variable names that are the same length as the maximum entry.

9. Documentation, messages, etc. should avoid the use of sexist language (he, she, him, her, etc.).