

STRAP INDEX

STRAP 1.2	Back up procedures (for 6500)	06/07/84
STRAP 2.3	Installation of CDC Products and PSR Code	06/17/82
STRAP 3.5	Installation of Michigan State University Modifications	06/07/84
STRAP 4.2	Writing Software Modification Documents	06/04/84
STRAP 5.2	Procedures for Efficient Maintenance of the Systems Group tape library	06/05/84
STRAP 6.2	Software Modification Proposal Document	06/08/84
STRAP 7.4	Monthly Reports - Content and Style	06/07/84
STRAP 8	---	
STRAP 9.4	Coding Practices and Conventions	06/08/84
STRAP 10.2	Software STIR Procedures	01/22/78
STRAP 11.2	MINI MOD Memos	06/04/84
STRAP 12.4	Disk Labeling and Flawing	06/07/84
STRAP 13.2	What to do when there is a Hardware Failure	06/07/84
STRAP 14.3	Protection of Disk Packs	06/06/84
STRAP 15.1	Guidelines for Proper Machine Room Conduct	06/07/84

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 1.2

Code Back Up Procedure

June 7, 1984

Definitions:

Program library tape (PL). A tape, that when processed by the UPDATE program, generates the source for a set of programs. Also referred to as old program library or OLDPL. There are generally multiple files on each tape. Each is an UPDATE file for a collection of related programs.

Deadstart tape (DS). A tape that can be loaded into the machine and used to execute programs.

Latest System Description (LSD). This is a notice used to transfer a DS tape from Systems Programming to Operations. All LSD's are numbered, nn.mm where nn is the major LSD number and mm is the minor LSD number. See STRAP 3 for more details.

Correction Deck Library (CDL). This tape contains a file for each PL used at MSU. Each file is UPDATE type data with the master control character changed from "*" to "\$". Each correction ident on a PL has an identically named deck on the corresponding CDL. Thus, if one desires a copy of the cards for ident JSF1SJ3, the procedure is to get the appropriate CDL (use GETPL with the CDL parameter, i.e. GETPL,SCOPE,CDL.), do an UPDATE (Q,*=\$\$\$\$,R=C) with an input card of \$COMPILE JSF1SJ3.

NOTE: Idents that are purged from the PL are also purged from the CDL.

I. On-site backup

These tapes will be stored in the systems programming tape library.

- A. A cycle of at least 10 DS tapes are maintained. The description of the tape, when it is given to production, will specify the current DS tape.
- B. A cycle of 4 tapes is kept for each PL tape. This contains the last version of the tape for each of the last 2 major LSDs, the PL tape of the most recent minor LSD, and the current PL tape.
- C. The CDL tape is backed up along with the PL tapes, but a cycle of 10 tapes is maintained.

II. Off-site backup

There are two purposes for the off-site backup:

- A. To get operations up and going after a "disaster." The deadstart tape will accomplish this.
- B. To get the systems group up and going after a "disaster." The PL, CDL, and PFDUMP tapes will accomplish this.

The procedure for taking backup tapes to off-site storage is quite straightforward with few problems or inconsistencies associated with it. This procedure should be done at the beginning of each calendar month, usually within the first five working days.

- A. Insure that the most recent LSD has been tied off. This can be done by entering STAT from any Systems account. If all jobs have been completed successfully, then the backup procedure can be started.
- B. Login to SIN or SYSGEN on problem number 016115.
- C. Enter: START,BACKUP. & START,TESTPL. Each entry will initiate a batch job. If you are not using the ID SYSGEN, a warning message will come up. If you want the jobs to run anyway, enter "YES" when prompted.

BACKUP copies the current PL and CDL tapes to the backup tapes. TESTPL runs UPDATE on the current PL tape set to insure that each can be read and updated. The BACKUP job will write onto the tapes DEAD, SCOPEM, BASICEP, DARFAR, AMACRO, MOSSS, SCPPLS, CDL, VIMPFS, and VIMDOC.

- D. When these two jobs have been completed successfully then the Monthly AF dump needs to be run. The person initiating the backup procedure must make a request to Operations in order for this to be done. The operator will dump the authorf file to the AFMONTHLY tape.

Note: the UPHALx and UPAXxx tapes are not written by Systems. It is merely our responsibility to take these tapes to the vault. UPHALx tapes are maintained by User Services. UPAXxx tapes are assigned to individual users, who have responsibility for the contents of their own tapes. (Currently, use of the user off-site backup service is very low. Soon Operations will have their own off-site vault and will do the transporting of the UPAXxx tapes.)

- E. When this step is completed then the backup tapes can be taken to off-site storage. Currently the tapes are taken to the safety deposit vault in the Michigan National Bank tower in downtown Lansing.

There are currently fourteen tapes to be taken to off-site storage. At the present time, most of the tapes are located at the lower left corner of tall tape rack immediately to the left of the 750 operator's console in the machine room. Tapes with the VRNs specified below are to be taken to off-site storage:

1. DEAD - Deadstart tape
2. SCOPEM - PL tape
3. BASICEP - PL tape
4. DARFAR - PL tape
5. AMACRO - PL tape
6. MOSSS - PL tape
7. SCPPLS - PL tape
8. CDL - Correction Deck PL tape
9. VIMPFS - This PFDUMP tape contains numerous permanent files:
 - a) The SIN and SYS Hal libraries, and the main HAL library.
 - b) The SYSTEMS, SIN, A.F. Utility, and DUMPTAP libraries.
 - c) The versions of FRENDD on permanent files.
 - d) Files containing information about the SIN listing tapes.
 - e) Various files needed for systems generation.
10. UPHAL1 - UIC HAL files
11. UPHAL2 - UIC HAL files
12. UPA100 - User files for off-site backup
13. VIMDOC - Systems documentation PFDUMP tape
14. AFMONTHLY - AF dump tape

It should be noted that each tape has a duplicate tape in the vault. The two sets of tapes are labelled "Set 1" and "Set 2". At any time, one set is in the machine room and one set is in the vault.

WRITTEN BY: Glen J. Kime and Michael H. Giddings

APPROVED BY: Richard R. Moore

MICHIGAN STATE UNIVERSITY

COMPUTER LABORATORY

SYSTEMS TASKS, RESPONSIBILITIES AND PROCEDURES

No. 2.2

May 16, 1979

Installation of CDC Products and PSR Code

Introduction

This STRAP describes the various procedures which may be used to install CDC code into the MSU operating system. The emphasis here is on the system installation procedures; however, these procedures should be reviewed while working on such a project.

Installing a New Product or Program Library

The following procedure is used to install a new CDC product or program library. If the new product is just a new version of an existing product, you may want to treat it as an update to the current product, since the existing installation deck and MSU modifications may be relevant.

- Step 1. Obtain the CDC PL from the Systems Integrator. This tape will be a copy of the latest CDC PL tape released by CDC, and will be on a VIM tape.
- Step 2. Generate the installation procedure for the product. See the Installation Handbook to obtain important information about the product installation and to find the location of the current CDC installation sequence.
- Step 3. Generate the new product, making any changes needed to the code and installation procedure and test the product.
- Step 4. Send the necessary information to the Systems Integrator (via code review). This includes:
 1. The CDC PL name or VIM tape number (from Step 1 above).
 2. The modifications made to the PL (see STRAP 3).
 3. The installation procedure used. A listing of the control cards used is sufficient. This listing should make the following clear:
 - What texts are used, and where (what PL) they come from, if not from the existing system.
 - What compilers or other utilities are needed from other PL's.
 - What texts, binaries, libraries, etc., are created for use by other PL's.

- What binaries go onto the system, and, on what libraries.

Updating An Existing CDC PL To a Higher PSR Level

When installing a new level of CDC code, it is important to check for conflicts, duplication, etc., between new CDC code and existing MSU modifications to the program library. In particular, look for "overlapping corrections" indicated by UPDATE, and for resequencing or restructuring of the program library or product.

The procedure for updating a CDC PL to the highest PSR level is similar to that for a new PL, with a couple possible exceptions. First, the installation procedure listing is not needed if there are rio changes. See the "Installation Procedure" section for details on finding the current MSU installation procedure deck. Second, the existing MSU modifications to the product need to be handled. There are two primary methods for doing this, which are detailed below. The choice of methods must be made clear on the gift certificate accompanying installation.

Method 1: This method can be used if there are few existing modifications for the program library, or if major changes are needed to the existing modifications. This is the simpler method.

- Step 1. Retrieve the current modifications from the correction deck library. The following control cards will accomplish this:

```
GETPL,pname,CDL.  
UPDATE,F,*=$$$$.  
(modifications remain on COMPILE file)
```

- Step 2. Place the CDL modifications in an EDITOR work file, and revise the modifications as necessary. This may include deleting entire modifications or rewriting others. If a substantial portion of a modification is rewritten, the ident card and correction history should reflect the name of the person revising the modification. Any "*/" LSD comment cards other than "*/ D" cards should normally be removed, since the modifications are just being reinstalled.

At this point, the modifications are treated just as any new modifications would be. If you desire that some or all of these modifications not be subject to code review, and you have made no changes to them, mark that clearly on the listing submitted for code review.

Method 2: This method is used only if there are a large number of existing modifications and they do not require major changes. It is not as simple to set up, but reduces the number of modifications shown in the LSD, and reduces the size of the UPIC listing sent to code review. The existing modifications still need to be checked for duplicates, conflicts, etc.

- Step 1. Retrieve the CDL modifications and apply them to the CDC PL, generating a new "working" program library. If this is accomplished with no UPDATE errors, proceed to step 3.

Step 2. If there were UPDATE errors in attempting to apply the CDL modifications against the CDC PL, such as references to non-existent decks or idents, it is necessary to modify the CDL modifications before the second UPDATE. These modifications should be the minimum necessary to successfully generate a "working" PL. Remember that the control character for the CDL PL is "\$". If a modification has many UPDATE errors, it is best to just purge it and reinstall it as a new modification.

When preparing the product for code review, the UPDATE output from this first UPDATE must be included. Also indicate what permanent file and line range contains these modifications.

Step 3. Using the "working" PL as the base PL, generate any new modifications that are needed. This is where changes should be made such as correcting problems with CDL modifications which did not produce UPDATE errors, but do not produce the desired result. If a modification is outdated or needs substantial rewriting, it should be purged here, unless it had to be purged in step 2.

A sample deck structure is shown below to show how to use this procedure.

Job 1 - create "working" PL.

```

GETPL,plname,CDL.
UPDATE,F,*=$$$$ ,C=MODS.
RETURN,OLDPL.
REQUEST,OLDPL.ffl,VRN=VIMxxx. CDC PL tape.
UPDATE,N,C=0,I=MODS.
PUT,NEWPL=NEWPSRLEV.
*EOR
    (modifications, if any, to CDL, e.g. :)
$PURGE abc
$/ ABOVE MODIFICATION REFERENCES NON-EXISTENT DECK

```

Job 2 - Use "working" PL to generate product

```

GET,OLDPLrNEWPSRLEV.
UPDATE,...
    (Rest of installation sequence.)
*EOR
*PURGE xyz plname (modification no longer desired)
    (other modifications)

```

Installation Procedure

The creation of the installation procedure for a product deserves careful attention, since this will contribute much to the success of the project. This section gives some helpful hints for determining and creating the correct installation procedure.

The CDC installation control card sequences are contained on the BCC tape, usually as the third file. The control cards are contained on an UPDATE PL with a master control character of "=" . The decks have names of "PLnnI" where "nn" is the CDC PL number. Note that these decks are general-purpose installation decks meant for all CDC sites, and contain a number of options, selected by UPDATE =DEFINE directives. All of these make the decks somewhat incomprehensible, but with some effort, some sense can be made out of them. Refer to the Installation Handbook for details on the various options and for other important installation information.

For PL's that are not new to MSU, there are installation decks maintained by Systems Integration which describe the installation procedure. These decks are maintained on a HAL library (SIN) and have deck names of the form I*plname. Typically, there are at least two parts to any installation procedure, with subsequent parts numerically suffixed. These names are also limited to 7 characters. The decks will normally be retrieved to local file "plname". For example, to list the COMPASS installation decks, the following control cards would be used:

```
HAL , L*SIN, ICOMPAS, ICOMP2.  
LISTTY, I=COMPASS.
```

The following description outlines the types of control cards or control card sequences which comprise an installation deck. Not all of these occur for every PL, and sometimes additional processes are needed.

1. Get the PL and do the UPDATE. The PL is either the VIM copy of a CDC tape or the working PL. Typically, an UPDATE,F is desired.
2. Retrieve any texts, compilers, utilities, and user libraries needed for this installation procedure. These are created by other installation procedures or are available from the running system.
3. Compile or assemble the source. This will create binary files and listings of the product.
- M. Load the binary, creating an absolute program, if applicable. There may be other necessary post-processing necessary to generate the desired finished binaries.
5. Save any texts, compiler binaries, utilities, or subroutine libraries which are generated by this procedure and needed by other installation procedures. Subroutine libraries are typically saved in EDITLIB user library format.
6. Save the binaries which are needed for installation on the final system (for testing purposes). This may be included in #5.
7. Process the listings, map files, etc., if needed. This may include a UPIC or a full listing or saving the listing for later processing.

Summary

The procedures described here are designed to ensure that the necessary items are made available and ready so that the Systems Integrator can incorporate the finished, tested product into the system. The process is general enough to allow for the wide variety of installation procedures required.

If any deviations are needed from the procedures outlined in this STRAP, especially regarding the various items given to the Systems Integrator, these deviations must be discussed in advance with the Systems Integrator.

In general, the actual system generation will be accomplished using only the items specified above as being needed by the Systems Integrator. Test binaries, PL's, etc., are normally not used except in cases where a dependency problem makes it necessary.

WRITTEN BY: Douglas E Nelson

APPROVED BY: Richard R. More

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 3.5

Installation of Michigan State University Modifications

June 1, 1984

1.0 Introduction

In order to avoid confusion concerning modifications made to the different operating systems maintained by the Systems Programming Group a definite procedure has been defined and is used in all cases. This STRAP describes the procedure.

2.0 SMP requirements

Any user visible change or any major modification must have a Software Modification Proposal (SMP) written and approved before the modification can be considered for installation. The SMP should be written before the modification is coded but it MUST be approved before Systems Integration will accept the "Gift" Certificate. See STRAP 6 for SMP details.

3.0 "Monday List" requirements

This is a list of all modifications to all systems that are not minor bug fixes. This list is published the day before the Computer Lab staff meeting. All modifications except minor bug fixes or crash fixes must appear on this list before they can be installed.

All user changes must appear on this list.

4.0 Debugging

Debugging of a modification is the responsibility of the programmer. It is expected that the Project Leader will ensure adequate testing has been done on the final version of a modification. The code review process is not intended to find bugs in the code or errors in the design.

5.0 The Gift Certificate

After the modifications have been debugged to the best of the programmer's ability a "Gift Certificate" (or "Gift") will be filled out. As the form will reveal, its functions are many:

1. System Integration—the Systems Integrator will have a hard copy of all of the changes that are being made to the system.
2. Dependence—gives an indication of what other routines must be changed in order for this modification to work.
3. Identification—what the modification is called, who wrote it, where the source of the modification is, what the modification does.
4. Routing Information—who the modification goes to next (and a record of where it has been).

6.0 Code Review

why don't they go through first time?

The Gift Certificate along with a machine highlighted (UPICed) source listing is sent to several reviewers described below. All Gifts go thru this review path regardless of the status of the coder.

6.1 Design group review

The Gift is first sent to a "design group" reviewer. This should be another programmer familiar with the project or area being modified. If the Gift is work done on a multi-person project, then the first reviewer should be a member of that project group.

The design reviewer will thoroughly review the code for adherence to the following:

1. All subroutines and modules have complete initial comments.
2. All code has sufficient (but not excessive) comments.
3. The code is modular and not convoluted.
4. The code follows the published documentation (SMP and design documents).
5. Complete testing has been done with adequate testing under the latest system.

6. The modification follows requirements for publishing an SMP and being on the Monday list.
7. The product is well designed from a maintenance, reliability, and modifiability viewpoint.
8. The modification follows the STRAP 9 specifications.
9. The modification was adequately tested under the latest production system.
10. The modification does not unnecessarily require abnormal, tricky, or complicated installation procedures. If this installation procedure seems abnormal, systems integration must approve the process.
11. The modification fulfills any other requirements the project leader wishes to impose.

6.2 Project leader review

If the gift is marked "ACC" (accepted if comments corrected), it is then sent to the coder's project leader. If the project leader is not available or was the design reviewer, another project leader may be used. The project leader checks for the same things as the design reviewer; however, since any major problems should have already been corrected, this review should concentrate on the overall modification and not the details.

6.3 Systems manager review

When the modification fulfills the above two checks it is given to the systems supervisor. The systems supervisor checks for the following:

1. The LSD data will result in a good quality document.
2. The modification was done in proper priority and followed the constraints imposed.
3. Whether any other changes should be included in this modification.
4. The modification satisfies any external considerations.
5. Spot checks the other reviewer's work.

(It is, of course, best to insure that items 2, 3, and 4 are fulfilled before the modification is presented for approval--this should be done by the project leader).

6.4 System Integration

Systems integration only reviews whether the LSD data will result in a good document and whether the Gift indicates that it should be installed.

6.5 Code review routing and problem notation.

If the proposed modification fails to meet any of the review criteria, the reviewer should explain the problems on the listing. If it is a specific problem in relation to a given set of code, the explanation should be written beside the problem code; a general, overall problem may either be noted at the beginning of the listing or at the first incidence. In any case, if the coder knows of a similar problem it should also be corrected even if it has not been found by the reviewer!

A problem may be considered "fatal" or "informative" by the reviewer. An informative problem is one that does not require correction before the code is installed, but should be noted by the coder so that future modifications will be correct. Fatal problems must be resolved before the code is installed. A reviewer further along the review chain may upgrade an informative problem to fatal; downgrading should not be done without discussion and agreement of the previous reviewer. Usually, the ink color will indicate whether the reviewer's comment is fatal or informative. Current tradition has orange, red, and purple being reserved for fatal comments; green, black, and brown indicate informative comments.

If the fatal problems are minor and can be corrected by fairly simple changes, the reviewer should mark the ACC box on the gift certificate and route the gift to the next reviewer. Otherwise, the gift should be returned to the coder.

If significant problems are found at a level beyond the design reviewer, the gift should be routed back thru the design reviewer. This will help these reviewers to improve their skills.

If the coder disagrees with a reviewer's comment, the coder is encouraged to discuss the situation with the reviewer. Discussion is encouraged because this will allow both people to improve their skills. If no resolution is achieved, an appeal may be made to the systems manager.

6.6 Installation

When the modification is fully approved, the Systems Integrator (S.I.) will hold the modification until all Installation dependencies (other modifications needed, Monday list timing, Major System needed) have been met. If there are extensive dependencies or a large number of modifications that have to wait several weeks, the S.I. may send the partial package back to the project leader. When the P.L. determines that all the Gifts are ready, the entire package is sent directly to the S.I.

After installation the Gift, marked as installed, along with the listing, will be returned to the Project Leader. The programmer should note any comments on the listing for future reference.

6.7 Exceptional problems

Occasionally, a modification may get lost in this routing process. The submitter is encouraged to follow up on any change that is not heard from for over one week.

7.0 The Modification Deck

Modification decks will be 'usable' by UPDATE. In addition, each IDENT will have the following five features to make installation and documentation as easy as possible.

1. The first line of the modification deck is an *IDENT card. The *IDENT line is of the following format:

```
*IDENT idname PLname
```

The PLname is the name of the UPDATE PROGRAM LIBRARY this modification affects. The PLname must start in or before column 73. *IDENT can be abbreviated.

NOTE: The optional UPDATE parameters B=, K= and U= may not be used without the approval of the Systems Integrator.

All *PURGE and *PURDECK lines must also contain the PLname starting in or before column 73.

"COMPILE lines may also use the PLname if the decks mentioned are not on the same PL as that appearing on the *IDENT line. This allows reassembly of routines on different PLs in order to make use of changes to texts or the COMDECK PL.

2. *DECK lines should always have the identifier "name.1" in the deck "name." When an identifier is also a deck name, lines from that ident should only appear in that file.

To facilitate LSD writing and documentation, UPDATE comment lines are added to the modification file. The initial deadstart, operational and user changes sections are 'stand alone' and should be neat and consistent with the remainder of the LSD. All of the comments of each type are gathered from each ident and then printed in the appropriate LSD section with other comments of the same type. The format of the comment line is:

```

1 4 6
*/ x ccr
7
2
c

```

Where C is the comment and X is the type of comment. Note that there is a blank between the X and the C—this is required. Also, all comments that are not of a required format must be complete good English sentences. Jargon and abbreviations should be avoided.

Comments of type 'D', 'G', 'O', 'U' should be in "upper/lower" format.

Below are the different values of X:

D - Description of the modification—this is the part that goes in the section of the LSD under IDENTIFIERS. It must describe to a reasonably knowledgeable person (someone who has taken the Systems student training program) all changes that were made. This must describe the changes; simply saying something like "Install modification to allow CYBER Loader to run on our system" is not acceptable. The first "*/" lines in any ident must be "*/ D" lines; however, it is not necessary to place all "*/ D" lines at the beginning of the ident.

E - EDITLIB lines in the following format; beginning in column 6 the EDITLIB Library Name affected (ex: NUCLEUS, PPLIB or SYSOVL) followed by a slash (/) and then the EDITLIB command. There should be no blanks next to the slash. These should be used to DELETE binaries on the deadstart tape that one no longer needs.

Example:

If routine A is to be moved from the NUCLEUS library to the FORTRAN library, along with the necessary modification to PRE, the source for the gift should contain a *COMPILE line to reassemble A and the following '*/ E' line.

```
*/ E NUCLEUS/DELETE(A)
```

F - Dayfile messages in the following format; beginning in Column 6 of the first comment line should be the dayfile message exactly as it will be in the dayfile. Column 10 or after of subsequent lines should be a further clarification of the message (but the message should be perfectly clear in itself). This clarification should be in "upper/lower" format.

Example:

```

                                1
                                6  0
*/ F PRE - INVALID CHARACTERS
*/ F      O\CCURS WHEN NON-ALPHANUMERIC
CHARACTERS
*/ F      \OCCUR IN A PARAMETER.
```

G - General comments--this is the part that goes in the section of the LSD under GENERAL. These are normally inserted by the Systems Integrator, and are used to indicate the major changes or "Monday list" items which appear in the system.

General comments should be kept short and of general interest to most knowledgeable users.

L - Level number changes. Whenever a dependent product PL is updated to a new PSR level number, a '* / L' line must be included. The level number should be in column 6 and the PLname in column 10. Example:

```
*/ L 420 FTN
```

O - Operational changes--this section must describe all changes that will affect the Operator, such as:

- changes to DSD visible to the operator when the keyboard is in LOCKed mode (new or changed commands or displays).
- new or different messages that the operator must respond to.
- new or different procedures for initializing attached mini computers.
- how to use programs that work only (or differently) from the console (or operator's terminal).

S - STIRs fixed by this modification. The format is one STIR number per line beginning in column 6.

Example:

```
*/ S 7628
*/ S 8219
```

The type D lines should also contain the sentence "This fixes STIRs 7628 and 8219".

U - User changes—this section must describe all user visible changes. It should be complete enough that a knowledgeable user (such as a consultant) would recognize and understand the change once seen.

blank -UPDATE comment lines with blanks in columns 3 and 4 may be used freely within an ident (following the initial "*/ D" lines). These lines will be saved on the Correction Deck Library, but are otherwise ignored.

3. A correction history section should be included in each modification. The format for these comments:

```
col. 1 3 6
      * author - date
      *   comments describing the modification
```

The date should be mm/dd/yy. Two digits are preferred (05/08/73). The correction history section does not occur in all routines. Any modifications made to routines that do not have the correction history should also add this section..

New routines may also include this section at the coder's option. The format for the correction history section:

```
col. 1                               2
                                      0
      *IDENT CHnnnnnn                PLname
      */ D START CORRECTION HISTORY FOR 'nnnnnn'.
      *I nnnnn.xx
      **CH*                           CORRECTION HISTORY:
      *
      *
      **CE*                           END OF CORRECTION HISTORY
      *C nnnnnnn
```

These lines should have the ident "CHnnnnnn" where "nnnnnn" is the deck name of the routine. All correction history comments will be inserted by the line "*I CHnnnnnn.2" (This will give the most recent modification first.)

When starting a correction history, try to keep it on the first couple of pages of the listing. Special warnings such as BASE changes or non-standard column conventions should immediately

precede the correction history so they will be visible in a UPIC listing.

When modifying the texts, a correction history is also included. The insert point is *I CHSYMBOLS.2 if you are adding new symbols. *I CHMACROS.2 is the point when the macro changes are made. By compiling the deck TEXTCH (*COMPILE TEXTCH or *COMPILE =TEXTCH). The correction history may be seen.

4. The last line(s) of each modification is the appropriate *COMPILE line(s).
5. Use of the following UPDATE directives is prohibited (without approval of the Systems Integrator):

ADDFILE	ENDIF	READ
CHANGE	ENDTEXT	SELPURGE
DECLARE	IF	SELYANK
DEFINE	MOVE	SEQUENCE
DO	NOABBREV	TEXT
DONT	NOLIST	

8.0 Installation

There are three 'levels' of installation.

1. Crash Fix—a modification may be installed in the middle of the week if it fixes the cause of system crashes or a very major bug. The modification must be walked through the approval process by the Project Leader or the coder.
2. Weekend systems—general bug fixes and improvements are installed in a system that is used for Saturday and Sunday production. Modifications for this system must reach the Systems Integrator fully approved by Tuesday 16:00.
3. Major systems—non-upward-compatible user visible changes (such as a new version of a compiler that won't accept all source statements the old version would) are saved for major systems i.e. LSD XX.00.

In all cases the Systems Integrator will make up the appropriate PL tapes after the deadstart tape is created. Systems integration will also maintain permanent file copies of certain current PLs. There is a transition period between systems in which we may not be sure which system is on the permanent file. As a general rule, the PFs will match the system that is currently fully operational.

WRITTEN BY Mark R. Riordan

APPROVED BY Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 4.1

Writing 6000 SCOPE Memos

March 28, 1977

A 6000 SCOPE Memo (6SM) or an M4 (see STRAP 11) is written for each system alteration. The 6SM should usually cover a project. However, when there exists a 6SM on some section of the project, then the existing 6SM should be brought up-to-date and the project 6SM should reference the existing 6SM. For example: the installation of the CYBER Loader modified many routines one of which was PRE. The 6SM for the CYBER Loader project should simply refer the reader to the PRE 6SM. The PRE 6SM would be brought up-to-date.

A 6SM is brought up-to-date by either an M4 appendix, a 6SM appendix, or by republishing the 6SM with revision number incremented by one. If the changes are extensive, a revised 6SM is preferred. If the 6SM is available in machine readable easy to update form, it should be revised. The revision can consist of change pages so that the entire 6SM does not have to be republished. Each 6SM is numbered by the Technical Assistant. The number has the form "n.r". "n" is the number of the 6SM in the sequence of all 6SM's; "r" is the revision number for this specific 6SM.

The 6SM serves many purposes. It provides a complete internal and external description; it gives the reason for the modification; it tells where to go to get further information. The 6SM can be used by the person writing user documentation; it can be used by the person learning about how this modification was done; it can be used to learn about the considerations that went into the creation of the modification. It can be used to describe to other installations what we did and how we did it so that they can judge whether they might attempt to install the modification.

However, the 6SM is not to replace or duplicate the incode documentation.

The 6SM for any low priority projects must be published before the code is installed. The 6SM's for high priority projects must be published within 4 weeks of installation. The 6SM is written by the coder (or coders) and is approved by the Project Leader that reviewed the code. The systems supervisor then reviews the 6SM before it is published.

The 6SM should have a heading that identifies the 6SM: the number, the title, the date written, and MSU Computer Laboratory. The bottom of the first page should have this copyright notice: "COPYRIGHT year, MICHIGAN STATE UNIVERSITY, BOARD OF TRUSTEES." (The "year" is the year the 6SM is published.)

This section format must be followed when writing 6SM's. This helps insure that all important points are covered.

1.0 INTRODUCTION

This section should briefly describe the purpose and justification of the modification.

2.0 EXTERNAL REFERENCE SPECIFICATIONS

This section should describe the external characteristics of the modification. If the change is strictly "internal", there should still be some external features. For example: "This modification will eliminate all channel hangs" or "It is possible to turn off the device and still accomplish the XYZ task by using the old ABC device."

If there are user effects, a manual change could be written from this section.

If this modification is a new user callable routine, this section is a complete description of how the user can use this routine.

If this is an extensive modification of an existing routine (usually supplied by an outside source) then it may be best to completely describe the external specifications.

3-0 SYSTEM PROGRAMMING CONSIDERATIONS

This section will mention the specific routines modified or added and the LSD under which it was installed. Any assembly options, other changes required, and general cautions should be described here. If general tables such as the control point area have been altered slightly, it should be noted here along with a picture of the new part of the table (at least one word) and what modules use this altered information.

4.0 INTERNAL REFERENCE SPECIFICATIONS

This section describes what internal changes were made. However, it must not be too detailed (most early 6SM's suffer from being too detailed). To use the forest and tree analogy: you should describe the various meadows and groves you might find, certainly warn where the dangerous bears lurk, possibly tell about a very notable tree. But, you should not describe each tree and bush,

nor should you tell about the bark and leaves. A more concrete example can be found in 6SM 99. However, that does not completely follow the outline below.

The data structures used should be described in pictures (preferably plotted) and what routines use or modify which fields should be told.

Organization of this section (assuming a major modification):

"First, a very general overall flow (again with pictures and examples) is given. Second should be the pictures of the tables with references to the descriptions of the modules that use these tables. Third, if necessary, a table of contents for this section which gives the section numbers of the descriptions of the various modules. Finally, the overall flow is broken down into the modules created when you did the top down design. Usually only the first three (possibly four) levels need to be described. Frequently, a block module picture will be useful.

The purpose of this section is to allow one to become familiar with the "forest" to the extent that a small local alteration can be made knowing only the immediate vicinity in detail.

5.0 OPERATOR COMMUNICATIONS AND PROCEDURES

This section should describe how this modification alters the way operations does things. It describes any new messages and displays. It tells what actions are requested of the operations staff.

6.0 USER ASPECTS

What benefit will the users see with this modification? This section should explain why the users should feel this change has helped them. It should also summarize and describe alterations that will be found by the user.

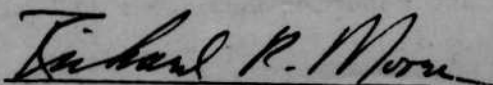
7.0 SYSTEM FILE CHANGES

This section will describe any additions or changes to the various system files such as the dayfile or the C.E. error file.

8.0 REFERENCES

This section should refer the reader to any associated 6SM's. It should also give the software modification proposal number. Both titles and numbers should be given when possible. If this 6SM will refer to a yet to be written 6SM, it is possible to reserve 6SM numbers ahead of time, but you do commit to writing that 6SM.

WRITTEN BY:



Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 5.2

Maintenance of Systems' Tape Library

June 5, 1984

1.0 Description.

The Systems group tape library consists of all tapes with a prefix of "VIM." Frequently used tapes are kept in the machine room, in racks separate from user tapes. Less frequently used tapes are stored in Room 301. They are arranged by number; each tape has a unique position in the racks.

A list of tapes is kept by the technical assistant (T.A.). The list contains the VRN (visual reel number), owner, use, date tape was assigned, length, location, and density for each tape. Two full lists are posted: one on the Systems' bulletin board in Room 301, and one on the Systems' tape racks in the machine room. A separate list of available tapes is posted on the bulletin board. These lists are updated weekly.

2.0 Procedures for Library Users.

2.1 Each tape in use has a sticker of the form:

VRN	
Usage	
Owner	Date

A VIM tape without such a sticker is available for use.

2.2 A person needing a tape can take any tape without a label as described above. The available tape list on the Systems' bulletin board should be consulted to verify that the tape is available. A person taking a tape must cross the tape name off the available list, and put a sticker (as described in section 2.1) on the tape. They should also send a note to the T.A. saying that they have acquired the tape. The note should include their name, the tape VRN, its new use, and tape location or density if either will be changed.

2.3 A person releasing a tape should:

- 1) blank label the tape (ensure PN=000000)
- 2) remove the paper label
- 3) notify the T.A. that it is free (send a note)
- 4) hang the tape on the tape racks where the other AVAILABLE-FREE tapes are.

2.4 Every month every person holding tapes will receive a copy of the list of tapes he holds. If the owner wants to update the information on the list (or return a tape to AVAILABLE), they should mark each correction on the list and return it to the T.A., or just send a note.

3.0 Procedures for the Technical Assistant.

A. The technical assistant will maintain a list of the contents of all the Systems group's tapes, update the list weekly, and distribute to each person holding tapes a list of the tapes they hold monthly. A weekly available tape list will be posted on the Systems' bulletin board. Three weekly full lists are sent to the Systems Supervisor, the Systems' bulletin board, and the machine room tape rack.

B. Current procedures for maintaining the listing are as follows:

A Query Update procedure is in use that generates master lists and individual lists.

- 1) The tape list and QU procedures are maintained on the permanent file TAPEASSIGN. This file is an Editor work file.

Information kept on each tape is VRN, user name, label description (use), date created, length, location, and density in that order. Tab stops are set up at all fields except VRN and NAME to facilitate updating information.

By convention, a tape with a VRN of VIMuuu is described on the workfile line luuu. For example, VIM247 is described on line 1247.

- 2) The full set of listings can be obtained by attaching the permanent file as EWFILe and saying GO,"MONTH".
- 3) The weekly set of listings can be obtained by attaching the file and typing GO,"WEEK".

Available tapes are assigned to AVAILABLE, use description FREE. Special tapes are assigned to MOORE-ARS (for archive storage).

- C. Each week the T.A. will examine a portion of all the tapes and update the list as needed to reflect the contents of the labels on the tapes.
- D. When the T.A. receives a note to make a tape available free, the T.A. should insure that:
 - 1) The paper "stickum" label has been removed.
 - 2) The tape has been blank labeled, i.e., the PN in the label is 000000.
 - 3) The tape has been placed with the other AVAILABLE-FREE tapes.

If any of the above have not been done, the T. A. should do them, after first ensuring that the tape really was owned by the person making the request. This is done by checking the stickum label, the PN on the tape label, and the tape assignment list.

Finally, the entry for the tape in the file TAPEASSIGN should be changed to have an assignee of the AVAILABLE, and use description of FREE.

WRITTEN BY: Glen J. Kime

APPROVED BY: Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 6.2

The Software Modification Proposal Document

June 8, 1984

A proposal must be written for any additional user software product or user visible change of any product. Also, a major modification of any product that is not user visible is proposed in writing. This proposal is then circulated for additional comment and viewpoints. This way all factors can be considered and modifications are not made in a vacuum.

Procedure

The Software Modification Proposal (SMP) is written following the format outlined below. The SMP is then approved by the project leader; informal review is also done by the system programming supervisor. Then the systems programming supervisor assigns the SMP number and attaches and fills in a cover sheet for reviewers' comments. The reviewers are expected to return the cover sheet by the final date for comments.

If the reviewers' comments indicate that a meeting to explain things further or to resolve some differences is necessary, a meeting of the reviewers will be held. Otherwise a reply to these comments will be written by the writer of the SMP.

Approval, Disapproval

Approval, or disapproval of the proposed modification is given, in writing, by the Director of the Computer Laboratory or their designee.

Approval does not mean approval to work on the modification. It simply means the proposal is deemed to be reasonable and is approved. Work on the modification must only be done when it receives high priority or when it is contained within some other high priority work and adds no more than 5% implementation time onto that work.

Disapproval of the proposed modification means that the modification is not acceptable in its present form. If the proposal is not to be rewritten, then it should be dropped.

Format

The following format should be followed when a Software Modification Proposal is written. The sub-paragraphs should be numbered.

1.0 Introduction

This should give an overview of the modification.

2.0 Present Condition

Describe the present state of affairs with which the proposal is concerned. This should show why the modification is needed.

3.0 Proposed Modification

Describe how the proposed modification will change the current system. How this will help the problems mentioned earlier should be noted. A precise description of the user interface must be included. In most cases, examples and error diagnostics should also be given.

4.0 Implementation Details

4.1 The modifications to specific routines and any new routines necessary should be briefly described.

4.2 Estimate the amount of time to implement the change and the amount of time before the full change is available for use. For example: 7 work-weeks of time—the change will be available 10 weeks from start of work; four full-time weeks, six half-time weeks.

Also give the dedicated and production computer resources.

4.3 Describe any special procedures that will be necessary for testing purposes.

4.4 Describe any additional, special effort required by other groups. Specifically, the documentation time for Technical Communications must be included.

A rough draft of the SMP is given to the Technical Communications section with a well defined user interface. They will return a detailed work time estimate as well as a list of the manuals that need to be changed. This should be included in this section.

5.0 The Effect of this Change

5.1 On the user

5.2 On operations

5.3 On the accounting system

5.4 On the system files (dayfile, C.E. error file, etc.)

6.0 Summary

This should summarize and motivate the proposed change. It should give the benefits and costs of the modification.

The SMP usually serves as the preliminary user documentation for user services. Any change in this documentation must be preceded with a memo to all reviewers describing the change.

The permanent file SMPSKELETON contains an Editor workfile with an RNF skeleton for an SMP. You should use FCOPY on this workfile and then fill in your copy as needed.

WRITTEN BY: Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 7.4

Monthly Report Content and Style

June 7, 1984

Everyone in the systems group writes a report at the end of each month describing all the work they have done during that month and the work they expect to accomplish next month. These reports are written using RNF in a manner described below. They are to be completed and given to your project leader or (if you have no project leader) the Systems manager by the end of the second working day of each month. The report should cover all activities of the previous month.

The report is completed when a rough draft has been approved by the project leader or manager.

These individual reports are merged into a final Systems Programming Monthly Report by the manager of systems development. This final report is distributed to other areas within the Computer Laboratory and to the Computer Operations and Finance committee. This report must be published by the 10th working day of the month.

All reports must use good English sentences. The reports may be edited if they are considered inappropriate for general distribution.

Descriptions may depend upon previous months' reports (with proper citation), but they should be as free of jargon as possible. They must not be a repeat of the previous months' report.

1.0 Preparation using RNF

The permanent file "MONTHLYREPORTSKELETON" should be copied (using FCOPY on HAL) to your local editor workfile. This file contains lines that will show you where to place the sections of your report. You should never delete ANY lines that were in the skeleton.

After you have finished writing the report, run it thru RNF and send this rough draft to your project leader or the Systems manager.

When the rough draft is approved, and not before then, the source from the ewfile (lines 100000-*1) should be saved and cataloged with the name "iiiMRnn". "iii" represents your initials and "nn" is the month number

(with a leading zero if necessary). If, for some reason, you did not use the current "MONTHLYREPORTSKELETON" permanent file, then you should save the source for only those lines you entered - none of the skeleton lines should be saved.

For example: when the rough draft of the August monthly report was approved, the commands:

```
SAVE,MR08,SO,100000-*L.
```

```
PUT,MR08.
```

would be done. NOTE: the line range MUST be 100000-*L, and the SOURCE must be saved. Do NOT save with the line range described by the string with your initials.

When the monthly report is published, then you should purge the permanent file you created.

2.0 Monthly report format and style:

The monthly report contains six sections. It is important that you follow precisely the style outlined so that the final report is a unit.

2.1 Written review and plan for next month section

In this section the programmer briefly describes how well the expected accomplishments were met during the month and specific projections of work during the next month. The reason for any deviation from projected goals should be noted here.

This section begins on line number 100000 in the monthly report skeleton workfile. The place where you should add your part is just after the line that contains ".rp iii" (iii is your initials).

There should be at least two paragraphs written. The first will describe what you accomplished this month, contrasting the accomplishments with the expectations mentioned the previous month. The second paragraph will contain the specific projections for next month.

2.1.1 Style:

```
.rp .iii
```

This month fixes for STIR'S 1125, 3943, and 4047 were installed as predicted. However, the PF RENAME feature was not completed because machine time was usurped by high priority projects. The time was spent analyzing STIR'S 4048 and 5143 instead.

Next month the RENAME project should be installed as well as fixes for STIR'S 4048, 5143 and 5021. Investigation will begin on the ten high priority SELDUMP stirs.

2.2 Project summary

This section describes, by project, the work done on that project. It should also include, in a separate paragraph, a review and projection for the project for major projects.

If you are describing a module of a large project which you have worked on with other people, then you will be describing a subproject. In this case, the global description for the project would be written by the team and included in one member's report. This global description would contain the review and projection. The Systems manager will edit the pieces into a whole report.

This section begins on line number 200000 in the monthly report skeleton. The place to add your report is indicated by a ".rem" line with your initials. Refer to the RNF macros section for an explanation of the ".proj" and the ".subproj" macros.

2.2.1 Style for a simple project:

```
.proj (project title)
.iii
```

[description of work done, problems encountered, solutions to problems, techniques discovered, etc.]

[review and plan for next month]

2.2.2 Style for a large project with separate modules worked on by several people:

```
.proj Project title
```

[possibly a general description]

Review and plan for next month

[review of previous months goals and accomplishment of them. Realistic expectations for next month.]

```
.subproj Sub-module 1
.iii
```

[description as in simple projects]

```
.subproj Sub-module 2
```

```
•jjj
```

[description]

2.3 Maintenance summary

This section describes, by project or item, the work done on maintaining the current system. This includes STIR fixes, system creation, products assigned a maintenance status, etc.

This section begins on line number 300000 in the monthly report skeleton. The place to add your report is indicated by a ".rem" line with your initials. Refer to the RNF macros section for an explanation of the ".proj" and the ".subproj" macros.

2.3.1 Style:

```
.PROJ System Generation  
.iii
```

[description of system generation work]

```
.stir 4135 - Incorrect buffer parameter error:  
.iii.cr
```

[description of work done to fix this STIR]

2.4 Work summary

This is a summary, by individual, of the projects assigned to that person, the work done, the priority, and the percent of effort spent on each project.

This section begins on line number 400000 in the monthly report skeleton. The place to add your report is indicated by a ".worksum .iii" line. Refer to the RNF macros section for an explanation of the ".worksum" macro.

2.4.1 Style:

```
.worksum .iii  
project 1 .t - .t W .rt VH .rt 75 .cr  
project 2 .t - .t C .rt H .rt 20 .cr  
project 3 .t - .t - .rt OG .rt 5 .cr  
project 4 .t - .t N .rt M .rt 0 .cr
```

These tabs position the development plan item number, the work done, the priority, and the percent of FTE spent on the specified project. If you set the tabs in SCREDIT to 54,63,69,76, and 80, the "dup-tab" SCREDIT feature will be effective and the final ".cr" can be omitted (the first *.t" would begin in column 55).

2.4.2 Work summary columns

2.4.2.1 Development plan item number:

Since the development plan item numbers have not been assigned recently, the first tab position has simply a dash unless the item number is known.

2.4.2.2 Work done column:

The allowable codes in the "Work Done" field (the codes must be capitalized):

W - worked on

C - completed (a project is completed only after all documentation is done)

N - no work done

- - a dash is be used for ongoing work

A project must remain on the list until it is completed.

2.4.2.3 Priority assignment column:

The allowable codes in the "Priority" field:

VH, H, M, L, VL, OG

where H, M, L are high, medium and low and the V means very. OG means an On-Going project.

2.4.2.4 Percent of work column:

The "Percent of Work" column gives the percent of a full time effort spent on that project during the month. If a half time person is reporting half time work for the entire month, then the total of the percent worked column should be 50. The accuracy, if such records are kept, can be to one digit to the right of the decimal point (i.e., 75.2 percent).

People should not include unpaid items such as studying for exams, etc. However, paid holidays and vacations should be included.

For example: assume that there are 22 weekdays in the month. If you worked for 10 days completely on project A, 11 full days on project B, and 1 day was a holiday, your percent of work would be calculated by:

$$\text{project A} \quad \frac{10 \times 8}{22 \times 8} = 45\%$$

$$\text{project B} \quad \frac{11 \times 8}{22 \times 8} = 50\%$$

$$\text{Holiday} \quad \frac{1 \times 8}{22 \times 8} = 5\%$$

If, however, you worked that holiday (on Project C) then you would add to the above summary a line for project C:

$$\text{project C} \quad \frac{1 \times 8}{22 \times 8} = 5\%$$

Therefore your percentage would total to more than 100%. Since the work done column represents the percentage based on a full time equivalent, whenever you work more than an FTE, the percentage will total more than 100%.

If during the next month (21 days) you worked 12 days on C, 8 days on A, and took one day off for comp time, then your work summary would be:

$$\text{project C} \quad \frac{12 \times 8}{21 \times 8} = 57\%$$

$$\text{project A} \quad \frac{8 \times 8}{21 \times 8} = 38\%$$

You do not show the comp time; it is implied by the fact that the total does not equal 100%.

2.5 Documents published

This is a list of documents published by you during the month reported. It should include the series identification (SMP, SMD, etc.), the titles, and the date published. The document must have been published; reports still being typed or printed should not be mentioned.

This section begins on line number 500000 in the monthly report skeleton. The place to add your report is indicated by a ".rem" line with your initials.

2.5.1 Style:

SMP 137.1	project 1 title	3/24/83
SMD 167.	6SM title	3/15/83
User Notice 3	title	3/28/83
misc. report	maximizing thruput	3/17/83

The actual positioning is not relevant; this section will be edited by the Systems manager.

2.6 Miscellaneous

This section contains any miscellaneous comments, general problems, accomplishments, etc.

This section begins on line number 600000 in the monthly report skeleton. The place to add your report is indicated by a ".rem" line with your initials.

2.7 Special reports:

The people responsible for crash analysis on the various computers must include summaries of system reliability (which gives crashes incurred, fixed, etc.) and a list of all crashes for the month. These reports will be included in the maintenance section. The actual style is particular to the computer system.

The person responsible for STIR's must include an accurate summary of the STIR activity during the month.

3.0 RNF Macro's

Names: Macros for each person's name are defined as ".iii" where iii is the person's initials. These should only be used with the ".PROJ", ".SUBPROJ", and ".STIR" macros.

Projects: Each major project should begin with the macro ".PROJ". The remainder of the line will be taken as the project title. If you want your name to be part of the title, place ".iii" on the NEXT line; otherwise, put a blank line following this title line.

Subprojects: Modules within a major project should begin with the macro ".SUBPROJ". The remainder of the line will be taken as the title for the subproject. If you want you name to be part of the title, place ".iii" on the NEXT line; otherwise, put a blank line following this title line.

STIR's: Each STIR should begin with the macro ".STIR" followed by the STIR number and then the title. The remainder of the line will be taken as the STIR title. The following line must contain the macro ".iii".

Work summary: The skeleton contains the ".WORKSUM .iii" macros for each person in the systems group. This macro will set up the tab stops for the various columns of data. You should use right tab (".rt") commands for the priority and percentage work columns. Each summary line should end with ".br", ".cr", or end in column 80.

Review and Plan: The skeleton contains the ".RP .iii" macros for each person in the systems group. This establishes where each person should place their review and plan comments.

Others: The macros described in the "RNF Macros" memorandum are also available.

If these guidelines are followed, the final report will be easy for people outside of the systems group to read. The monthly report is the main vehicle to tell others about our work to facilitate their usage of the computer systems. It is important that this report be clear and concise.

WRITTEN BY: Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 9.4

Guidelines for Coding Practices and Conventions

June 8, 1984

The charter of the Michigan State University Computer Laboratory Systems Group is to best support the system software needs of the Lab. This means the development and maintenance of large amounts of code with high reliability and modifiability.

The most important criteria to be considered when designing and coding then follow:

1. High reliability. Program units must work accurately, according to the designed external specification, with no harmful side effects. This includes both ensuring that the overall stability of the system is not decreased by a modification (i.e., don't cause crashes!) and ensuring that existing, working programs are not broken by any changes.
2. Modifiability. Program units are continually being changed, to fix bugs and to add new features. A routine which cannot be easily modified is a liability to our system.
3. Usefulness. To be worthy of the time in programming (and in later maintenance), a routine should provide a significant benefit (directly or indirectly) to the user community. Since we are in business to provide computing service to the University users, our time must be spent on projects which most benefit the user, even if those projects are not necessarily the most exciting or fun to code.
4. Ease of use. Any new feature available to the user should be simply described—always rough out the user documentation during the design. Difficulty in writing a simple, easy to understand external specification usually indicates a product that users will never be able to use confidently and without error (i.e., a worthless product).
5. Efficiency. In most cases high efficiency is a low priority goal. However, since good turnaround and response time are very important, modifications in certain critical areas must be designed so they do not reduce the effective amount of computing power to the user. Note that in almost all cases, efficiency goals are handled at the design level, or by not doing a project at all if it will detrimentally affect

turnaround. Trying to squeeze a few microseconds out of a piece of code by "tricky" programming typically results in no significant gain at a great cost in reliability and modifiability.

Simplicity and Precision

The above goals lead to two primary principles for design and coding: simplicity and precision. These two principles are the primary basis for the rules laid down in STRAP 9.

Simplicity. Complexity is probably the greatest single foe to each of the above mentioned goals. Be modest. Design small modules that have a single, well-defined function.

Simplicity should apply to external specifications, internal design, module function, module entry/exit conditions, programming technique, etc. A good structure design chart, written before coding begins, will insure that your modules are simple.

Precision. The other major enemy of successful systems programming is imprecision (vagueness, indefiniteness, or downright slop). A design is of little value if specifics are not clearly laid out.

Precision in programming is of tenfold more importance. Exact definition of data and processes is vital to the health of any non-trivial (greater than 30 lines?) program. All variables need their use described, including type, possible values, context of use, etc. Every routine must have an itemized list of entry conditions, exit conditions and how they are related (i.e., what the routine does).... This is the key to modular programming. Violate it and instead of a simple collection of simple modules (which is generally easy to follow and understand), you will have essentially one giant program (since the reader must keep all of the unwritten specifications in his head) which will prove to be hard to maintain, if not impossible to debug.

These two primary principles, simplicity and precision, are the key to designing and programming code that can be debugged, documented, and maintained.

Table of Contents

1.0	Comments within a Program	7
1.1	Global comments:	7
1.2	Within the Code	8
1.2.1	Subroutines (or other logically complete sections of code)	8
1.2.2	MACROS	10
1.2.3	Tables	10
1.2.4	Narrative	10
1.2.5	Correction History	10
1.3	UPDATE comments:	11
1.4	DOCK	11
2.0	Design and Coding Practices	12
2.1	Structure	12
2.2	User messages and diagnostics	13
2.3	Error detection, analysis, and processing	13
2.4	Copyright	15
2.5	UPDATE common decks	15
3.0	Symbols and Names (including routines, variables, macros, etc.)	15
3.1	General naming rules	15
3.2	Symbol usage	17
4.0	UPDATE Conventions	18
4.1	Deck Resequencing	18
4.2	UPDATE Abbreviations	18
4.3	Correction Deck Structure	18
4.4	Common decks	19
4.5	*YANK Identifiers	19

4.6	Miscellaneous.....	19
5.0	COMPASS Conventions.....	20
5.1	General Guidelines.....	20
5.1.1	Be clear.....	20
5.1.2	Be careful.....	21
5.2	General in-code comments.....	22
5.3	Global Restrictions.....	22
5.4	Symbol definition and usage.....	24
5.5	Scope text symbols.....	25
5.5.1	Symbol conventions to reference table entries.....	25
5.5.2	Other text symbol conventions:.....	25
5.5.3	Table prefixes in symbol table.....	26
5.6	General pseudo-op usage.....	27
5.6.1	DATA and CON usage.....	27
5.6.2	BASE and CODE usage.....	28
5.6.3	COMMENT usage.....	28
5.6.4	VFD usage.....	28
5.6.5	USE, ORG and other block counter operations.....	28
5.6.6	LIST options.....	28
5.6.7	QUAL pseudo-op.....	29
5.7	Macro definition (including OPDEF, CPOP, PPOP, etc.).....	30
5.7.1	General.....	30
5.7.2	Things you must not do with macros:.....	31
5.8	MACRO usage.....	31
5.9	CDC Central Processor Coding.....	31
5.10	CDC Peripheral Processor Coding.....	33
5.10.1	Direct storage usage.....	33

5.10.2	General practice	35
5.10.3	New or significantly altered PP routines	37
5.10.4	PP Macro Usage	37
6.0	INTERDATA coding	37
6.1	Mechanics	37
6.2	General restrictions	38
6.3	Alignment problems	39
6.4	Loading halfwords	40
6.5	Logical and arithmetic compares	40
6.6	BXH and BXLE	40
6.7	Register Conventions	41
6.7.1	R0 and R1	41
6.7.2	ISR's and SVC's	41
6.7.3	Subroutine Parameter Registers	41
6.7.4	Scratch Registers	41
6.7.5	RC Problems	42
6.8	Disabling Interrupts	42
7.0	<u>FTN Coding Standards</u>	43
7.1	Standard conforming	43
7.2	Indentation	43
7.3	Symbols	43
7.4	DO loops	43
7.5	IF statements	44
7.6	GOTO statements	44
7.7	Intrinsic Functions	44
7.8	Diagnostics	44
7.9	Subroutine usage	44

8.0	General Practices.....	45
8.1	File manipulation.....	45
8.2	Use of the no-op feature in the OAN,IAN instructions.....	47
8.3	Writing into ECS (or reading from ECS).....	49
8.4	Channels and Interlocks.....	49
8.5	Files.....	49
8.6	Recovery.....	49
8.7	Memory usage.....	49
8.8	Documentation.....	50
9.0	Summary.....	50

General Guide-lines

There are many guide-lines for design and coding that should be followed regardless of the language being used. The particular implementation of these guide-lines may differ among the languages, but the intent spelled out here must be observed.

If you feel that any of these conventions seriously adversely affect your project, talk it over with your project leader (or team member) and the systems supervisor; exceptions can be made.

All quotes are from E. Yourdon's book, Techniques of Program Structure and Design, the first six chapters of which you are expected to read and know thoroughly.

1.0 Comments within a Program

Page ejects, titles, spacing, etc. should be used to visually aid the reader of your code.

Abbreviations should only be used when the meaning is clear. For example. "RBT" for Record Block Table is acceptable in a routine dealing with this table; however RBT for Retry Block Test would not be acceptable in the same routine. "O.R." is the only acceptable abbreviation for output register (do not use "OR"); "NUM" is the only acceptable abbreviation for number (do not use "NO.").

It is always better to avoid abbreviations for names (spell out "number") and multiple abbreviations for the same name must never be used.

In line in code comments should be terse, but still give the program the information needed.

At all times an effort should be made to provide meaningful comments that will be read and understood by other programmers. Cute, coarse, derogatory comments may be funny the first time, but they inhibit easy comprehension of the code and must be avoided. Furthermore, all comments should be spelled correctly and not be chopped off by sequencing. Minor spelling mistakes are acceptable as long as they are not frequent and do not impair the comprehension of the code.

1.1 Global comments:

At the beginning of the main module of a routine there should be a complete description of the routine. This would include concise external specifications, internal flow, functions of the major modules, locations of descriptions of major tables, description of files used, any debugging aids, and how to compile, load, and execute the routine. Also, the program library on which it resides should be told. The compile, load, etc. information should be near the very beginning of the routine.

This description must be visually presented in a neat, easy to follow manner. It should be expected that programmers would read this section to orient themselves before making any modifications.

1.2 Within the Code

1.2.1 Subroutines (or other logically complete sections of code)

Subroutines must be documented at the beginning with (in this order):

- purpose.
- what high level language other than the one being used to write this can call it (FTN, COBOL, SYMPL) if none, this line can be omitted.
- entry conditions ('A' register, direct cells, high memory pointers, registers set, etc.).
- parameter meanings and possible values (that are entry conditions).
- exit conditions.
- parameters that are exit conditions
- special exit conditions.
- direct cells, registers, or other data areas altered. (Registers used are not given if the routine is written in a high level language.)
- error handling.
- subroutines called. (and, on the same line, what the subroutine will do)
- special conditions or cautions (particularly for the modifier of this routine).
- narrative of how it is going to accomplish its task. The narrative must be in outline or structured english style with proper indentation. If it is truly straightforward a statement to that effect may be made.

Again, this narrative should be concise. Since it should be describing a small module, if you have to say too much then you have probably included too much in the code.

These comments must be presented in visually neat fashion. The entry/exit statements must completely specify the requirements of the routine. That is, this subroutine could be replaced by a 'black box'¹¹ that would function correctly if the specifications in this description were followed.

For example (a Compass subroutine):

```
SUBROUTINE READUR -          ROUTINE TO READ A UNIT RECORD.
*   CALLABLE BY:            FTN
*   ENTRY:-                 CALL READUR (LFN,ARRAY)
*   WHERE:
*
*                           LFN   = THE LOGICAL FILE NAME.
*                           ARRAY = THE ARRAY WHICH WILL
*                               RECEIVE THE DATA
*
*   EXIT:                   ARRAY IS FILLED WITH THE NEXT CARD
*                           IN 80R1 FORMAT.
*                           ARRAY(1) HAS -1 IF EOF OR
*                               -2 IF EOI FOUND.
*
*   USES:                   ALL BUT AO
*
*   CALLS:                  CPC     TO MAKE THE READ REQUEST.
*                           OPENFL TO OPEN THE FILE
*
*   METHOD:
*
*   IF FILE NOT OPEN THEN OPENFL(LFN);
*
*
*   WHILE...»•
```

An alternative ENTRY/EXIT description is

```
*   CALLING SEQUENCE:      CALL READUR(LFN,ARRAY)
*
*   ENTRY:                 LFN HAS THE LOGICAL FILE NAME LJZF
*
*   EXIT:                  ARRAY HAS THE DATA READ IN 80R1 FORMAT
*                           ARRAY(1) HAS -1 IF AN EOF WAS READ;
*                               -2 IF AN EOI.
```

If ARRAY were something which was set before entry and modified by the subroutine, it would appear in both the ENTRY and EXIT comments.

It is very important that any modification which causes this documentation to be incorrect also include a change to the documentation.

1.2.2 MACROS

Macros must include the same kind of general comments that are specified for subroutines. Specifically, every parameter must be described. The general purpose of the macro must be stated. The registers, direct cells, data areas, (or what ever is relavent for the language) used must be stated. For non-trivial macros, the internal symbols, algorithm, etc. must be described. The entire macro description must be done in a visually clear fashion where, for example, the meaning of each parameter is readily apparent.

1.2.3 Tables

All tables must be documented with:

what the table is, the meaning of each field in the table, and how to add a new entry to the table (including all things which must be altered when a new table entry is added).

Adding a new entry to a table and causing the proper routines to function with this new entry must be as simple as possible. It should not require modifications in many places in the code.

1.2.4 Narrative

Each small section of code should contain a narrative header. This narrative should be a further refinement of the general narrative placed at the beginning of the routine. The narrative, unless it is repetitive, should also mention what the vital data areas (registers, direct cells, etc.) contain that will be used or must be preserved over the next section.

This narrative must be visually separated from the code by either blank lines or a line with only the comment indication character preceding and following the narrative.

1.2.5 Correction History

If this is a modification to a routine, it must include an entry in the correction history.

The format for these comments is in STRAP 3.

It is important that the justification for the modification be thoroughly explained in the correction history section. If this is a fix to a problem, the correction history must specifically state what the problem was and what was done to fix the problem. Simply "fix bug in 1AJ" or "convert routine

to SCOPE/HUSTLER" is not appropriate. However, it should not, usually, detail specifically what was done - leave this for the narrative to follow. That is, "Add processing of the RG parameter on the job card for rate group determination" would be acceptable.

If the- correction history section does not exist, the modification must include it under a separate identifier. This identifier would be "CHnnnnnn"; The format for creating the correction history section is in STRAP 3.

1.3 UPDATE comments:

Most UPDATE comments are to facilitate the preparation of the LSD document. These comments must be well written, in good English, and they must be easy to understand. See STRAP 3 on modification deck comments.

UPDATE comments (not intended for the LSD) can be used to explain the reason for the modification when this reason would not be appropriate to appear in the source listings.

1.4 DOCK

While it is not required to set up your comments so that DOCK can extract them, you are strongly urged to consider this. In any routine that does use DOCK, all modifications must continue that practice.

2.0 Design and Coding Practices

2.1 Structure

All routines are expected to conform to structured programming, top down development rules. "...Structured programming is a philosophy of writing programs according to a set of rigid rules in order to decrease testing problems, increase productivity and increase the readability of the resulting program."

Before coding of the routine begins, several structured design charts of the routine must be created. These designs should be evaluated and the best one chosen. The book, Structured Design, by E. Yourdon and L. Constantine describes how to create and evaluate a design. If this procedure is followed, then your design will be modular, simple, and straightforward. This design chart must be reviewed by your team member or project leader. You are encouraged to have several design walkthroughs for any major project.

The resulting program should be simple and straightforward. For example, you must avoid programming that branches back on itself, has multiple exit points or multiple return points. (Only one entry-one exit per routine).

For example, this structure would be unacceptable:

```

                IF(X EQ. 0) GO TO 20
                A=7
10             CONTINUE
                .
                .
                .
                GO TO 99
20             A=23
                GO TO 10

```

The only time that a convoluted or "jumping into the middle" code can even be considered is when memory or time is extremely critical. This is unlikely to be truly the case in any situation.

"More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason—including blind stupidity. One of these sins is the construction of a "rat's nest" of control flow which exploits a few common construction sequences. This is precisely the form of programming that must be eliminated if we are ever to build correct, understandable and modifiable systems."

The "IF...THEN...ELSE" structure can be used to avoid multiple returns in a function:

```

LOGICAL FUNCTION ANSYES(INPT)
CHARACTER INPT*(*)
IF (INPT .EQ. 'YES') THEN
  ANSYES = .TRUE.
ELSE
  ANSYES = .FALSE.
ENDIF
RETURN

```

Indentation of higher level languages to show structure and levels should be done whenever possible.

To promote and insure modularity, each module should be a separate subroutine and have one entry and one exit point. Usually, the routine should return any error condition to its caller. When the routines are under one identifier in COMPASS, then the QUAL pseudo op should be used to insure separation. See the COMPASS section for further details.

2.2 User messages and diagnostics

User messages must be polite, concise, jargon free sentences. Abbreviations must be avoided whenever possible. Cute remarks are not allowed (i.e., "Sorry, Charlie"). As much information as possible must be given to help the user find his problem. For example, do not give the messages "CONTROL CARD PARAMETER ERROR", "IN .LT. FIRST", or "FL EXCEEDS MFL"; instead use messages such as "UNRECOGNIZED PARAMETER: LO", " IN (0354) IS LESS THAN FIRST (1354)", or "MEMORY REQUEST FOR 120000 EXCEEDS THE 100000 MAXIMUM".

Existing error processing and message delivery routines, such as D00 and 6WM, must be used whenever possible.

2.3 Error detection, analysis, and processing

When testing user or operator input you must check for all legal values and issue a diagnostic if the input fails these tests. For example, you must not assume that if the input is not "YES" it is "NO" or if input starts with an "N" it means "NO".

Similarly, you must insure that only legal input is given as parameters to MACROS.

Output to users must be checked for reasonableness. For example, it is doubtful that one could use 262142 CPU seconds in one job; consequently, any routine that tells the user how many CPU seconds were used could test for a reasonable value. This is especially important for any accounting or authorization file deductions.

When adding entries, values, or codes to a CDC defined table, diagnostic list, etc., you must either use the values that have been allocated for installations by CDC or you must leave a sizable **area**

into which CDC can expand. A typical method is to use negative numbers when CDC is using positive numbers. Also, when this is to be done, tell the Systems Supervisor so that an installation reservation can be obtained from CDC.

This may cause some difficulty since CDC code tends to have many hidden assumptions, such as the number will be positive, etc. However, we must avoid the incompatibilities that will occur when CDC expands their usage. Consequently, the extra hassle is worth the time spent.

Whenever reasonable (and it is usually reasonable) you must include checking on internally generated data. This includes tests such as insuring a valid push/pop of a stack, insuring a value in a jump vector is good, an index is within the dimension allocated, an argument has a valid range, or a linked list is properly formatted.

This internal error checking is best started during the design phase of the project. For example, a linked list could be designed so the modules could easily tell whether an entry had been misiinked or the links clobbered. Furthermore, this design analysis would also determine whether the fragile linking process was truly necessary!

At some module levels it is not practical to do internal checking—you must depend upon earlier checking. However, you must design with the idea of always testing and eliminate those tests that consume too much resource.

For example a routine that packs 'n' characters into one word should check that 'n' is not bigger than the number of characters that will fit in a word. Also, the termination value on a DO loop could be checked for being within an array if the loop variable is used to index into the array. However, it is not practical to check every reference to the array for being within bounds.

No code should modify itself unless there is no other way to accomplish the task. With PP's because of the restricted memory space and registers this does become necessary. However, for CP and Interdata code this is seldom (if ever) absolutely necessary and will be permitted only on a case-by-case "prior approval by the project leader" basis.

Counted loops (like DO loops) that are supposed to exit before the count is exhausted must include code for the case when the count is exhausted.

No infinite loops should be written. All loops should have a finite termination within a reasonable time. In a few cases, a loop may depend upon some outside action to terminate it (the error flag being set, for example). It is better to abort the job or the system than loop forever. A system crash is preferred to an infinite loop because users are notified more quickly about a problem and it is easier for the crash analyzer to determine what

happened. Also if we ever develop an automatic restart process, it will probably be initially done based on the occurrence of a crash.

All entries to error routines or system crashers must be such that a dump can show the precise caller and reason. For example, FORTRAN code with a number of "GO TO 999" statements where 999 is the error section is not acceptable unless there is some error index that will uniquely specify where the error occurred.

2.4 Copyright

New routines must contain, at the beginning of the first program, a copyright statement. This statement must read: "COPYRIGHT year, MICHIGAN STATE UNIVERSITY, BOARD OF TRUSTEES." where year is the year when coding started. This should be among the first lines you write when starting to code a new routine.

If a routine is extensively revised, it must include the copyright statement. If the routine already has a copyright statement by MSU, then the new year should be added to the year list (do not remove the original year).

2.5 UPDATE common decks

UPDATE common decks used between program libraries should be placed on the COMDECK program library. The creation of general utility routines is encouraged. These should also be placed on the COMDECK PL. Routines placed on this program library must conform to the conventions stated there (and in STRAP 9) when applicable. For example: the DOCK comment style must be preserved, the register usage must be followed, etc.

It is expected that project leaders will take the extra time to cause common decks to be created when they recognize a general utility appearing in their group's code.

3.0 Symbols and Names (including routines, variables, macros, etc.)

3.1 General naming rules

The appropriate choice of names allows for easier comprehension of the code. However, you must be very careful about the names you choose; otherwise, you can create confusion, increase your debugging time, and create maintenance problems for those that follow.

Examples:

1AR and IAR are a very poor choice of names because it is easy to confuse the "1" with an "I". Also, "ONE" and "0NE" would be easily misread if the "oh" were not slashed.

A branching macro named "NUMTEST" is to test whether a character is a number. You might expect it to branch if character was a number, but you are not sure and could easily be wrong! A better name which would not be ambiguous should have been chosen.

Also suppose two flags were defined: DATAFLG and ERRFLG. DATAFLG has zero if there is data present; ERRFLG is non-zero on an error. The testing of these flags would soon become confusing because of their opposite meanings on a zero test.

Consider, also, the symbols "FETP" and "FNTP" which can be easily read as FET Present, and FNT Present. However, the bit is zero if the FET (or FNT) is present. The sense of both bits were kept the same, fortunately, but the symbol name chosen was backwards since it is usual to associate the set (or 1, or non-zero) condition with true.

The previous example also illustrates another problem that must be avoided: There is only one character difference between the two symbols. A mistake that replaced the "N" with an "E" will create a bug that may be difficult to find.

The eye has a tendency to see what it expects to see. What it expects to see is influenced by the surroundings. Consequently, symbols must be chosen to maximize their differences.

To minimize the problems illustrated above the following rules are to be followed when creating symbols:

- a) Meaningful mnemonic symbols must be used; words should be spelled out (correctly) where possible.
- b) Symbols must differ in at least two characters, preferably three; one of the differing characters must be the last character (if there is only a difference of two).

For Example:

Unacceptable pairs

W.FETXYZ

W.FNTXYZ

ABCX

ABCY

LETCRUP

GETCRUP

Acceptable pairs

W.FETXYZ

W.FXYZ

ACBX

ABYC

ABCDEFX

HIJKLMX

- c) If it looks like a text symbol (see below) then it must fulfill the meaning for that style. For example: W.xxx must be a word offset in a table; V.xxx must be a mask based on starting (right-most) bit and length; Y.xxx is not used alone as a shift count, etc.
- d) Symbols that name flags or test conditions must be named and tested so that a true condition is a "1" (or non-zero) and a false condition is a "0" (or zero). That is, if a bit is zero when a file is busy and one if it is not, then it should be called a not busy flag, (or bit), or complete bit, or some similar name.
- e) Macro names that test and jump or test and set a condition code should be named to indicate when they jump (and that they jump) or when the condition code is set to a one. For example, "IFNUM" or "MTRDIEIF" not "NUMTEST" or "KILLMTR".
- f) Because of the problems with 63 vs. 64 character set, the use of colons in symbol names must be avoided.

3.2 Symbol usage

Any program in a high level language that does not have a text facility (like FORTRAN) must have a COMPASS routine that establishes system text symbol values in COMMON blocks for the high level routine to use. Or, alternately, the high level routine can call upon a COMPASS program to extract the value.

A set of similar symbol definitions must be grouped together. For example, all symbols for direct memory, all constants for accessing a specific table, etc. Also, symbols used throughout a program (or those that might be) must be defined at the start of the routine. Symbols used only in one place, such as to define a table, should be placed just before they are used.

Only one symbol name should normally be used for a location, or variable (this does not include the D.TW, etc. names for PP direct memory). Therefore, EQUIVALENCE in FORTRAN or similar assignments in COMPASS must not be done unless there is a valid reason. Since space is least available in PP memory, most of the valid reasons will occur there. See the COMPASS section for further restrictions.

4.0 UPDATE Conventions

4.1 Deck Resequencing

Resequencing must not be done unless there is nearly a total rewrite of the code. It is also possible to resequence a deck if the deck name must be changed and there is no one else working on changes to the routine (including CDC). Resequencing should be avoided since it causes the identifiers in the listing and the correction history to be lost.

4.2 UPDATE Abbreviations

Any UPDATE abbreviation should not depend upon a previous update control card. For example:

```
*D      CIO.359,370
```

is fine. However:

```
*D      359,370
```

is not allowed because it depends upon a previous control card to set the deck name. This is done to prevent possible errors when someone (such as the originator) slightly modifies a correction deck.

Update abbreviations for the directives are allowed.

4.3 Correction Deck Structure

The correction identifiers for UPDATE should include your initials for the first three characters and then some mnemonic for the correction. For example:

```
•IDENT,VMSADF
```

(See STRAP 3 on modification decks for further details and examples.)

However, the texts have unique identifiers starting with a slash followed by the deck name. This structure should be preserved as much as possible. Consequently, the modifier's initials should be appended to the end of the standard identifier; only the last characters of the deck's name should be deleted when necessary to create a unique identifier. For example:

```
*IDENT /CPAVMSST
```

This would be a correction identifier for the Control Point Area text deck written by a person with the initials VMS; the characters ST might have some meaning about the modification.

*IDENT /MTRCOJSF

This would be a correction identifier for the MTR communication tables (/MTRCOM); the 'M*' is deleted in order to fit in the modifier's initials.

The correction deck must end with the proper "*COMPILE" control card for that deck.

One IDENT (or set of IDENTs) should be used for one modification or correction. For example, the ident JSFRATS could be used for the modification to install the rapid access service. However, if this was too extensive or extended over several PL's, then JSFRATS1, JSFRATS2, etc. could be used.

4.4 Common decks

The use of common decks is encouraged. This use is especially important for COMMON storage declarations, common procedures, etc. However, the use of the common deck must be well documented as to its beginning and end as well as its purpose.

Calls to common decks must have cards preceding and following them such as:

```
*      COMMON DECK ABC USED AT *+1
*CALL ABC
*      END OF COMMON DECK ABC AT *-1
```

4.5 *YANK Identifiers

If a modification is being removed with a YANK, all code under the identifier used for the YANK should relate only to YANK. That is, if at some later time we wish to undo the YANK by YANKing the YANK ident, we should not also lose unrelated code.

Frequently this means that the YANK is done under a separate identifier; however, if the YANK results in unusable code, then the yank identifier should also include the necessary code to make it usable, [usable does not mean bug free].

4.6 Miscellaneous

Please refer to STRAP 3 for additional UPDATE restrictions based on installation considerations.

5.0 COMPASS Conventions

Compass always takes longer to correctly code, debug, and maintain than higher level languages. However, it does currently offer the possible advantages of smaller code, faster executions and flexibility. Do NOT use Compass if these are not needed. Use some higher level language (if available).

5.1 General Guidelines

5.1.1 Be clear

All COMPASS code must be straightforward and simple. Fancy tricks to save a few microseconds are very, very seldom worth the time to debug and maintain.

For example:

1X4	X7-X3				
AX4	59		MIN	WHEN	x3/x4,lt,x7
BX7	X4*X7	VS.		BX7	X3
BX3	-X4*X3		MIN	DONE	
BX7	X7-X3				

Both of these will end with X7 having the smaller of X7 and X3. While the code on the left is faster on a CYBER 175 and it might take less memory, it is definitely not worth the cost of confusion, maintenance and debugging except in very rare circumstances.

Furthermore, tricks to avoid NOP's such as SX3 A3+0 must not be done. Again, it is not worth the maintenance cost (a future modification might alter the instruction placement).

There exist some obscure algorithms on the COMDECK PL. This is the only place for this type of code since the maintenance and debugging cost can be spread over a number of projects.

Before the introductory comments on a routine or module, place a SPACE card with sufficient line count to insure that the comments are all on one page (or use a TITLE card). Also, the last introductory comment line must be separated from the code.

When a table is created that uses the ordinal of the table to obtain an entry, then the LOC pseudo op should be used to show the table ordinal in the listing. Always set "*" back at the end of the table (e.g., LOC *0).

When modifying CDC code for installation into our system, there may be times when CDC implicitly assumes a table structure. For example, there may be a "LDD D.Z1+3" to

obtain the number of the current word pair in the FNT. The best method of installing this CDC code would be to change the instruction to "LOADFLD D.Z1+C.FRBORD,FRBORD"; however, this may be impractical because of the number of such changes. An alternative is to place ASSUME's at the beginning of the program which would document the dependencies.

5.1.2 Be careful

There have been a number of problems recently when coders forgot that a value (such as a line count) could become greater than $2^{17}-1$. When this happens the set instructions do a sign extend (or truncate the value).

When doing an indexed jump (i.e., JP B2) you must insure, whenever possible, that the jump is within range. You should also leave some trace as to the last value used on the jump. That is, store the register before the jump. This will help the person analyzing the crash know whether the jump value was good.

A similar problem can occur with linked lists. It is quite possible for some bug to cause a PP to alter a link or for another part of your program to incorrectly alter a link. Consequently, you must include validation tests when you move along a list or use values from a list.

For example, you might design a linked list with forward and backward pointers. Then, as you move forward along this list, you should check to insure the backward pointer points to the location you came from. Another possibility is to check a known value in the data (supply redundancy). The verification of the PF flag in the RBT, for a file whose FNT says it is a PF, is an example of redundancy checking.

If the CPMCTX macros are available, a routine written in Compass must have entry registers specified by the EREG Macro, return registers specified by RREG, registers used specified by UREG, and routines called specified by CALLS.

Macros must be used to generate tables so that a new table entry simply consists of a simple macro statement. The only exception is when a table entry can be created by a CON or DATA pseudo op. (There is a small subset of tables for which this is not true, but they will be handled on a case-by-case basis).

Always explicitly state whether a byte or a constant is to be used as a test value. Frequently, one will forget to specify a byte, and the default constant assumption will be wrong.

5.2 General in-code comments

Accurate line-by-line comments must be placed there when the code is written, not after debugging is finished. The comment on an instruction line should refer only to that instruction, do not continue the comment onto the next instruction line. In case a higher priority project interrupts your work, then it will be possible for you or someone else to pick it up several months later. (If you don't know what you are doing, you have not spent enough time designing. Go back and complete the design before you begin to code.)

NOTE: This does not mean that every line must be commented. In fact, every line must not be commented. If good global narrative comments are provided and modular coding is followed then in-line comments will be much less.

As a guide-line:

1. All jumps and tests must be meaningfully and concisely explained. However, it is not necessary to explain the computer instruction. That is, for "NJN ALPHA" the comment could be "JUMP IF FILE NOT BUSY"; you do not need to say "IF FILE NOT BUSY THEN GO TO ALPHA" or "IF NONZERO THEN FILE IS NOT BUSY GO TO ALPHA."
2. All storage locations must comment on what is stored there.
3. All symbols assigned values through EQU'S or SET'S must be commented.
4. Except for trivial cases in higher level languages all the variables (except temporaries and loop counters) must be commented on at the beginning of the program.

5.3 Global Restrictions

The "*" symbol or similar self-reference symbols must not be used. This means that code such as:

```

ZJN  *+3      or  HH  SA1  BI
LJM  NOTHERE      NZ  XI,*

```

must not be done. Similarly 'IF' tests, 'ECHO*' ranges, 'DUP' ranges, etc. must have labels on them. They must not be bracketed with unnamed termination or have a counted range. The only exception to this prohibition is for spacing in a BSS, for example: "BSS D.PPONE-*."

RMT code must have a name and be assembled by the named HERE request. The ending RMT should also be labeled and specified as the end.

When making system requests you must use a macro if one exists. For example, the CPSTAT macro must be used instead of making an "SYS 3" RA+1 (or SYS=) call. Note that a macro exists if it is on any text. You can not avoid the macro because it does not exist on the text you are using.

For all new routines (or major rewrites of old routines) the location field must start in column 1; the operation field must start in column 11; the address field must start in column 20; and the comments field must start in column 36. These are the standard columns. However, any modification to a routine with a clearly defined convention other than 1, 11, 20, and 36 must follow that routine's convention. If the entire routine is greatly jumbled, then the standard convention should be followed even if three or four lines around the modification are consistent. Exception: The operation field for a structured programming macro (WHEN, ORWHEN, ORELSE, DONE, LOOP, WHILE, etc.) should begin in column 9 for the first level and column 10 for the second level. This will allow a small amount of indentation to visually bracket the code.

Display coded data must never be given in octal. For example, "ZRO EQU 2R00" not "ZRO EQU 3333B".

Numeric data, whenever possible, should be given in its natural format. If it is not possible, a comment must describe its natural form. (Natural form means, for example, using 1.35 not its floating point octal equivalent.)

Expressions should be used when they will clarify an operation. For example, "LX2 24-2" would be used instead of "LX2 22" if the shift would have been 24, but the register was already shifted by 2. Of course, symbols would be even better. ("LX2 59-Q.BIT1+Q.BIT2").

No address constants can be used. That is, the "STM 1357", "LDD 25", or "SA1 1" type of instructions are not allowed.

There may be no Compass warning error diagnostics in the routines being installed. This means that any modification must include removing warning errors even if that modification did not cause them to occur.

Loop tests must terminate with the most inclusive test possible. For example, on a positive, counting down to zero, loop where the test "NZ B3,LOOP" could be used to execute the loop; "GT B3,B0,LOOP" would be better because it will terminate when B3 is minus—even though this could "never" happen.

5.4 Symbol definition and usage

Symbols must be used to describe and to reference the length, width, byte position, bit position, etc. of each entry in all tables that have more than one entry per word. This symbol usage must be such that if an entry within a table is moved, a reassembly of the code will cause the reference to be correct.

Occasionally, significant time and development can be saved by implicitly assuming the position and size of an entry. When this is done, the ASSUME macro must be used to mark this assumption.

However, it is preferred to not make any assumptions.

Symbols must be used in instructions or table entries to convey meaning and provide cross reference information even when they are not expected to change.

For example:

```
EQPSTAT EQU 1300B equipment status
NEWPAGE EQU 1RT top of form
```

Neither the equipment status nor the forms control for top of form are likely to change. However, by using symbols it is much easier to find all references to these constants. Also, experience has shown that some "unchanging" symbols do, indeed, change! Note that this does not mean the symbol for the top of form should be used in a DIS pseudo op. However, a micro could be defined and used in the DIS. (It is expected that micros will someday be cross-referenced; also it is much easier to change a single micro definition.)

When symbols are defined, they must be defined in terms normal for that symbol.

For example:

```
Y.CMADDR EQU 17 NEWPAGE EQU 1RT
      not      or      not
Y.CMADDR EQU 2IB NEWPAGE EQU 24B
```

Also, a comment explaining the symbol must be given.

5.5 Scope text symbols

Scope text symbols help clarify the code, improve flexibility, and provide the means to obtain a global cross reference by symbol and routine throughout the system. Table additions to the text must use the FIELD and WORD macros. Mew tables, etc., must follow the current conventions in this STRAP. (See below)

Most text symbols have a prefix code followed by a period (SC.DSTF, for example). These prefix codes have a specific meaning. Symbol names with prefixes that conflict with these meanings must not be used:

5.5.1 Symbol conventions to reference table entries (bit positions are numbered right to left, 0 to n).

- W. - The word address
- C. - The byte address within the word (12 bit bytes). Bytes are numbered left to right, 0 to 4. If a field crosses a byte boundary, the value of the C.xxxxx symbol is the lowest numbered byte.
- S. - The bit position within the byte
- Q. - The lowest (rightmost) numbered bit position within a 60 bit word for that field.
- Y. - The width of the value. This should be used (along with the DECMIC pseudo op) to determine the maximum value for the entry.
- V. - A mask that will isolate the value (based on position and width) relative to a byte.
- P. - The actual absolute address (or unsealed absolute address) that points to a table
- SC. - The scale factor for addresses that need to be scaled
- T. - An absolute table address.
- LE. - The length (in CM words) of each entry in a table
- L. - The total length of a table
(Note that in some cases L. is used instead of LE.; however, future use must be as described above.)

5.5.2 Other text symbol conventions:

- AB. - Job card errors (obsolete)
- CC. - Connect codes
- CE. - Function codes for M.ICE monitor requests
- CF. - CP.LISP function codes

CH. - Pseudo channel numbers
 CX. - Corridor requests for CPU monitor
 DT. - Device type codes
 EC. - ECS access codes
 FDB. - Parameter codes for the FDB macro
 F. - Flag values (i.e. F.ERxxx and F.SSxxx)
 I. - Instrumentation Table reference
 IL. - CPU monitor interlock table offset
 10. - CIO function codes
 IP. - Installation parameters
 IT. - IIT function codes
 M. - Monitor requests made by PP's
 N. - Counts and numbers of things
 OV. - Display code value of PP name
 O. - Stack processor order codes
 PF. - Permanent file manager function codes
 PH. - Phase symbols
 RA. - Obsolete symbols to refer to the lower part of
 a CP program's memory
 R. - PP resident routines or symbols
 SR. - Source codes; input/output destinations
 SS. - Swap states
 TP. - ITP function codes
 WS. - Wait states for CPUMTR EXEC tasks
 WT. - Wait states Hustler jobs
 XJ. - Exchange package words
 XT. - Comdeck name

5.5.3 Table prefixes in symbol table

Characters to the right of the dot are frequently used to identify the table to which a symbol refers. Any symbol added to an existing table must use the prefix defined for that table. A unique prefix should be chosen for any new table added to the system. The following is a partial list of current prefixes; if you plan to add another prefix, you should first look thru the 'U.' symbols in the cross reference listing of a text to insure that the prefix you pick is unique.

- APF - APF Table
- .CP - Control point area
- .CS - CPSTAT return block
- .DCT - Disk controller table
- .DC - Disk controller table entries
- .DFB - Dayfile buffers in CMR
- .DF - Dayfile monitor requests
- .DIR - Library tables
- .DLL - Disk label protection

- .DST - Device Status Table (a device is an RMS disk)
- EC - ECS partition numbers
- EF - Code value for ECS FNT's
- .EP - ECS partition word
- ER - Error codes
- .F - FNT entries
- .FD - FEDATA return block
- .FDB - Permanent file definition block
- FET - FET entries
- .FNS - FNTSTAT return block
- H - Frequently for Hustler pool table pocket entry
- .IN - Installation area
- JS - Job scheduling
- .LB - Disk label pointers
- MC - Monitor Communication (between CPU and PP)
- .PFC - RBTC entry
- .PFD - PFD entry
- .PFI - Installation area within RBTC
- .PP - PP communication area
- .RA - Words 0-77 of control point FL
- .RB - RBT bytes
- .RBR - RBR table
- .RBT - RBT entry
- .RWPP- PP disk I/O communication word
- .ST - Stack request
- .SV - Miscellaneous save area in ECS pool entry
- .T - Tapes table entry
- .US - User table

5.6 General pseudo-op usage

5.6.1 DATA and CON usage

The DATA or CON pseudo ops should normally be used only for constant data (data that is not altered during the execution of the program). It is usually better to always put values in changing locations rather than depend upon preset load. If all values are set during execution, then the routine can be more easily be made serially re-entrant. Specifically "DATA 0" must not be used if some value is to be stored there later. "BSSZ 1" should be used instead.

If DATA item must be altered during execution, then it must be commented as to the various values it can assume and where and why the setting takes place.

There should only be one piece of data for each DATA or CON pseudo op.

5.6.2 BASE and CODE usage

The BASE or CODE pseudo ops must not be used unless truly necessary. When used, they must be heavily commented and used only in a brief area. The normal BASE and CODE are always assumed (decimal base and display code for CDC computers).

5.6.3 COMMENT usage Any program that is placed on the dead start tape (or other library tapes) must include a COMMENT card giving a short description of the program. This will make any ITEMIZE of the tape more meaningful. This COMMENT should precede any copyright comments (including the one in the SST macro).

5.6.4 VFD usage

The VFD pseudo op should not fill more than one normal word. That is, the maximum for one VFD in PP code would be 12 bits; however, if this were setting up a word to be written to CM, then it could be 60 bits.

The VFD pseudo op must not be used where a CON or DATA pseudo op can be used.

5.6.5 USE, ORG and other block counter operations

Pseudo ops that alter the block counter, such as USE and ORG should be used carefully. When they occur outside of macros or common decks there should be a reset to the base block; inside there should be a reset to the previous block.

These pseudo ops should normally be used to set up data areas at the beginning of a routine or to allow buffers to overwrite initializing code. They must not be used to flip between code blocks several times within a page. REQ is a current example of what not to do.

The main concern here (as elsewhere) is to prevent tricky, obtuse, complex code.

5.6.6 LIST options

Compass list options should be used to display the contents of any macro generated table entries or tables now being assembled with a HERE pseudo op. Usually a "LIST G" or "LIST D" is appropriate. Also, see section 5.8.1.

5.6.7 QUAL pseudo-op

The QUAL pseudo op must be used with care. Properly used it can ease maintenance, prevent bugs, and insure modular code. Improper use can just create more confusion.

QUAL must be used on higher level modules (or subroutines) within a program or routine. This will insure that there are no entries or references within a module that are not explicitly stated. Only the entry point would be a global symbol. Small subroutines should be grouped with one qualifier under their parent module.

A qualifier before a symbol "/SUB1/LOOP" must almost never be used (nor can a sequence of QUAL's to accomplish the same thing be used). That is, global symbols must be globally defined (preferably at the beginning of the program - subroutine entry points, of course, cannot be). There exist some CDC routines where this prohibition cannot be followed.

The macro's SUBRT and ENDSUB must be used (when available) to specify the start and end respectively, of a subroutine. These will cause the appropriate QUAL's to be generated.

Some examples of where qualifiers should be used:

- * To separate individual functions grouped under one COMPASS IDENT (see SYS and PFU). In this case all subroutines local to a given function would be qualified under that function; subroutines available to all functions would have globally qualified entry points.
- * To separate initializing code which will be overlaid from the non overlaid code. In this case there must be no reference to any code within the overlaid section from outside (except for the entry point).

In summary QUAL's are required to help insure modularity by helping to prevent convoluted code and insuring that improper references are caught early in the debugging of a modification.

The COMPASS assembler program is an example of how QUALs must not be used. This is because each few lines of code has a QUAL PASS1 and then QUAL PASS2. This adds to the already difficult process of understanding what is going on.

5.7 Macro definition (including OPDEF, CPOP, PPOP, etc.)

5.7.1 General

The intelligent definition and use of macros is encouraged. Macros should improve readability, reliability, and flexibility of your code. Macros used throughout a program or subroutine must be defined at the beginning of the routine. If a macro is used in only one place (defining a table for example), it must be defined just prior to its use.

The sections on comments, symbol names and general coding practices apply to macro definitions. (Both the macro names as well as their parameters). For example; the parameters must be completely verified; "NO" should not be assumed if not "YES". Condition parameters must be "ZR", "MI", "PL", "NZ" not "Z", "M", "P", "N" respectively. Value relationships must be EQ for equal, LT for less than, GT for greater than, GE for greater than or equal to, and LE for less than or equal to.

When a macro is placed in a text, it must be preceded by the appropriate LAB.MAC call. This will keep the cross reference map up-to-date. If your program has many macros defined, the use of LAB.MAC is encouraged (required if you have very many).

The use of MACROE (with meaningful parameters) instead of MACRO is encouraged. For example:

```
"TABLESET RD=1, OP=1, WR=0, JUMPOPENRD"
is much clearer than
"TABLESET          1,1,0,OPENRD".
```

When you define a macro you must not be inconsistent with existing general practice and text macros unless the practice is being phased out (probably it will be counter to STRAP 9). For example: testing and jumping macros give the jump address in the address field; consequently you must not define a macro that will have the jump address in the location field.

When defining a macro, especially for a table entry, try to get the actual code generation into one compass statement. This will reduce the number of lines generated for the macro call. If this is not practical, a "LIST -J" can precede the table to reduce the size of the listing (put a "LIST *" after the table).

5.7.2 Things you must not do with macros:

The function of any existing op code, pseudo op, or text macro should not be redefined. PP channel instructions are an exception. Other exceptions may be allowed only if they do not significantly alter the external original process and your project leader approves in advance .

Within the macro definition, parameter names must be used explicitly. ";A" type of parameter references must not be used.

Macros, as a general rule, should only use registers named as parameters (unless they call routines). However, it is permissible at the beginning of the routine to declare a register "off limits" and then use it within the macros. WARNING: CPMACTX operations will frequently use X5 as a scratch register if no scratch register is specified.

5.8 MACRO usage

Modifiers of code must, at least, follow the existing code's use of macros. That is, if the original code uses a macro, then you should follow that usage.

If you notice a repeated sequence that would be improved by a macro, then a new macro can be defined and used. In general, you should alter every occurrence of that sequence to use the new macro. However, considerations such as check out time, the extent of the original modification, etc. may alter this decision.

5.9 CDC Central Processor Coding

When writing in COMPASS for the CPU these guidelines should be followed:

1. When the CPMAC User's Guide is available, then the structured programming macros, entry/exit macros, description macros, etc. should be used. EXCEPTION: The WHEN/DONE, LOOP/REPEAT, etc. blocks should not span too many lines of code.
2. Unless there is a clear, compelling reason all subroutines must "be entered with a return jump and exited through the entry. The technique of using a B register to hold the return address and leaving the subroutine by a "JP Bn", for example, must not be done.
3. The use of multiple entry subroutines must be closely examined and clearly justified. Since this leads to many confusing complexities (most of which are prohibited), it rarely can be correctly used.

4. If it is better to reference a location by using the contents of another register, you must still include a cross reference entry. Usually you can avoid this by using the "=reg" pseudo operation. You need only one reference for all code in the immediate vicinity.
5. The naming of registers in CP code must be avoided since it frequently confuses the person modifying or debugging the code.

There may be exceptions to this prohibition, but they will be handled on a case-by-case basis.

6. The instruction must always appear by itself in the operation field; the unpack and normalize instructions are the only exceptions since two registers receive data. For example:

NZ,X1	ALPHA
PX6,B7	X5

are not permitted but

UX5,B7	X6
NX1,B7	X6

are allowed.

7. Entry points to subroutines should be such that an immediate error is given when the entry point is jumped to without an RJ having been done. This error must show where the problem occurs. This can be accomplished by a PS, a "BSSZ 1", or a "JP 40000B+*". This does not apply to routines that have no mode error processing available. (IRCP for example.)

The "PS" instruction would seem to be the most useful since it will reset the sub-sub title.

8. The explicit use of B0 should not be done when there exists an alternative instruction that reads better. However, you must use B0 when it is necessary for a complete comparison statement. For example: "NZ B3,JMP" should be used instead of "NE B3,B0,JMP". However, "GT B3,B0,LOOP" must be used instead of "GT B3,LOOP", and "LE B3,B0,EXIT" must be used instead of "LE B3,EXIT". If the program is to jump when B5 is plus, then use "PL B5,JMP"; however, if you are coding a loop that loops through a table from "n" through zero, then "GE B5,B0,JMP" should be used.

Of course, in most cases, this entire problem can be avoided thru the use of the JUMP, WHEN, REPEAT, etc. macros.

9. B1 should be used to hold a constant one if this is desired. Some older programs use B7; this practice must not be continued to new programs. If you make small modifications to programs that have another convention, then, of course, you should follow that convention. The Fortran 5 library appears to frequently use B5 = 1.
10. All subroutines must use the "UREG", "RREG", "EREG" and "CALLS" macros in the subroutine preamble if these macros are available. These macros will be available when the CPMAC User's Guide is available. This will allow machine verification of some parts of the linkage.
11. The "LOCK" and "UNLOCK" pseudo ops must be used to protect constant registers over a stretch of code. The interval before which protection is required varies by the register types. B registers are traditionally considered constant for a much longer period than A registers. If an A register is to be constant but its associated X register is not, then the A register should be locked even over short ranges. This becomes even more important if the A register was loaded "invisibly" through a macro. The purpose of locking registers is to prevent a quick change being installed that uses a register that it shouldn't.

5.10 CDC Peripheral Processor Coding

5.10.1 Direct storage usage

The direct storage locations between zero and 17 octal (D.Z0 thru D.T7) are used for temporary storage. This means that you must not expect their contents to be preserved over several levels of modules. This also means that if you are using a word in the temporary direct cells for a word count to read or write central memory, you must set that word as close to the actual read or write instruction as possible. For example:

<u>incorrect</u>	:	<u>correct</u>	:
SETK		LE .FNT,D.Z3	.
.		(Code altering FNT)	.
.			.
WRITE		FNT,D.Z3	.
MI	D.		SETK
			LE .FNT,D.Z3
			WRITEMI
			D.FNT,D.Z3

This will help prevent disastrous bugs when someone alters a temporary and does not notice its later use.

In the PP barnyard, while all these words are temporary some are more temporary than others. D.Z0 is the most temporary, D.T0 through D.T4 are the next most temporary (their contents must not be considered preserved over any PP resident request except a

call to R.TFL), then comes D.Z1 through D.Z5 and finally the remainder. The more temporary the word, the shorter the distance you can consider it preserved without extensive comments. (Note that a deadstart destroys D.Z0 through D.Z5).

Note that D.Z0 must not be used as the word count in a CM read/write instruction. That is, although "CWM BUFF,D.Z0" will work, it must not be used.

The temporary direct cells must not be renamed. In some existing PP code D.TO is renamed CM and is used to read the contents of central memory. This practice must not be continued to new PP programs.

If any of the constant direct cells (such as D.PPONE, D.TR, etc) are used in the program, they must be initialized at the start of the program and left constant throughout. This will prevent a modification making use of the cell before it has been set (this can create a very difficult bug).

The non-temporary direct cells must be renamed to indicate their use. There should normally be only one use in the program. However, for separate functions or states it is permissible to reuse a direct cell. The QUAL pseudo op should be used to prevent misuse.

When naming direct cells, all direct cells used (explicitly or implicitly) must be declared. This declaration must appear in one place and be visible to the person reading the listing. The declaration must be processed by COMPASS so that an error is generated if the allocation is done incorrectly. This style must be followed:

	ORG	D.TWO	
D.FNT	BSS	LE.FNT*5	has entire FNT
	BSS	D.PPIRB-*	free space
D.PPIRB	BSS	5	
D.RA	BSS	1	
D.FL	BSS	1	
D.ESTFWA	BSS	1	has FWA of EST
D.CPNUM	BSS	1	has control
	BSS	D.PPONE-*	point number
D.PPONE	BSS	1	free space
	BSS	D.PPIR-*	
D.PPIR	BSS	1	
D.PPOR	BSS	1	
D.PPMES1	BSS	1	

Note that there is only one ORG. Additional ORG's are permitted only if you intentionally plan to overwrite.

If multiple names for the same direct cell are used, it must be clear how the cells were set. For example:

at the beginning of the listing:

```
D.FNT    BSS    5
D.FST    BSS    5
FSTFT1   EQU    D.FST+C.FXYZ
```

Then somewhere in the program:

```
CRD      D.FST
```

and somewhat later occurs:

```
JUMP    EMPTY,FSTFT1,ZR
```

This will require breaking the line of thought to pursue thru the cross reference map exactly how and when FSTFT1 was set. Debugging and crash analysis becomes much more difficult because you have to remember extraneous details. The preferred sequence would be to simply use:

```
JUMP    EMPTY,FLD(D.FST+C.FXYZ,FXYZ),ZR
```

5.10.2 General practice

In PP code it is sometimes necessary to alter the assembled instructions. When this is done, care must be taken to insure that it is obvious that this is occurring and what instruction will appear. Frequently, for example, a constant is stored into an LDC instruction. When this is done, the following sequence must be used:

```
NME      LDC    **  has relative location of FET
NMEFIL   EQU    NME+1
```

or if several known sequences are possible

```
RWIO     OAM    **,CH    read/write disk
*        *        IAM BUFI,CH    FOR READING
*        *        OAM BUF0,CH    FOR WRITING
```

That is, if known, the instruction that might be placed there must be noted in the comments.

When accessing a central program field length, the absolute address can only be obtained just prior to the access. The absolute address must never be stored. The ADDRA or LDCA34 macros must be used to compute the absolute address; the error return must always be checked—even if it could "never" happen.

When writing multiple CM words, remember it is possible to be locked out of CM for a long time after the first word is written due to ECS transfers. Consequently, do not set requests until you are sure all of the parameters have been set. For example, a PP communicating to a CP program needs to write two words in central memory. The first word initiates the request, the second gives some vital parameters. While it would be tempting to do a CM write with a word count of two, this will not always work since the parameters may be set long after the request is made.

All PP's must have a test to prevent their growing too big. The CKPPLWA macro must be used to test for PP's loading within a fixed area (e.g. below 7777) since it rounds the overlay length up to a multiple of 5 words.

Not only should the end of the PP be tested, but also any parts of the PP that cannot be overwritten by an overlay it calls. Any PP routine that is modified which does not already contain this test must have the test included as part of the modification.

Some existing PP programs first test whether one of its overlays is already loaded and do not reload it before jumping to it if this is true. This is a highly dangerous practice. It has happened in the past that the test was not thorough enough and the PP jumped into garbage. Consequently, you must be very sure that this technique will save significant time and that there is no way to design the functions to avoid this problem.

No code should assume P.ZERO has a value of zero. P.ZERO used to have value of zero; it no longer does. Central memory location absolute zero is no longer expected to contain zero. Certain machine errors will cause the hardware to place error codes into location zero.

All PP programs should begin with the STARTPP macro as the first or second instruction executed (the first instruction may be a jump to the initializing code). STARTPP will request CM access; this should be terminated as rapidly as possible.

The "no-op" feature of the I/O instructions must not be used except for the "DCN" instruction and, if the guide lines in section 8 are precisely followed, the IAN,OAK instructions. This feature must never be used for the FAN, FNC instructions since a critical function may not be done.

A DPP or ABORT monitor request is made to terminate the PP program; all interlocks must have been released before these requests. Usually there should not be any code between one of these requests and the LJM R.IDLE.

5.10.3 New or significantly altered PP routines.

When a new PP routine is added to the system (or an old PP routine's function is significantly altered), there must be an entry added (or altered) to the /PPOV text. This entry will give the PP name and its function. The standards noted in the deck will be followed.

5.10.4 PP Macro Usage

The PP macros defined in the text should be used whenever appropriate (unless they have been declared obsolete). For example: the LDK macro must be used when loading a text symbol since its value may change (also, using LDK is encouraged because it denotes a constant; not infrequently do people mistakenly write LDN when they mean LDD and conversely).

The MOVE macro should normally be used only to move data. The optional instruction should be avoided - It must not be a flow changing instruction.

The ADDRA, LDCA34 macros should normally specify the error address and allow non-error conditions to flow to the next instruction.

Backward branches should insure that the proper jump (relative or long) will be calculated; forward branches must use whatever takes the least memory.

The macros defined in the PPMAC Users Guide should be used unless clarity is lost. Of course, those macros declared obsolete must not be used in new code.

6.0 INTERDATA coding

Some additional rules are needed for coding Interdata machines using COMPASS and FETEXT. These rules arise for a variety of reasons: undocumented hardware quirks, limitations of FETEXT and the loader, and problems that we have experienced.

6.1 Mechanics

Interdata code may be absolute or relocatable. The formats for COMPASS IDENT's are as follows:

Absolute assemblies:

```

          IDENT          name
          ABS
          FEPROG      f   wa,lwa,xferadd
                    .
                    .
                    .
constant definitions
                    .
                    .
                    .
          xferadd ____
executable code
                    .
                    .
                    .
          END

```

The ORG to fwa, and definition of the transfer address, are done by FEPROG.

Relocatable assemblies

```

          IDENT          name
          FEPROG
          ENTRY          xferadd
          EXT            externals
                    .
                    .
                    .
constant definitions
                    .
                    .
                    .
          xferadd ....
executable code
                    .
                    .
                    .
          END            xferadd

```

The transfer address is optional.

6.2 General restrictions

FETEXT and the loader place some additional constraints on the code:

- No relocatable IDENT may be greater than or equal to 8000₁₀ bytes since when the RX2 (relative addressing instruction format) is used, the address field may be exceeded.

- Unlabeled RMT may not be used to generate code in relocatable assemblies, since FEPROG uses it to round the IDENT to a multiple of 8 bytes long. Labeled RMT is not restricted.
- Labeled common may be used, but the programmer must ensure that each block is an exact multiple of 8 bytes long. Names must be less than seven characters long. There are no constraints on the use of blank common.
- Local USE blocks may be used, but must be padded to a multiple of eight bytes, or assembly errors will result. Names of USE blocks may not exceed six characters.
- Absolute values must be defined before their first use. If this is not done, unworkable code may result, without assembly errors. External values must either be defined before first use via the EXT pseudo-op, or may be prefixed by the "=X" modifier on each use.

6.3 Alignment problems

The ALIGN macro does BSSes until the origin counter falls on a 2-, 4- or 8-byte boundary. It is called by the .. and macros, as well as data definition macros such as HWORD, WORD, and BLOCK. The following code will not work :

```

LABEL    ..
          WORD    1234
          WORD    5678

```

The .. macro will align to a 2-byte boundary, and the WORD macro will align to four. Thus "LABEL" may not be the address of the first word. The macro should be used instead with full word tables.

Here is some more bad code:

```

          .
          .
          .
          BR      TAG
CELL     BYTE    23
TAG      LDB     R1,CELL

```

Since the instruction macros do not do any alignment, TAG will be on an odd boundary, and the machine will crash with an illegal instruction interrupt when the LDB is executed.

Any block of code following data definitions must begin with a `..` or `....` macro call:

```

CELL   BYTE   23
TAG    ..
      LDB    R1,CELL

```

Problems of this type may be avoided by always using the `..` macro in conjunction with labels - labels should never be put in the location field of instructions. Note that the use of the `..` macro also aids future modification.

6.4 Loading halfwords

On the 7/32, the LDH instruction extends the sign of the operand through the upper half of the register. Therefore it should be used only on signed values. LDHL must be used in other cases. Although in most general cases LDH would work fine, using LDH for only signed values is an excellent habit to get into - it can save much trouble.

6.5 Logical and arithmetic compares

The use of explicit CMP, CMPL, etc. instructions is not allowed. The WHEN/ORWHEN/ORELSE/DONE or JUMP or LOOP/WHILE/REPEAT macros should be used instead, for readability and terseness. One should beware of the following, however:

A careful reading of the 32-Bit Series Reference Manual will reveal that the CMP, CMPL, and CMPI instructions do not set the condition code the same way that CMPL, CMPLH, CMPLI, and CMPLB do.

This means that if you use the logical compares (which are faster, and CMPLB has no alternative), you must avoid a following BGT, BLT, BLE, BGE, BNM, BM, BP, or BNP. These jumps will work rationally after CMP, CMPL or CMPI, but they are unpredictable after a logical compare.

The jumps to use after a logical compare are:

```

BEQ    branch if A ,EQ.  B
BNE    branch if A .NE.  B
BL     branch if A .LT.  B
BNL    branch if A .GE.  B

```

6.6 BXH and BXLE

The index branching instructions do not work as you might think when the signs of the index and limit are different. For instance:

If R1=1, R2=1, and R3=-1, BXH on R1 will not branch, even though R1 .GT. R3.

If R1=-2, R2=1, and R3=1, BXLE on R1 will not branch, even though R1
.LT. R3.

Both of these instructions work correctly (algebraically) only when the
signs of index and limit agree, whether positive or negative.

Note that the BXLELOOP/ENDBXLE macros should generally be used in lieu of
explicit BXLE instruction loops. There may be times when the BXH
instruction is necessary, or when "tricky" coding is being done (such as
modifying the BXLE parameters explicitly), but these are generally shakey
and are to be avoided.

6.7 Register Conventions

The 7/32 has two register sets: set 0 is used in interrupt service
routines (ISR's) and SVC (supervisor call) routines; set F is used by
background tasks. Because of this separation, and because tasks only run
end-to-end, a task need not preserve any registers. However, there are
problems that arise because of subroutines that are callable by both
ISR's and tasks, and because of ISR's calling SVC routines.

6.7.1 RO and R1

In an ISR, these registers contain the old PSW—destroy them, and
you can't return to the task that was interrupted. (The same
applies to RE and RF in SVC routines and certain ISR's.)

6.7.2 ISR's and SVC's

When an ISR makes a SVC request (such as REQTASK), some of its
registers will be destroyed. SVC routines must be guaranteed to
preserve RO through R4, at least. ISR's should generally save
their REQTASKs for the very end, when register contents are no
longer needed.

6.7.3 Subroutine Parameter Registers

Where possible, try to use R5-R8 or R9 for passing parameters to
subroutines. This is true for tasks in FRENDR, and is useful when
the subroutine call goes only one level deep. With nested
subroutine calls, this rule can be abandoned.

6.7.4 Scratch Registers

Since a task needs to preserve no registers, it can use all for
scratch. Subroutines should use RA-RF (with caution about RC) for
scratch, and others only if necessary—R9 first, then R8, etc.

6.7.5 RC Problems

RC is universally used to hold the return address during a subroutine call. To allow nested subroutine calls, the SUBR macro stores RC at the front of the subroutine, and the RETURN macro reloads it.

It is nice to be able to exit a subroutine by doing a conditional branch to the contents of RC. This may be done, but only in short & simple subroutines, and only with a warning in the subroutine comments that RC must be preserved.

RC may be used as a scratch register, but only with a warning to do no branches to RC.

It is best to avoid references to RC except through the SUBR, RETURN, and CALL macros.

Since the SUBR macro defines only one cell to save RC in, any subroutine which is called by both tasks and ISR's may exit only by branching to RC—the RETURN macro must not be used. An alternative is being developed.

6.8 Disabling Interrupts

Cells which are modified by both tasks and ISR's must be interlocked. We do this by disabling interrupts temporarily in the task, using the DISABINT and ENABINT macros. Some cautions about this must be observed:

- DISABINT should generally only be used in tasks. Note that this macro becomes a no-op if interrupts are already disabled (it simply saves the old PSW, clears the immediate interrupt and system queue bits, and then ENABINT restores the old PSW), so it may be used in subroutines called from both tasks and ISRs, but this use is confusing and should be avoided.
- Interrupts must be disabled only for the minimum period of time. If two cells are modified in quick succession, it might be good to re-enable interrupts between steps.
- For good interlock management, try to keep code between DISABINT and ENABINT calls short, and straight-line. If a test must be made, do it after the ENABINT. Never branch into or out of a range where interrupts are disabled.

7.0 FTN Coding Standards

All new programs must be written in FTN5; any FTN4 program that is modified (except for minor changes) must be converted to FTN5.

7.1 Standard conforming

FTN5 programs should conform to the standard whenever possible. Nonstandard functions are preferred over nonstandard statements. This means, for example, that the AND and OR functions should be used instead of the ".AND." and ".OR." bit operators.

7.2 Indentation

The structure of the FTN5 program must be shown by indentation. Two columns should be used to indicate each level. Since FTN5 does not automatically list the program with this indentation, your source code must show it. Furthermore, any modification that alters the levels must include changes to correctly show the new structure.

Statements within the range of a DO statement should be indented.

The ENDIF, ELSE, and ELSEIF statements are indented at the same level as their associated IF statement. It is also usually beneficial to place a comment after an ELSE or ELSEIF statement explaining more precisely what is being tested.

Comments should be indented beyond the code in which they appear. That is, comments should not alter the visual presentation of the structure.

In general, the visual presentation of the structure is very important. Nothing should interfere with this presentation.

7.3 Symbols

Just as in Compass, symbols must be used when possible. This means that PARAMETER statements should be used for array sizes, character constants, etc.

7.4 DO loops

The "extended range" feature of DO loops must not be used. That is, no DO loops should exit in the middle and then return.

Any DO loop that expects to be terminated by some condition other than exhausting the loop count must still include code to handle the case when the loop exits by exceeding the loop count.

The DO statement should be of the form "DO slab, v=el_fe2[,e3]". That is, there should be a comma after the statement number.

The statement number that terminates the DO statement should be on a CONTINUE statement. The CONTINUE statement should be indented to the same level as the DO statement.

7.5 IF statements

The three branch IF statement should not be used.

The CASE statement should be mapped into the IF/ELSEIF/ELSEIF/ELSE statements.

7.6 GOTO statements

GOTO statements should be used sparingly. Refer to the general comments on program structure. GOTO statements should usually be used to code structured programming constructs not available in FTN5. For example, the GOTO statement can be used to EXIT a loop.

7.7 Intrinsic Functions

The generic name of the intrinsic function should be used whenever possible. For example, use MAX for the maximum function instead of MAXO, AMAX1, or DMAX1.

7.8 Diagnostics

All informative diagnostics must be eliminated whenever possible. However, clarity is much preferred over eliminating informative diagnostics. For example, it may be possible to avoid an informative diagnostics by changing a constant from a Hollerith constant to an octal constant, but to make this change would make the code more obscure; therefore, it should not be done.

There should be no stray or unused variables.

As with COMPASS code, even unrelated modifications must clean up informative diagnostics.

7.9 Subroutine usage

The use of LOCF subroutine must be avoided. This usually indicates some design that is either too assembly language oriented or else some module that should be done in COMPASS.

8.0 General Practices

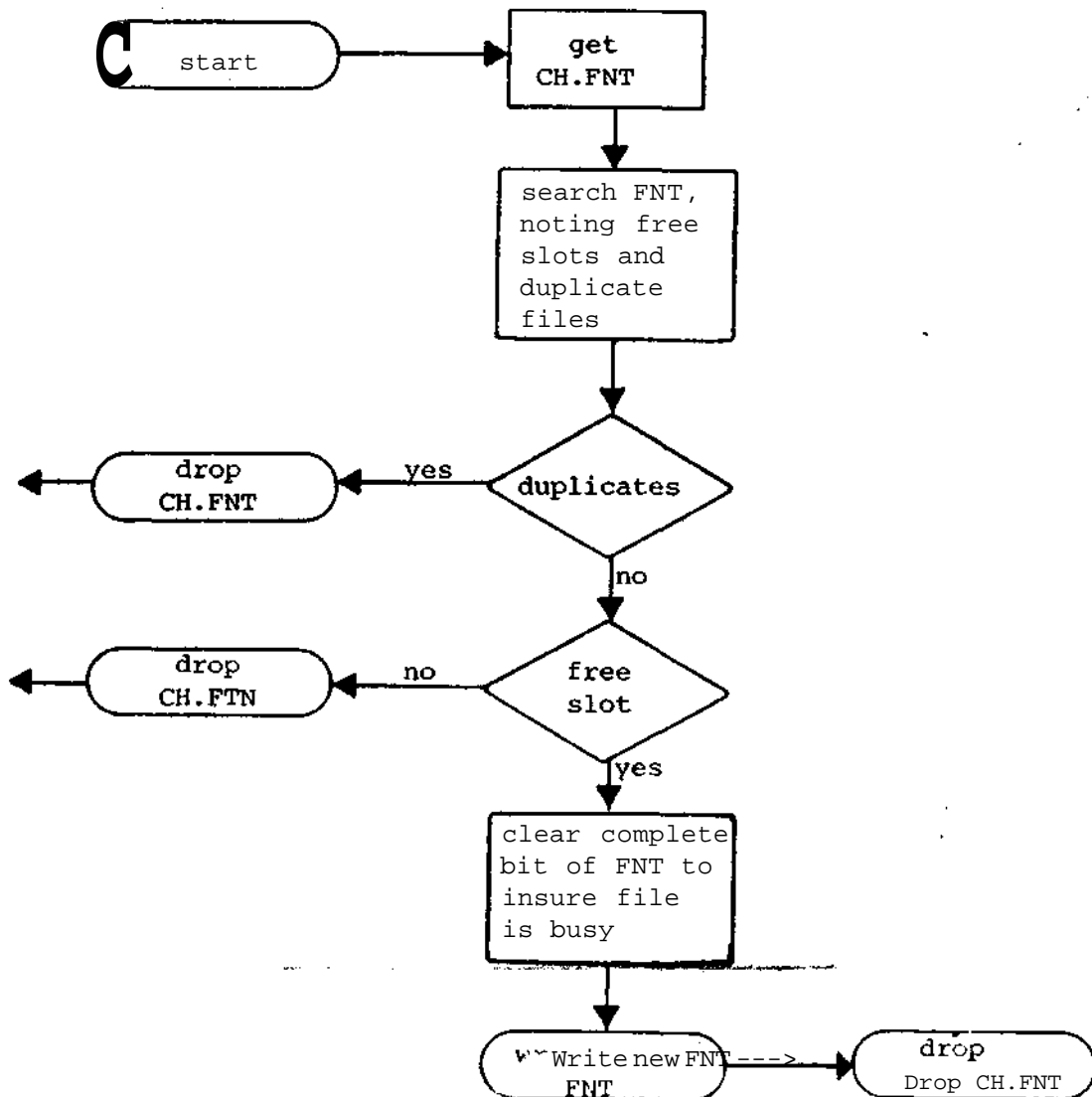
8.1 File manipulation

The proper interlock procedures must be strictly followed when creating, accessing, or destroying a file.

The two major processes which must be interlocked are: 1) finding a free FNT entry to put a file with a unique name, and 2) obtaining exclusive access to a file so that you can manipulate it or destroy it. Item two has many variations in our operating system. Discussed below are flow charts for these problems as they pertain to files assigned to a non-zero control point.

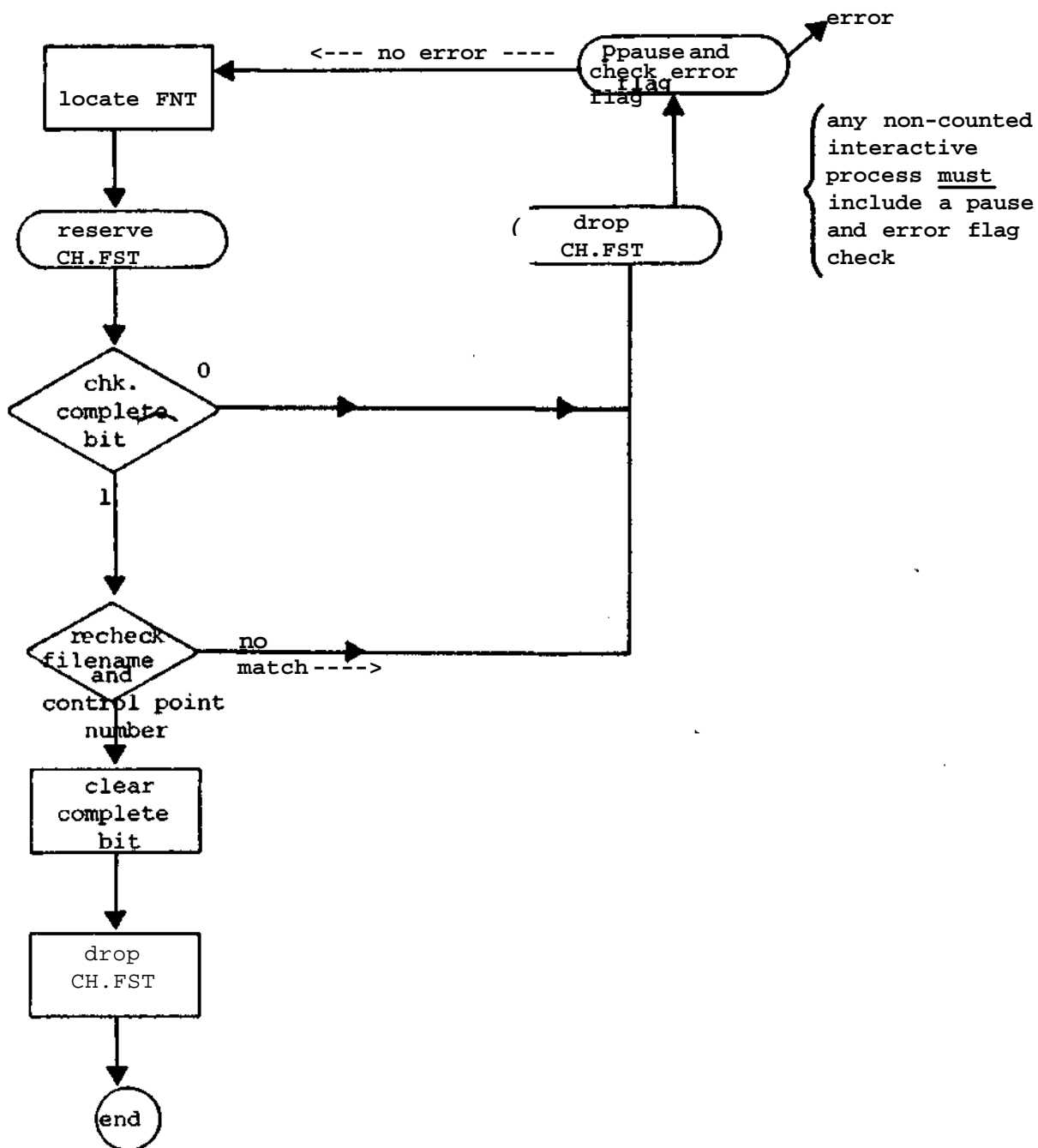
Creating a file:

If the file needs a unique name (file at a control point, for example), a search for a duplicate name should be done after the channel is reserved to prevent two PP's from generating the same file simultaneously.



Accessing a file:

To access a local file at a control point, follow the following sequence:



(to terminate exclusive access, set the complete bit).

Destroying a file:

Before destroying a file (zeroing its FNT), be sure to get exclusive access. If equipment is assigned to that file, or an RBT chain is present, be sure to deal with them before zeroing the file. Also, zero the FNT in the proper order: word three, word two, then word one (or word two, word three, then word one). Of course, this process should not depend on there being three words in the FNT.

Some programs (1AJ, 1EJ, etc.) don't bother with CH.FST, etc., as they are (presumably) the only program which might be attempting to access the file in question. While this scheme does perhaps save time, employing it deserves a (very) long, hard examination.

8.2 Use of the no-op feature in the OAN,IAN instructions

In addition to no-op DCN (DCNPSN) the no-op (unhangable) feature of the OAN/IAN instructions (OANPSN/IANPSN) can be used to avoid hangup situations:

code like

```
FCN
ACN
LDD
OAN
LDD
OAN
DCN
```

can hang if the equipment doesn't accept the 1st byte output (2nd OAN hangs on a full channel). A DCN entered by the console operator doesn't help because then the OAN is hung on an inactive channel. The channel must be DCN,ACN by the operator to unhang the operation, and then the program cannot detect what happened.

The above sequence also hangs if the equipment DCN's the channel between the ACN and OAN. OAN's hang on an inactive channel, and it is unlikely that an operator can distinguish a channel hung inactive from a channel not being used.

This problem has two solutions:

1) use OAM:

```
FCN
ACN
LDK word count
OAM
DCN
NJN error (not all data taken)
```

This requires data properly formatted in memory.

2) use OANPSN:

```
FCN
ACN
LDD
OANPSN
IJM error
DCN
```

Either sequence allows operator to DCN a channel hung full, and will take the error branch if it happens. Similarly, if the channel is deactivated by the equipment, both sequences execute without hanging and branch to the error routine. The case for IANPSN is virtually identical (except replace "full" with "empty" and vice versa).

IANPSN and OANPSN sequences must be followed by a "IJM error" to detect the failure condition. Obviously, this cannot be used if the equipment deactivates the channel after data transfer (as happens on status sequence for 7054):

```
FNC 0012B
ACN
IANPSN
IJM error
```

This cannot be used since the equipment will (legitimately) deactivate the channel after the IANPSN instruction. Therefore, the PP cannot determine if the inactive channel is correct or if it occurred because the operator DCN'd the channel.

```
FNC 0012B
ACN
LDN 1
IAM STAT
NJN error
(wait for channel to be inactive)
```

This is correct, as the controller will deactivate the channel after the IAM. If the operator had to DCN the channel, not all the data would have been transferred and the non-zero A-register would have indicated this situation.

8.3 Writing into ECS (or reading from ECS)

The ECS I/O process is described in 6SM 71. It should be followed. All ECS access must be relative to a given partition.

8.4 Channels and Interlocks

If possible, only one channel or other interlock should be held by a PP or task at a time (and it is usually possible). If it is necessary to obtain two or more interlocks simultaneously, the second and subsequent interlocks must be requested with refuseable requests. If a request is refused, ALL interlocks obtained to that point must be released, and the process restarted with the first interlock request (normally after a short delay and a check of the error flag if appropriate).

8.5 Files

Scratch files should have four Z's as the first four characters in their name. Any program may use a scratch file without concern for its previous contents. (Files with five Z's are reserved for use by CDC)

Connected input and output files are named depending upon the character set being used:

character	file name	
<u>set:</u>	<u>input</u>	<u>output</u>
DC	ZZZZIN	ZZZZOU
AF	ZZZZIAF	ZZZZAF
BF	ZZZZIBF	ZZZZOBF
AS	ZZZZIAS	ZZZZOAS
BI	ZZZZIBI	ZZZZOBI

All scratch files must be returned after execution.

8.6 Recovery

Any new features that implement new tables or fields in CMR or the control point areas or add new file types, etc. must consider the questions of recovery after a crash, of rerunning a job, etc.

8.7 Memory usage

In most cases memory is our most valuable asset. Don't waste it. Frequently, this means using several overlays, using I/O buffers for set-up code, etc. However, reliability, readability, and flexibility must remain.

8.8 Documentation

Documentation on modifications is expected to come with or soon after the code. This will mean updating the appropriate SMD or writing a new SMD. See STRAP 4. An M4 may also be written if you wish. If the SMD is not in machine readable form and if the modification meets the qualifications of an M4, then only an M4 needs to be written. See STRAP 11.

9.0 Summary

These conventions and practices, intelligently applied, should result in flexible, reliable code. Rapid initial development is rarely required - following these guidelines is usually more important.

WRITTEN BY: Richard R. Moore

(with comments and suggestions from the entire systems group)

MICHIGAN STATE UNIVERSITY

COMPUTER LABORATORY

SYSTEMS TASKS, RESPONSIBILITIES, AND PROCEDURES

NUMBER 10.1

Software Stir Procedures

January 22, 1978

1.0" INTRODUCTION

Problems with the MSU computer operating system are reported to the Computer Laboratory by use of a Systems Trouble Internal Report form (a STIR)- Any of these STIRs which refer to software problems are sent to the Systems Programming Group (Systems). This document describes the procedures used internal to the Systems Group to track the progress of software STIRs.

2.0 SYSTEM^ PERSONNEL INVOLVED

Three people in Systems are involved with a STIR at any one time. These people are:

The STIR MONITOR

The STIR MONITOR'S duties are to keep a record of the progress of all STIRs, and to produce reports containing this information. This person is aided in this task by the STIR ASSISTANT.

The STIR ASSISTANT

The STIR ASSISTANT'S duties are to handle all the paperwork and actual information recording which goes with the STIR process. This person keeps books containing all the STIRs that have been through systems. This person follows the STIR MONITOR'S direction.

The ASSIGNEE

The ASSIGNEE is the person in the Systems Group who will actually do the work of isolating and correcting the problem. This person has a large number of options as to the action to be taken to correct the problem. These are described in section 3-

3.0 STIR FLOW WITHIN SYSTEMS

This section describes in detail the paths a STIR may follow between the time it is submitted to the Systems Group and the time that the problem is corrected.

3.1 NEW STIRS

All new STIRs go directly to the STIR MONITOR, who does the following:

- a) Establishes a title, routine, category, priority, and number.
- b) Makes an initial assignment.
- c) Backs up all needed files on magnetic tape.
- d) Writes the Systems STIR number on all documentation.
- e) Attaches an acknowledgement form.
- f) Routes the yellow copy, all documentation, and the acknowledgement form to the ASSIGNEE.
- g) Routes the other copies to the STIR ASSISTANT for logging.

The ASSIGNEE should then analyze the STIR within 5 days. This action will result in a positive acknowledgement of the disposition of the STIR. It will be one of the following:

- a) User error. The STIR is returned to the STIR MONITOR with an explanation.
- b) CDC bug. A PSR is written and returned with the STIR.
- c) MSU bug. The attached acknowledgement form is detached from the STIR and sent to the STIR MONITOR. Any needed changes in routine name, priority, and/or category may be specified on this form.
- d) Inadequate documentation. In this case the STIR MONITOR will request the person who submitted the STIR to supply any needed materials. If the submitter cannot or will not supply what is needed, the STIR will be rejected.

If the STIR MONITOR does not receive an acknowledgement from the ASSIGNEE within five days, the STIR MONITOR should take any steps necessary to get an immediate acknowledgement. These steps could include talking with the ASSIGNEE, reauesting support from the ASSIGNEE'S project leader , requesting support from the Systems supervisor, etc.

3.2 CHANGING STIR RECORDS

A number of the pieces of information that the STIR MONITOR records about a STIR may be changed to keep the record up-to-date. These items are:

- a) Routine name
- b) Category
- c) Priority
- d) Description
- e) Assignee

To change any of these items, the ASSIGNEE should send the STIR MONITOR both the yellow copy and a note indicating the desired change. The STIR MONITOR will instruct the STIR ASSISTANT to change the records and make the change on the white copy. The STIR MONITOR will mark the chanr •> on the yellow copy and return it to the ASSIGNEE.

Note that in the case of a re-assignment, the ASSIGNEE will now be a different person, and the original ASSIGNEE will not receive the yellow back.

3.3 ANSWERED AND FIXED STIRS

If the STIR is to be replied to or rejected, the yellow copy is returned to the STIR MONITOR with a signed explanation written in the 'reply box'. The STIR MONITOR will log the reply, and send the STIR to the STIR ASSISTANT, who will send the reply to the submitter.

If the STIR has been fixed by a modification to the operating system or one of the dependent products, the yellow should be returned to the STIR MONITOR with the LSD and IDENT of the modification noted in the 'reply' box. Note that this should only be done after the modification has been installed.

The STIR MONITOR should monitor the LSD documents to be certain that a yellow copy is received for each STIR listed as fixed.

If the STIR has been fixed by a modification, but is still to be PSR'd to CDC, the LSD and IDENT of the modification should be written in the reply box, and then the STIR and PSR sent to the STIR MONITOR. The user will be notified that the problem has been corrected, and the PSR will be sent to CDC.

3.4 PSR'S

A PSR (.Programming .Systems Jgeport) is CDC's equivalent of a STIR. It is used to report problems with software which is supported by CDC.

When a STIR is to be PSR'd, the completed PSR form and all documentation, and the yellow copy, should be sent to the STIR MONITOR, who, with the STIR ASSISTANT, will handle all paperwork involved in sending the PSR to CDC.

The documentation supplied to CDC with the PSR should display the problem as clearly as possible. The program should be reduced to the minimum which will produce the problem, and the error should be clearly pointed out. Note that it is a good idea to mark the output to show exactly where the product is not performing correctly.

The PSR documentation should include the following where applicable:

- a) A punched card deck of a batch job which shows the problem.
- b) A permanent file containing this batch job. (An EWFILE is preferred.)
- c) Two copies of the output which shows the error. This should be the execution of the above batch job. One copy will be sent to CDC's local representative and the other will be filed by the STIR MONITOR.
- d) The PSR form, with the problem clearly described.

To send a PSR to CDC, the STIR MONITOR verifies that the supplied materials are sufficient to explain the problem. Two photocopies of the PSR form are made. The original PSR form, one photocopy of it, and one copy of the STIR documentation are sent to the STIR ASSISTANT. The STIR ASSISTANT logs the action, sends the photocopy to CDC's local representative, and sends the original PSR form and the documentation to CDC's office in Sunnyvale. The STIR MONITOR files the following items:

- a) The yellow copy of the STIR.
- b) The second photocopy of the PSR form (attached to the yellow copy).
- c) The original documentation.
- d) A copy of the documentation sent to CDC (unless identical to [C]).
- e) Copies of any permanent files submitted (on tape).

(CDC does not return PSR documentation. All of this filing will enable the problem to be resubmitted to CDC or worked on locally if necessary.)

If the STIR is to be fixed locally in addition to the PSR, this should be noted in the 'reply' box. The STIR MONITOR will then return the yellow to the ASSIGNEE. This STIR should be sent back to the STIR MONITOR when the local fix is installed.

The STIR MONITOR will track the progress of all outstanding PSR's. This is done by scanning the PSR SUMMARIES sent from CDC. All action by CDC regarding a PSR is recorded on the yellow copy (in the STIR MONITOR'S file) and in the STIRLOG. When a PSR is closed by CDC (whether rejected or fixed) the STIR MONITOR will retrieve all the original documentation from the files, re-attach it to the yellow copy, and return the STIR to the original ASSIGNEE with a note of the disposition by CDC. The person receiving the returned STIR/PSR should take one of the following actions:

- a) If CDC has fixed the problem, either by issuing corrective code or by reporting that the problem cannot be reproduced at a higher level, hold the STIR until the fix is installed. The STIR is then handled like any fixed STIR.
- b) If CDC has refused to fix the problem, and we have fixed it locally, the STIR may immediately be replied to as fixed.
- c) If CDC has refused to fix the problem, and we will not be fixing it locally, the STIR should be replied to as 'NOT TO BE CORRECTED'.

PSR's which CDC could not reproduce should always be retested. If the problem still occurs after the product is brought up to the level at which CDC tested it, the STIR should be treated as an MSU problem.

Any PSR for which the reply from CDC is incomplete or unacceptable should be resubmitted to CDC. Additional documentation should be incl >d to clearly show why CDC's

initial response was inadequate. The systems supervisor should be informed whenever this is done.

Problems with Minnesota FORTRAN are reported to the University of Minnesota with a MNF Bug Report. The Bug Report is handled like a PSR, except that no photocopy is sent to the University, so only one photocopy of the form is needed.

3.5 PERMANENT-FILE-VERSIONS

When a permanent file version of some product which contains a fix for a STIR is created and made generally available, the STIR(s) which are fixed in this version should be sent to the STIR MONITOR with the permanent file name noted in the 'reply' box. The STIR MONITOR will notify the submitter of the availability of the corrected version. This permanent file should remain on the system until the fix is installed.

3.6 RSM'S

When a STIR is received which describes a feature which works according to the documentation, but really should be changed, it can be classified as an RSM (Request for Software Modification).

These STIRs will be dropped to priority zero, but will remain in the STIRLOG for consideration when modifying the product.

3.7 REJECTIONS

A STIR may be rejected for several reasons. These reasons are:

1. If the documentation supplied is insufficient to reproduce the problem, and the submitter cannot or will not supply what is needed, the STIR may be rejected. (Note that the STIR form states that 'you must supply adequate documentation to make the problem easily reproducible'.)
2. A STIR may be rejected if the user was in error; that is, if the program or feature worked as documented.

For a rejection the ASSIGNEE simply states the reason for rejection in the 'reply' box, and sends the STIR to the STIR MONITOR. The STIR MONITOR logs the rejection and sends the STIR to the STIR ASSISTANT, who sends the reply to the submitter.

3.8 DOCUMENTATION CHANGES

If the program or operation worked as intended, but the documentation is incorrect, the ASSIGNEE may request a documentation change. This is accomplished by replying to the software STIR as 'documentation change¹', and writing a documentation STIR. Local documentation STIRs should be sent to the User Information Center. CDC documentation STIRs should be PSR'd directly to CDC.

The document which will be modified should be noted in the reply box.

4.0 STIR REPORTS AND THE STIRLOG

All the information about active STIRs is kept by the STIR MONITOR on a permanent file data base. This data base is known as the STIRLOG. The STIR ASSISTANT handles the majority of data entry into the data base. The STIR MONITOR takes care of any unusual situations or problems that arise.

"4.1 STIR REPORTS

The STIR MONITOR is responsible for producing reports of progress on currently active STIRs. The following reports are produced:

Every week:

1. A SUMMARY REPORT, containing any STIRs for which some action took place in the last week.
2. A FULL REPORT, of all STIRs currently in the STIRLOG. This is used as a reference by the people working on STIRs.

Every month:

1. A FULL REPORT, showing the progress on STIRs in the past month.
2. An ASSIGNMENT REPORT, detailing the STIRs assigned to each person or group.

The weekly SUMMARY REPORT is produced interactively with the following control sequence:

```
HAL,L<STIR,LIB.  
REPT,OUT.
```

Enter '(space)*' in response to the prompt for input and dispose the file OUT to print when done.

To produce the full weekly reports, simply type:

```
HAL,L"STIR,E-WEEKLY.
```

Both monthly reports are produced by the same exec file. The control card sequence is:

```
ATTACH,MONTHLY,MONTHLYSTIRREPORTGOFILE.  
USE,MONTHLY.  
GO.
```

This produces the assignment lists and a preliminary copy of the full report (for approval by the systems supervisor) after prompting for the first date of both the current and the previous months. To produce another assignment list and the additional copies of the full report, replace GO with GO,'DATE','FINISH¹.

4.2 STIR CATEGORIES

The following categories are used in the STIRLOG.

<u>CATEGORY</u>	<u>PROGRAMMING SYSTEM</u>
AF	Authorization File
AL	APLIB
AP	APL
AR	ARGUS
BA	BASIC
CB	COBOL
CD	Cyber Data Control System (CDCS)
CM	COMPASS
CP	CPUMTR and MTR
CR	Cyber Record Manager
DD	Data Description Language
DG	Diagnostic routines (CEFAP)
DT	DUMPTAP
DU	Data Base Utilities (DBU)
ED	MSU EDITOR
El	Export/Import 200
EL	EDITLIB, LIBEDIT, and PRE
ES	ECS Related
FE	FTN Compiler
FO	FORM
FR	Front End System
HL	HAL
HP	HAL Programs
LD	Cyber Loader
ME	MERIT Computer Network
MF	Minnesota FORTRAN
MI	Interactive Subsystem (MISTIC)
PI	PP Interpreter
PF	Permanent File System
QU	QUERY UPDATE
RC	Recovery (IRCP)
RM	MSU Record Manager
SC	SCOPE 3.2 Operating System
SO	SORTMRG
SP	SPSS
SY	SYMPL
TP	Magnetic Tapes
UP	UPDATE
UT	Utility Programs
8B	8-Bit Subroutines

4.3 STIR PRIORITIES

STIR priorities are based upon 2 factors:

1. The inconvenience to the user.
2. Where the STIR currently is assigned.

Priorities are assigned as follows:

5 - CRITICAL

Assigned when the user cannot circumvent the problem, i.e. the user is stopped dead until the problem is

fixed. A STIR remains at priority 5 only as long as no work around exists.

4 - REGRESSION

A feature used to work, but is now broken. These problems are priority 4 to ensure that existing features continue to work regardless of other system changes. In addition, certain very serious bugs will be given a priority of 4. These frequently involve a new feature which shows a serious problem just after it is installed. A bug that can cause catastrophic problems, such as destroying a workfile, may also be assigned a priority of 4.

3 - SERIOUS

A feature does not work as documented, interfering with the correct execution of a program or operation. Generally a user will be able to work around these problems, but with some effort.

2 - MINOR

A feature does not work as documented, but this does not interfere with the task the user is trying to accomplish. These problems may be regarded as inconveniences. Generally no work-around is needed, but some further explanation may be required (as in the case of a misleading error message).

1 - PSR'd

Any problem which has been PSR'd to CDC will be dropped to priority 1. This is to differentiate it from STIRs which require active work from SYSTEMS. Priority 1 indicates that we are waiting for an outside party to fix the problem.

0 - RSM'd

Any STIR which requests a feature addition, or a change in specification, will be classified as an RSM, (Request for Software Modification). Additionally, problems judged to be not of sufficient importance to warrant prompt repair may also be placed in this category.

blank - CANNOT REPRODUCE (CLOSED)

STIRs describing problems which cannot be reproduced, but are verified to have occurred, will be replied to. These STIRs will be carried in the STIRLOG for at least 6 months after they are closed to ensure that recurrent, but sporadic, problems are properly recognized.

NOTE - These guide lines apply only to STIRs which affect users. Any STIR which affects only the Systems group should not have a priority over 2. Any STIR which affects only some Computer Lab internal group and has a work-around should not be given a pr...city greater than 2.

This insures that > ^k will ^e done on user problems.

4.4 STIRLOG ACTION CODES

The path of a STIR is recorded in the STIRLOG by use of 'ACTION CODES'. The following codes are used:

- ASG - Assigned by the STIR MONITOR.
- CDC - PSR sent to CDC.
- CLO"- The problem has been examined, but is not reproducible.
- FIX"- The STIR is fixed in the LSD noted.
- FXL - Fixed locally. Will be PSR'd.
- INQ - PSR acknowledged by CDC.
- MSU - Acknowledged as a local problem.
- PFV - Corrected in permanent file version noted in reply.
- REJ"- Rejected.
- REP«- Reply sent.
- RTF - PSR returned by CDC with a fix which requires installation.
- RNF - PSR returned by CDC with no fix (i.e., user error, documentation change, or refusal to fix).
- RSM^k- Held for consideration. (See priority=0)
- UMN - Bug Report sent to University of Minnesota.

A '.' denotes a final action. These codes result in a final reply being sent to the submittor, and the yellow copy takes up residence in the 'fixed' books. The entry is removed from the STIRLOG (except for RSM's, which are kept on the STIRLOG until the modification is made or the request is no longer relevant).

4.5 SUMMARY OF ACTIONS FOR EACH CODE

The following is a summary of the actions that result for each STIRLOG entry.

<u>CODE</u>	<u>ACTION(S)</u>
ASG	YELLOW - ASSIGNEE
CDC	YELLOW - STIR MONITOR
CLO	WHITE - submittor YELLOW - fixed book PHOTOCOPY - STIR MONITOR (filed with documentation)
FIX	WHITE - submittor YELLOW - fixed book
FXL	REPLY - submittor YELLOW - STIR MONITOR
INQ	NONE
MSU	NONE
PFV	REPLY - submittor YELLOW - last ASSIGNEE
REJ	same as FIX

REP same as FIX
RIF YELLOW - last ASSIGNEE
RNF YELLOW - last ASSIGNEE
RSM WHITE - submittor
YELLOW - fixed book
UMN YELLOW - STIR MONITOR

WRITTEN BY: Remett E. Jensen

APPROVED BY: John K. Rauer

SECTION MAP

<u>SECTION</u>	<u>PAGE</u>
1.0 INTRODUCTION	1
2.0 SYSTEMS PERSONNEL INVOLVED	1
3.0 STIR FLOW WITHIN SYSTEMS	1
3.1 NEW STIRS	2
3.2 CHANGING STIR RECORDS	2
"3.3 ANSWERED AND FIXED STIRS	3
3.4 PSR'S	3
3.5 PERMANENT FILE VERSIONS	5
3.6 RSM'S	5
3.7 REJECTIONS	5
3.8 DOCUMENTATION CHANGES	5
4.0 STIR REPORTS AND THE STIRLOG	6
4.1 STIR REPORTS	6
4.2 STIR CATEGORIES	7
4.3 STIR PRIORITIES	7
4.4 STIRLOG ACTION CODES	9
4.5 SUMMARY OF ACTIONS FOR EACH CODE	9

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 11.2

MSU Mini Mod Memo (M4)

June 4, 1984

To alleviate the documentation burden, the MSU Mini Mod Memo (M4) is available. This type of memo is for documenting small changes; changes which don't affect the user, changes which are described by one or two paragraphs. For more details on documentation requirements refer to STRAP 4.

The M4 can be used as an appendix to an existing SMD, or as a stand-alone document. To use:

1. Attach the permanent file "M4 FORM SKELETON EWF" and copy it or obtain a blank M4 form from the technical assistant (T.A.).
2. Fill out title, coder, date
3. Fill out idents, routines
4. Fill in DESCRIPTION: include any references, justifications, etc.
5. Fill in EXTERNAL CHANGES: include operation procedure changes, change in the way that the routine interacts with the system. Note space for DAYFILE messages, new symbol definitions and new usage of system tables and pointers.
6. Fill in INTERNAL SPECIFICATIONS: besides describing how it works; include any cautions, assembly options, and any notes on modifications. This must not duplicate the incode documentation.
7. Send it to your project leader for review and comments.
8. Go to the T.A. and get either an M4 number (if this is a stand-alone memo) or an appendix number for that specific SMD, and put the number in the top center box.
9. Print the appropriate number of copies and give to the T.A. for distribution (the T.A. can tell you the number of copies needed).

Do not use an M4 if:

1. There is more documentation than fits on a page
2. There are user changes (unless this is an appendix to a SMD)
3. The modification is significant enough to deserve a SMD

If there exists a machine readable SMD (see STRAP 4), it must be updated even if an M4 is written. In this case, the M4 would save reprinting the entire SMD (or parts of it).

WRITTEN BY: Richard R. Moore

(original idea by David M. Dunshee)

MSU Mini Mod Memo

PL _____ _____ _____ installed in LSD _____	IDENT _____ _____ _____	ROUTINE _____ _____ _____	fill in one: M4 # _____ appendix to SMD number: _____	Title: _____ _____ coded by: _____ date: _____		
1.0 Description (What was done; why; general effects; references.)						
2.0 External Changes (operational changes; changes affecting the rest of the system.)						
<table style="width: 100%; border: none;"> <tr> <td style="width: 50%; border-right: 1px solid black; padding-right: 10px;"> <i>System file changes (dayfile, etc.)</i> </td> <td style="padding-left: 10px;"> <i>New symbols; Table usage changes</i> </td> </tr> </table>					<i>System file changes (dayfile, etc.)</i>	<i>New symbols; Table usage changes</i>
<i>System file changes (dayfile, etc.)</i>	<i>New symbols; Table usage changes</i>					
3.0 Internal Specifications (How the change works. Include cautions and assembly changes.)						
*** use an M4 only if the information fits on it (one page), and user changes are insignificant. See STRAP 11.						

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 12.4

Disk Labeling and Flawing

June 7, 1984

Every fixed disk unit or removable disk pack has a label found in the first RB of the first RBR of the device, and which contains information identifying the disk and its contents to the system. This includes the RBR ordinal to be used for the first RBR of this disk, a table of flawed RBs for each RBR of this device (called flaw tables), and optional pointers to a PFD and RBTC if this disk is a PF device. For further details on the label contents and structure see common deck /LABEL in the texts.

This document will describe the various procedures and utilities for writing disk labels, determining flaws and adding them to the labels, and writing a new PFD/RBTC on a disk unit.

1.0 Basic Label Writing Sequence

This is just a prototype label writing sequence. It can and will be slightly different depending on what exactly is being done.

A PFD/RBTC can be written only by a PFD/PFC initial deadstart and the disk must already have some form of label on it.

If the disk does not already have a label, only a FLAW initial deadstart can write a new label.

1.1 Determine if there are files on the pack to be labeled that cross any of the RBs that will be removed by flawing or will be over-written when a new PFD/RBTC is created. If so, they must be removed.

If the RB to be flawed is in the PFD/RBTC area on a PFD or ALT pack, it is necessary to write a new PFD/RBTC on the pack after the flaw is set. This will cause the PFD/RBTC to migrate into the next RB on the pack. If a file crosses this RB, it must be cleared from the pack.

To determine the RB that the PFD/RBTC will migrate in to, use FSTAT and CHAIN from SYSTEMS library to find and look at the word pair chains for the appropriate RBTC file; ORBTC for the primary RBTC and 1RBTC for the alternate RBTC. Find the last RB byte in the last word pair and add two (2) to its value.

Use 'PFLIST,RB=rbrrb.' to determine if there is a file crossing the RB to be flawed. If so dump, purge and reload the file with the drive set to LBL to prevent the RB from being rewritten. The dump and reload is usually done by operations during production upon receiving an appropriately routed request.

If the pack to be flawed is currently used in production, the drive should always be set to LBL (or OFF) until the initial deadstart to prevent any files from being written in the RB to be flawed.

- 1.2 If necessary, mount any disk packs that are to be labeled on the appropriate disk drive. The assignment of an RBR ordinal to the first RBR of a disk pack is made according to the EST ordinal of the disk drive at the time the label is written. The RBR ordinal will be the ordinal of the first RBR found in CMR that has the same EST ordinal as the disk drive writing the label on the pack.

This means, to get a desired RBR ordinal assigned as the first RBR on a pack, you must physically mount the pack on the drive in the correct EST position. You can not set the first RBR by any command.

- 1.3 Physically turn off or write-protect all disks except those that are to be labeled. This is to prevent accidental erasure of critical data such as the PFD/RBTC of a PF device.
- 1.4 Deadstart and bypass the dump option matrix. You should now see the deadstart option matrix.
- 1.5 Indicate that an initial deadstart will be done as follows:

Set word T.IDSFLG(45B) to the type of initial deadstart desired, i.e.

45=06140127555555555555 (10HFLAW) or
20060450200603555555 (10HPFD/PFC)

- 1.6 When the EST display appears, logically turn off all disks except those that are to be labeled. IRCP will not attempt to process any disk that is 'OFF¹'. Finish with just a carriage return.
- 1.7 As insurance that you intend to label the disks, a warning message that requires you to type in 'INITIAL' is displayed. Do it and go on.

- 1.8 Next you usually get a message asking whether or not you want to do the default label sequence. This is intended primarily for initial deadstarts done by operators, and usually you type 'NO', or enter CR (NO is the default).

This message may not come until after making an entry in the label information display described in the next section. After responding with "no" the deadstart returns to the label information display.

- 1.9 Then there is a display of label information on the left screen and a display of the flaws in the first flaw table on the right screen.

The label information will have been taken from the existing label or from default setting in IRCP. You should check and if necessary change the following information:

- a) The VID (visual ID). For proper forms of the VID see STRAP 14.
- b) The date that the label was written. This is of the form 'mnddy'.
- c) The disk family number (denoted by 'FAM,n', n=1,2,3). Family 3 is reserved for non-removable disk devices (like the 885). Removable disk devices are divided into families 1 and 2. For each device type, there must be at least one member in each family to ensure the success of a 'clear pack' operation should a disk drive fail. The family number should be changed only when there are no PFs on the disk. The system uses the family numbers in determining to which device a file may overflow, by attempting to keep the file always on the same family of disks. The family number is automatically determined from the EST.
- d) The default disk flag (denoted by 'DDF,YES" or 'DDF,NO'). This flag indicates whether or not the system can allocate files to this disk by default - that is, without the job explicitly requesting the device. Currently, all regular (non-PFD) disks are set with "DDF,YES'. The PFD units (PFD and ALT) are set with 'DDF,NO¹.
- e) Set and/or clear flaws in the various flaw tables by using 'FT,n' or '+' or '-' to display to flaw table number 'n', 'FLAW,nnnn' or 'FREE,nnnn' to flaw or free (un-flaw) RB number 'nnnn' in the current flaw table. Each flaw table corresponds to one RBR assigned to the device.
- f) If this is a PFD/PFC initial deadstart, use the 'PF,YES\ 'PF,NO', and 'PF,ALT' commands to declare the device to be either a primary PF device, not a PF device, or an alternate PF device, respectively. Note that this command is ignored by a FLAW initial deadstart.

Note that the RBR ordinal of the first RBR for the device is displayed on the left screen as 'RBR^n'. 'RBR,nn'. The RBR ordinal for the current flaw table within the device is displayed on the right screen, along with the flaw information for that flaw table.

3.0 Determining What Flaws Need to be Set

There are several ways to gather flaw information. In some forms the flaw information is presented in physical address form (cylinder, track, sector) rather than logical address form (RBR, RB, PRU) and you must convert it. (NOTE, 'head' and 'track' are used interchangeably in different pieces of documentation.) In some cases the physical addresses are reported in decimal, and in other cases in hexadecimal. There are many utility programs on L*SYS to assist you. These are fully documented on L*SYS, but a short description will be given in section 7.0.

4.0 Factory detected flaws.

With each disk pack we get from CDC there is a sticker on the base of the cannister listing the physical addresses of the flaws detected at the factory (except for some of the older single density packs). These addresses are for individual sectors or for entire tracks.

The individual sector flaws are specified by a 7-digit decimal number. The first 3 digits are the cylinder number, the next 2 digits are the track number, and the last 2 digits are the sector number. Entire track flaws are specified by a 5-digit number which is interpreted just like the sector flaw numbers except that the sector digits have been omitted.

To convert these to logical addresses you may use the program RBCYL on L*SYS. For further details consult section 7.0 or the documentation on L*SYS.

5.0 Surface analysis detected flaws.

Occasionally the CDC CEs here at MSU will perform a surface analysis of a disk pack. This procedure checks out all the sectors of a disk for errors and reports them in a printed output. The error reports are very detailed and (unfortunately) the numbers are in hexadecimal. To ease the hassles of figuring out what to do with all this information there is a program FLAW844 on L*SYS to help you out. You can prepare a summary of the surface analysis on a file with the numbers still in hexadecimal (basically just extracting information from the printed reports) and give it to FLAW844 and it will break it all down into easy to understand and useful information. For further details consult section 7.0 or the documentation on L*SYS.

6.0 Flaws detected by the running system.

Whenever ISP (the disk driver PP) encounters a parity error on a disk I/O operation, whether it is recoverable or unrecoverable, it will put a series of dayfile messages into the system dayfile describing what happened, what disk location was involved, etc. When DAYFILE is run ('n.DAYFILE,M1") it will copy any such messages from the system dayfile onto the permanent file 'PFERRORS' where they can be subsequently analyzed. The program DPE on L*SYS may be used to access this PF and provide a summary and a copy of these parity error messages. The output of this provides the logical as well as the physical addresses of the disk locations involved. For further details consult section 7.0 or the documentation on L*SYS.

To make effective use of this you need a method to get ISP to exercise the disk unit. To this end there are two utility programs on L*SYS.

- 1) WRITDSK will simply request a disk unit, write a bunch of data on it, and read it back and compare it. It is primarily intended to provide a quick checkout of a new factory formatted disk pack or a disk pack that the CEs have just surface analyzed. In these cases the disk will not have a label so it must first be labeled without setting any flaws. Next a running system is brought up but with the disk pack turned 'LBL' in the EST to prevent any files from being assigned to it. Finally, WRITDSK can be brought up at a control point, the disk turned 'ON', and assigned to WRITDSK. Using this technique you can get ISP to write and read every PRU of the disk except those in the label. Any parity errors will show up in a subsequent DPE analysis following a 'n.DAYFILE,MT'. Once you are satisfied you have found all the flaws, the flaws can be added to the disk label via a FLAW initial deadstart and you are all set. For further details consult section 7.0 or the documentation on L*SYS.
- 2) TMS is similar in theory of operation (getting ISP to dayfile parity error messages). This program is a PP program and must be EDITLIB'ed into the system to use it. TMS can write/read any PRU of any disk, even the labels. It can be used if necessary to examine every PRU of a given RB if all you know is that the RB has a flaw in it. It was originally written before WRITDSK and to perform a similar function, but usually WRITDSK is much faster and safer. To compensate, TMS is much more powerful, although that is usually not needed just for flaw detection. For further details consult section 7.0 or the documentation on L*SYS.

7.0 Utility Program Descriptions

- 7.1 DPE - routine on L*SYS - DPE will list disk parity errors that were extracted from the system dayfile. It can produce a summary as well as a copy of the error messages themselves. In addition it can purge the 'PFERRORS¹' PF if desired. The summary includes the number of times a particular PRU was involved, the EST of the disk drive, the physical address (in decimal) and the logical address (in octal). Note that the RBR number will always be the same for a given disk pack although the EST ordinal can change if the disk packs are shuffled around. The detailed output also contains the date and time of the error, the job sequence number, and various status information. The control card format is:

```
DPE,D=lfm,0=lfm,C,Print,PURGE,Tally=opt,Detail.
```

For complete details use 'HELP,L*SYS,F*DPE'.

- 7.2 FLAW844 - routine on L*SYS - FLAW844 will print selected flaw and/or history information about any or all 844 disk packs for which it has data. Normally a data PF is maintained on disk with all known flaws and history for every 844 disk pack. When new flaws are discovered they can be added to this PF. The data PF format is suitable for easy insertion of information from a surface analysis output (the PF uses hexadecimal numbers, the error codes, etc.). The output will include a count of the number of times each sector appeared in the data, the error type, the physical address (in decimal), the logical address (in octal), and an English description of the error (only for the most common errors). Following all that a summary of the flaws (RB and RBR) that must be set for this disk pack is listed. The control card format is:

```
FLAW844,D=lfm,I=lfm,O=lfm,P=nn=nn=...=nn,  
E=nnnn=nnnn=...=nnnn,F=ON/OFF/PART,H=ON/OFF/PART,  
M=NEW/OLD.
```

For further details use 'HELP,L*SYS,F*FLAW844'.

- 7.3 RBCYL - routine on L*SYS - RBCYL is an interactive program designed to assist in converting logical disk addresses to physical addresses (and vice versa) for all disk types here at MSU. It uses connected I/O and can accept input and produce output in either octal, decimal, or hexadecimal depending on what you want. The program is very self-explanatory.

- 7.4 TMS - data file on L*SYS - TMS is a PP program and as such is kept on a data file on L*SYS. It must be retrieved and then EDITLIB'ed into the system to use it. TMS can read/write any PRU of any RMS (disk) device here at MSU, including the labels and the PFD/RBTC if it a PF device. TMS communicates with the operator via B-display messages, and the operator communicates with TMS via the sense switches and commands issued via 'n.CFOxxx'. There are special precautions in TMS to make sure that you do not accidentally write on a disk label or the PFD/RBTC. TMS can do its I/O with a CM buffer at the control point thus allowing the operator to inspect and/or change the contents of a PRU and write it back out if desired. For further details use *HELP,L*SYS,F*TMS'.
- 7.5 WRITDSK - routine on L*SYS - WRITDSK will request a disk unit assignment and then proceed to write an alternating pattern, concocted from a seed value and the value returned from the system clock, on the file that the operator assigned. After writing a control-card specified number of RBs or when the operator tells it to stop writing, it will rewind the file and read it all back to check for parity errors. WRITDSK does its I/O with the error processing flag set, so it will not abort when an uncorrectable parity error is encountered. When WRITDSK encounters an error in the data pattern it will issue a B-display message and wait for an operator response before continuing. For further details use 'HELP,L*SYS,F*WRITDSK'.
- 7.6 DISKMAP - relocatable subroutine on L*SYS - DISKMAP is a subroutine designed to convert physical disk addresses to logical disk addresses (and vice versa) for all disk types here at MSU. This subroutine is the heart of the RBCYL program. If it is given illegal input it will not blow up but rather return a detailed error status indicating what it did not like about the input. For complete details use 'HELP,L*SYS,F*DISKMAP'.

WRITTEN BY: -^Jerry R. McAllister

APPROVED BY: Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 13.2

What To Do When There Is A Hardware Failure

June 7, 1984

The purpose of this STRAP is to establish some general guidelines for action when the computer or some component fails during systems time or during unattended time and you are the only one here. This STRAP assumes the failure occurs without any operations personnel on duty. If there are operations personnel on duty, let them take over.

Like all guidelines exceptions can easily develop; you are expected to use your best judgement in that situation. The Systems manager (or the designated representative) can usually be contacted for additional input.

Note the definitions (glossary) in Section 5.0.

1.0 Actions after computer component failure

When any component of the computer system fails, follow the posted procedures for notifying the people responsible for maintenance of the component. Also, if it is a reasonable hour and the failure involves a critical component, the OM should be notified.

If the component failure prevents production and the remaining production time is extensive (much longer than it will take to fix the machine) before the next maintenance period, then the CE's may need to be called in to fix the problem. If the failure is with equipment maintained by MSU, the EIC should be contacted. If the failure is another vendor's (CDC, perhaps) responsibility, requesting a service call must be authorized by someone on the authorization list.

For example: If an 885 controller dies, the EIC is notified, but not called in since production can run without the one controller. If a PP stack fails on the 750 and the production period coming is a light one (Sunday morning, for example), then you should run without that PP stack until the OM authorizes calling in the CE's. If all ECS fails at 0800 Saturday morning, then you should get authorization to call in the CE's to fix the problem.

MAF's or EMR's should be written in all cases. These forms are usually on a table by the sliding door. They should be filled out as completely as you can.

If the CE's are not called in, then they should be scheduled for 0600 on the next coverage day to do the repair. For example, a failure on Saturday may not be repaired until Monday, but a failure at 0400 Wednesday, should be scheduled for repair at 0600 Wednesday.

In no case must there be any attempt to repair the component (including resetting circuit breakers or replacing fuses) unless the CE you notified permits it. If you do anything, it may void our maintenance agreements.

2.0 Other equipment failure

If the air conditioning, electrical power, etc. break down, the operations procedures established should be followed. These procedures are usually in the NOTES. If there are no procedures written for the situation, call the people on the list for authorization given above. Non Computer Lab personal may need access to the computer room. You may have to remain to let them in the computer room and stay with them while they make the repair. If you can not be there, then contact someone on the authorization list to obtain staffing. The OM should be informed. Whether to call in the CE's to restart or power up the machine will be determined by the OM or from someone on the authorization list.

3.0 Other considerations

If the system will be down for a while, you should insert the recording in the phone answering machine which will announce this and busy the phone lines. It would be considerate to also set the login message thru the front-end (if possible^, and set the Merit status message. The procedures should be in the NOTES.

Whenever the system will be down for a while, during systems time notify those people that have scheduled time.

If a component fails and will not be fixed, let anyone in systems that may be doing work with that component know about the failure. For example, if a 844 controller goes bad and you know someone was going to work with that controller, call them and tell them of the situation.

4.0 Parties responsible for maintenance

CDC is responsible for maintenance of the 750 subsystem, ECS, disk subsystem, tape subsystem, 512 printers, 405 card readers, 415 card punch and some miscellaneous items.

MSU is responsible for maintenance of the Interdata (Perkin-Elmer) 7/32 communications system, 6500 subsystem, the ConTel network, and the printers connected to the 7/32.

5.0 Definitions and glossary.

Authorization List: A list of people in the Operations group who can authorize emergency maintenance. They can also be called for information on how to handle a certain problem, provide staff, etc. These people are (call

- 1) the first shift assistant manager,
- 2) the operations manager,
- 3) the second shift assistant manager,
- 4) the director of the computer lab.

The first and second shift managers rotate, the current person's name and phone number should be posted. If there is any doubt, you should call the OM.

Call in: Calling the responsible person and asking them to have the problem repaired.

CE's: Customer Engineers: the people responsible for the maintenance of the computer system. (The EIC is the head CE.) Note that the CE's for the CDC system are different from the CE's for the Interdata system or the CE's for the 6500.

EIC: Engineer In Charge: the person in charge of the maintenance service. Each responsible party will have their own EIC.

EMR: Equipment Maintenance Report. This is a report of a malfunction on a piece of equipment filled out for MSU maintained equipment.

MAF: Maintenance Activity Form. This is a report of a malfunction on a piece of equipment filled out for CDC equipment. Some people may refer to these as EOR's.

Maintenance Period: The time when the CE's are scheduled to do preventive maintenance according to the established schedule.

NOTES: These are a series of notes and memos concerning the operation of the computer. They are usually in a card index file kept by the 750 console, or in a binder in the file by the console. If these instructions are inaccurate, then either do the best you can, or call someone on the authorization list if it is important.

Notification; A simple phone call to inform the person of the situation.

OM; Operations Manager: the person incharge of the operation of the computer system.

Production: The time when the published schedule (including revisions) states that the computer is to be available to the user community.

Reasonable hour: Usually 0700 to 2300.

WRITTEN BY: Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 14.3

Protection of Disk Packs

June 6, 1984

The following measures have been implemented to prevent accidental destruction of the contents of 844 disk packs.

1. Each pack has a unique permanent ID of the form PACKnn. This permanent ID is displayed on the pack itself and on its cannister and is used to reference history/ flaw information maintained by the assigned systems person (nn may be 2 or 3 digits).
2. Each pack has a unique VISUAL ID (VID) assigned to it, which reflects the current use and/or owner of the pack. This VID is written into the machine label and is displayed on the cannister if different from the permanent ID (see part 1).

Rules for VID assignment are:

- a) Production packs have a VID of the form SYSnnn. The "nnn" field should match the "nn" field of the PACKnn permanent ID.
- b) Non-removable disks- (like the 885) have a VID of the form SYSnnn, where "nnn" numbers down from 999.
- c) Special use packs have a VID of the form Illnnn, where III = initials of user, and "nnn" matches the "nnn" field of the PACKnn permanent ID. Note that III may not be SYS.

This allows the system to detect non-production packs at deadstart time.

3. The physical external labels on the cannister contain the following information:
 - a) Use and/or owner of the pack.
 - b) Date that these labels were last changed.
 - c) What the pack contains or is used for.
 - d) PACKnn permanent ID, and VID.
 - e) Pack serial number (found on bottom of pack).
 - f) If used in production, a separate sticker indicating which drive the pack is normally mounted on.
 - g) If it is a double density pack, a separate sticker indicating so.
 - h) Any comments you may wish to record.

4. Packs not mounted on a drive are to be in their proper cannisters and put in the cart we have for that purpose. The empty cannisters of mounted packs are to be left on top of the drive on which the pack is mounted for identification purposes.

5. NORMAL or RECOVERY Deadstarts

During the PFD checking phase, the VID from the label of each EST is displayed-

EST	VID
01	SYS999
30	SYS005
31	SYS022
.	.
.	.
.	.

The top line of this display is one of 3 messages:

NORMAL PRODUCTION PACKS
MIXED DEBUG + PRODUCTION PACKS
NON-PRODUCTION DEBUGGING PACKS

If the mixed packs message is chosen, IRCP posts a big warning message and requires the operator to "ACKNOWLEDGE" before continuing.

WRITTEN BY: Jerry R. McAllister

APPROVED BY: Richard R. Moore

Michigan State University

Computer Laboratory

System's Tasks, Responsibilities and Procedures

Number 15.1

Guidelines for Proper Machine Room Conduct

June 7, 1984

The following rules should guide you in your behavior in the computer room. "Production" machine time is whenever the 750 computer is scheduled by operations for use by external (to the Computer Lab) users. "Computer Lab" machine time is when the 750 computer is scheduled for use by the groups in the computer lab (and for preventive maintenance by the CE's). "Systems'" machine time is Computer Lab time scheduled for Systems' use. 6500 usage is mentioned in section 2.0. \

Production machine time is either "attended" (an operator is on duty) or "unattended" (no operator on duty). The published production schedule has both attended and unattended times; however, unattended time may be scheduled without much advance notice (e.g., the only operator in attendance has to go home sick). Scheduling of attended or unattended production is a responsibility of the Operations group; Systems should never change production to attended (and attend the computer) without the prior permission of the Operations Manager.

1.0 General Prohibitions:

1.1 No food, drink, or lit smoking materials should be brought into the computer room or the I/O area. This includes the transporting of these items through the computer room.

1.2 Do not allow unauthorized people in the computer room.

For example, someone knocks on the viewing room door and requests admittance to maintain the DEC computer. During attended production, let one of the operations personnel handle the situation. During unattended production, if there is no pre-authorization, do not allow them to enter.

If you find an unauthorized person in the machine room during unattended production, you should ask them to show that they are authorized or to leave.

It is permissible to give short, escorted tours of the computer area; these must be true tours and demonstrations.

1.3 Do not ask for the computer room door code from any person not in Systems. Do not give out the door code to any one not in Systems. The T.A. or the project leaders are responsible for giving out the door code to people in Systems.

1.4 Do not provide any attended production services. This includes:

- do not mount tapes
- do not file output
- do not print users' output

However, if a user asks for some attended service during unattended production (tape mounting, output, etc.) and if you wish to provide this service, then you may do so. For example, if a remote batch job sits in the input queue because it needs a tape, you should not "automatically" bring the job in (and maybe mount the tape), but if the user specifically asks for this to happen (by the B display message or any other personal means), then you may decide to do this.

If you do provide an "attended production" service, you should be sure that the user realizes this is not a normal occurrence. For example, in response to a request described above, you might reply: "Normally during unattended production time there are no tapes mounted, but since I am mounting one of my tapes, I'll do it."

Output from users' jobs is not to be filed by Systems personnel. During unattended production it is permissible for you to cause Systems output to be printed.

2.0 6500 Usage

Production time on the 6500 is normally from 1230 to 1700 Monday thru Friday. Usually only Computer Lab staff run on the 6500; however, some outside users access some software packages available only on the 6500. All other time is Computer Lab time. During periods when the 750 is not heavily used, it is possible to extend the Computer Lab time.

2.1 6500 Operation During Computer Lab Time

You may use the 6500 to debug your programs that require dedicated access to debug. However, you must not interfere with 750 production. If, for example, you are in the initial stages of debugging new ECS code, you should do this only during 750 Computer Lab time. If your debugging seems to always cause PFD/RBTC checking to be required, then either use the private pack system or wait until 750 Computer Lab time: PFD/RBTC checking causes the 750 system to halt until the checking is finished.

If you need access to a tape unit, you should first get permission from the 750 console operator.

2.2 6500 Operation During Production Time

It is possible to use a nonstandard system on the 6500; however, it must be a working system. It is also permissible to EDITLIB working changes to any system. Of course, there may be bugs in these routines, but you should have debugged them enough to be reasonably certain that they will run correctly. /

3.0 Computer Usage During Systems' Time

3.1 Computer Availability

The computer is never available during Systems' time. LOGIN will prevent user access during regularly scheduled Systems' time. If this is specially scheduled Systems' time, then you must be sure to have the phone lines busy, and batch I/O turned off.

3.2 Beginning of Systems' time

If the production period just before Systems' time is unattended, then you must first dump the dayfile before beginning your work. If this is a regularly scheduled Systems' time, other users will have been logged out. If this is not regularly scheduled Systems' time, then you must proceed thru the shutdown process:

- 1) Shutdown the interactive lines giving at least a 10 minute warning (%SHUTDOWN IN 10).
- 2) Prevent further remote batch submissions (OFFECCR).
- 3) Warn the remote batch stations (L.MSG E** down in 10 minutes).
- 4) After the warning period has expired (after interactive jobs are logged out):
 - a) Prevent batch jobs from entering the pool (JCL,0).
 - b) Turn off all central I/O (OFFIO).
 - c) Turn off remote batch entry (OFFEI2).
 - d) Turn off all front-end printers (%OFF,nn).
 - e) Turn off network batch (OFFNBC).
 - f) If any batch jobs remain executing, decide whether to let them finish or rerun them.
 - g) If any jobs remain printing, decide whether to let them finish or rewind them.
- 5) When everything is quiet, save the dayfile (n.DAYFILE,MT)
- 6) Note the end of production in the machine log.

The Shutdown procedure in the operations NOTES should be consulted for the most current process.

If an operator is on duty, they should idle down the computer.

3.3 Ending Systems' time (or when you are finished with your time).

If there is attended production after Systems' time, then simply turn the machine over to an operator when Systems' time has ended. Otherwise, you must place the machine in an unattended production state. It is important to complete this process at least three minutes before the scheduled start of production.

This unattended production setup must also be done if you are the last person to use the computer even if it is not yet time for production.

To set up unattended production:

- 1) Dump the dayfile unless you are using private packs or have modifications that make dumping the dayfile unsuitable.
- 2) Deadstart the production system unless you are sure you have a production system with no changes running. If you deadstart the system, some of the following steps may already be done. Also, be sure to alter any equipment status as may be noted on the console.
- 3) Set the unattended flag (UNATTENDED).
- 4) Turn on Merit batch processing (ONNBC).
- 5) Turn on remote job processing (ONEI2, ONECR).
- 6) Bring up MANAGER, ARGUS, and NEXT control points (AUTO).
- 7) Login the operator's terminal.
- 8) Be sure that all I/O (front-end and ARGUS) is off except for the self service (source "A") printer and card reader.
- 9) Be sure that the self service I/O devices are on.
- 10) Be sure that all tape EST entries are off (OFF,MT. OFF,NT.).
- 11) Unbusy the phone lines (the switch in the grey box beside the 750 console should be down).
- 12) Set JCL and RDC values (usually, JCL,7777777. RDC,77)
- 13) Note the beginning of production in the machine log. The description should include the system deadstarted and the type of deadstart. Any changes of equipment from the normal should be noted.

After you have placed the computer in unattended production, you should attempt to login to verify that users can access the computer.

3.4 Machine Log

You should enter on the machine log (located on the console) any problems you encounter. Include the time, description, and your initials. You must fill out MAF's for any hardware problems.

3.5 Hardware Problems and Emergencies

See STRAP 13 for what to do on hardware problems. Also, refer to the operations' book of procedures for hardware, air conditioning, etc. problems.

4.0 Machine Room Behavior During Attended Production

4.1 Do not loiter

Do not enter the machine room during production except for a specific purpose. Accomplish that purpose and then leave immediately. Do not "hang around" - you will get in people's way and cause confusion.

4.2 Do not operate the machine.

Unless given specific, prior, approval by the console operator do not enter anything at the console, do not mount or dismount tapes, etc. The console operator is responsible for everything that happens at the console during production so do not interfere.

Similarly, do not retrieve your own output at the printer without prior permission. If you believe it has taken excessively long to separate and file your output, see the shift supervisor. If you feel that the shift supervisor's response is inadequate, then accept whatever the shift supervisor says and discuss it with the Systems manager as soon as possible (no phone calls in the night!).

Other services such as plots, special forms, etc. should be done just as regular users obtain the service. Just as users may request special handling by asking the shift supervisor, so can you.

4.3 When the system crashes

When the system crashes, it is best to let the operators follow their standard procedures; therefore, unless asked to stay, leave the area. If you are asked and can help, then help. If you cannot help find someone who can.

Do not loiter during a crash. The old saying regarding too many cooks is especially appropriate here. Do not come in and ask "What's happening?"; wait until later. If you are not the person assigned to fix crashes, do not assume you can help "unasked." Let the Operations staff complete the recovery.

5.0 Machine Room Behavior During Unattended Production

5.1 Console Operation

It is permissible to perform those operations that will allow you to do your job; however, you must not directly interfere with any users' jobs. For example, you may cause your job to run and assign a tape to it; you must not swap out a user job to allow your job to run faster.

If some abnormal situation is impeding the processing of all jobs, then you may take corrective action. For example: all batch job pool slots are taken by jobs waiting for the same permanent file and the job with the permanent file is a long running job; then you may rerun those batch jobs so that all users can run.

If you decide to print a Systems' job, you must be careful to not print any user's job. This is done (on the front-end printers) by:

- 1) Setting the PRU limits on the printer to zero (%PRU,nn,0)
- 2) Turning the printer on (%ON,nn)
- 3) Forcing the output job to be scheduled (at the 750 console enter NEXTJOB,SBSYSnn,0)
- 4) Turning the printer off when you are finished (%OFF,nn)

Note that this procedure can not be done with ARGUS.

If you mistakenly print a user's job, then place the output on the I/O tables for operations to file.

5.2 System crashes

When the system crashes, you should make your best effort to return it to production as soon as possible. You should perform the normal operational procedures such as taking and processing the dump.